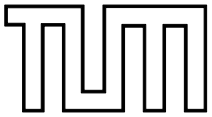


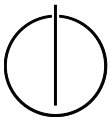
Comparison and evaluation of cross-platform frameworks for the development of mobile business applications

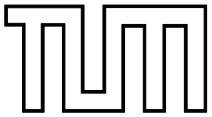
Master's Thesis in Informatik

Fakultät für Informatik
Technische Universität München

Andreas Sommer

Oktober 2012





Comparison and evaluation of cross-platform frameworks for the development of mobile business applications

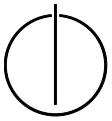
—

Vergleich und Evaluierung plattformunabhängiger Frameworks für die Entwicklung mobiler Business-Anwendungen

Master's Thesis in Informatik

Lehrstuhl für Angewandte Softwaretechnik
Fakultät für Informatik
Technische Universität München

Bearbeiter:	Andreas Sommer
Aufgabensteller:	Prof. Bernd Brügge, Ph.D.
Betreuer:	M.Sc. Stephan Krusche
Abgabedatum:	15.10.2012



Erklärung

Ich versichere hiermit, dass ich diese Master's Thesis selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

I assure the single handed composition of this master's thesis, only supported by declared resources.

Datum, Ort

Unterschrift

Abstract

The recent growth in market share of smartphones brings companies to enter the market of mobile applications. Recent research shows that businesses tend to require mobile applications with shorter development and life cycles in order to improve worker productivity and react to market changes and customer needs. Thus, it is important to develop business applications on mobile platforms with effort and cost efficiency, and to fulfill user experience expectations.

Considering only a single mobile platform, the vendor-supported SDK typically fulfills all requirements for developing an application, such as mature development tools, the ability to use device functionality and well-designed, highly usable user interface components. Native applications often have a uniform look and behavior, deliver high performance and can utilize the capabilities of the underlying device, such as acceleration sensors or a built-in camera. However, the quantity of popular mobile platforms (Android, iOS, BlackBerry, Windows Phone, etc.) suggests that additional effort and resources are necessary if the application is developed separately for each platform, using the native SDKs. This master's thesis compares and evaluates different solutions that allow for a mostly platform-independent development on mobile platforms. Different categories are compared, including support for native device functionality, developer tool support, reliability, UI performance and features, costs, deployment and supportability. As base for the evaluation, a well-defined sample business application is developed with each of the cross-platform solutions and with two native SDKs.

In order to put the results in a meaningful context, the cross-platform solutions are compared with native development based on the same set of criteria. The comparison concludes with an evaluation based on objective scores and subjective experiences. Finally, this thesis gives a conclusion and recommendations, stating whether the cross-platform approach should be preferred to native development, and in which cases it is not worthwhile, regarding aspects of typical business applications.

Durch den steigenden Marktanteil von Smartphones drängen immer mehr Unternehmen auf den Markt der mobilen Anwendungen. Jüngste Studien zeigen, dass Unternehmen häufig mobile Anwendungen mit kürzeren Entwicklungs- und Lebenszyklen benötigen, um die Mitarbeiterproduktivität zu verbessern und auf Marktänderungen und Kundenwünsche einzugehen. Deshalb ist eine zeit- und kosteneffiziente Entwicklung von Business-Anwendungen auf mobilen Plattformen und die Erfüllung von Erwartungen an die Benutzerfreundlichkeit sehr wichtig.

Betrachtet man eine einzelne Plattform, erfüllt das vom Plattformanbieter bereitgestellte SDK üblicherweise alle Anforderungen der Anwendungsentwicklung, wie z.B. ausgereifte Entwicklungswerkzeuge, Unterstützung von Gerätefunktionalität und durchdachte, einfach nutzbare Bedienelemente. Native Anwendungen haben oft ein einheitliches Aussehen und Verhalten, bieten hohe Performanz und können die Möglichkeiten des Geräts, z.B. einen Beschleunigungssensor oder eine eingebaute Kamera, ausnutzen. Jedoch deutet die Anzahl beliebter mobiler Plattformen (Android, iOS, BlackBerry, Windows Phone etc.) darauf hin, dass mit der Anwendungsentwicklung zusätzlicher Aufwand und Ressourcen notwendig sind, wenn jeweils separat mit dem nativen SDK entwickelt wird. Diese Masterarbeit vergleicht und bewertet verschiedene Lösungen, die eine größtenteils plattformunabhängige Entwicklung auf mobilen Plattformen ermöglichen. Verschiedene Kategorien werden bewertet – darunter Unterstützung für Gerätefunktionen, Entwicklungsunterstützung und -werkzeuge, Verlässlichkeit, Performanz und Features der Benutzeroberfläche, Kosten, Auslieferung, Portierbarkeit und Wartbarkeit. Als Grundlage für die Bewertung wird eine wohldefinierte, beispielhafte Business-Anwendung mit jeder plattformunabhängigen Lösung und mit zwei nativen SDKs entwickelt.

Um die Ergebnisse in ein Verhältnis setzen zu können, werden die plattformunabhängigen Lösungen anhand derselben ausgewählten Kriterien mit nativer Entwicklung verglichen. Der Vergleich schließt mit einer Bewertung, der ein objektives Punktesystem und subjektive Erfahrungen zugrunde liegen. Zum Schluss gibt diese Arbeit ein Fazit und Empfehlungen darüber, ob und in welchen Fällen es lohnenswert ist, eine plattformunabhängige Lösung anstatt nativer Entwicklung als Entwicklungsansatz zu verwenden, wenn man typische Aspekte einer Business-Anwendung berücksichtigt.

Table of contents

Abbreviations and glossary

API	*Application Programming Interface*, interface offered by a system (e.g. framework, library, web service) for accessing functions or data of the system. In this document, the term mostly refers to functions or classes offered by a framework.
App	Colloquial short name meaning an application for a mobile device/phone
B2B, B2C, B2E	*Business-to-business*, *business-to-customer*, *business-to-employee*
Continuous delivery	Automation of software release cycle, consisting for example of automated build, tests (unit tests, integration tests, etc.), packaging and release/deployment
CORS	*Cross-origin resource sharing*: With modern browsers, web sites can only make explicit requests to other domains if that server allows such requests. CORS defines headers that can be returned after a HTTP OPTIONS request in order to allow requests from certain or all other domains, for example.
CRUD	Typical operations on data: *create, read, update, delete*.
CSS, CSS3	*Cascading Style Sheets*, a language that enables defining style templates for structured content – typically HTML pages. *CSS3* stands for version 3 of the standard as defined by *W3C*.
DOM	*Document Object Model*, a tree-like structure used for example with HTML documents. Nodes of the tree can be modified using predefined functions (e.g. from JavaScript code).
Framework	See definition on page 13.
HTML, HTML5	*HyperText Markup Language*, a language for presentation of content in a structured way. The term *HTML5* means version 5 but is often associated directly with *CSS3*, JavaScript and additional extensions and APIs like drawing/rendering or data persistence (e.g. *local storage* or *Web SQL databases*) based on standards by organizations like *W3C* or *WHATWG*.
IDE	*Integrated Development Environment*

Internationalization	Internationalized software is architecturally designed in a way that it can be easily translated and changed according to differences between locales/cultures, e.g. number or date formatting.
Inversion of Control	Software architecture in which, for example, the application registers callback functions that get called for certain events, instead of only calling functions of a library directly
JSON	*JavaScript Object Notation*, a compact, human-readable data format often used for data exchange (e.g. with web services)
KB/MB	*Kilobyte*/*megabyte*, note that the definition of 1 KB = 2^{10} bytes and 1 MB = 2^{20} bytes is used
multipart/form-data	Data encoding typically used to transfer binary data and possibly other values of form fields over the HTTP protocol
MVC	*Model-View-Controller* architecture, separates source code or application architecture in three parts: “models” (definition of data model and possibly business logic for operations on the data or database), “views” (presentation of data that is visible and interacts with the user) and “controllers” (manages views and handles user interactions, e.g. switching to another view). Implementations of this pattern can vary and even add more components, such as “stores” (handles storage details, while models only declare data fields and types).
native	Please see page 13 for an explanation of the relevant terms.
NFC	*Near Field Communication*, a standard for wireless transmission of data over short distances, can be used for payments and other purposes and is a built-in functionality of some modern mobile devices
ORM	*Object-relational mapper*, a database abstraction layer that allows to access and change data objects and their properties without the need to program database queries explicitly
OS	Operating system

Pluralization	Method of translating a word or text with a defined quantity, respecting differences between singular and possibly one or more plural forms
ppi	Pixels per inch, the number of pixels within 1 inch (2.54 cm)
REST	*Representational State Transfer*, a loosely defined, HTTP-based and thus stateless paradigm for identifying resources (via URLs) and modifying them (HTTP verbs such as GET, DELETE, PUT are used to determine the action). This paradigm is gaining popularity in the implementation of web services.
SDK	*Software Development Kit*: Set of libraries or framework(s), development tools (e.g. IDE, debugger, compiler, simulator). The purpose of SDKs is to allow creation of software for a certain platform or operating system.
Theme / theming	UI libraries (or similar) with theming support allow to switch certain or all design parameters of displayed components. For example, a different theme can be applied depending on the operating system on which an application is running, in order to have a similar look like other applications on the same system. A theme is a set of values for design parameters (e.g. colors, forms, sizes).
UI	User Interface (refers to a *graphical* user interface in this document)
Virtual machine	Here: Software that interprets a specific type of code (“bytecode”) similar to a hardware processor by executing instructions represented by the code. Virtual machines (VMs) can often be implemented independently of the underlying hardware platform so that the bytecode is interpreted in the same way on multiple hardware architectures. The *Java Virtual Machine* is an example.
W3C	*World Wide Web Consortium*, an organization made up of many member organizations, engaged in the development on standards related to the world wide web, e.g. HTML, XML, CSS
WHATWG	*Web Hypertext Application Technology Working Group*, a group involved in further development of HTML and related standards
$	Refers to *US dollars*

List of tables

List of figures

1 Introduction

The market of handheld computers (here: smartphones and tablets) is a promising and fast-growing market. *Gartner* states that 472 million smartphones and 60 million "media tablets" were sold in 2011. Both markets are expected to grow even more, with a prognosis of 98% increase of tablet sales in 2012 [Gartner2012a] [Gartner2012b]. Combined with opportunities in these emerging markets come changes to the way software is built and released, and various advantages and disadvantages can occur when using mobile applications, for example in an enterprise.

In a 2010 survey by *Forrester Research*, 75% of companies confirm increased productivity of their employees through deployment of mobile applications, enabling faster decision making and problem resolution, better customer experience and other benefits [Forrester2012]. Application lifecycles, as opposed to software for desktop computers, are getting shorter in order to adapt to customer needs and market changes quickly. Some mobile applications are even developed in a matter of days, requiring effective tools for continuous delivery to simplify and accelerate software releases [Gartner2011b].

Currently, an increasing demand for many types of business applications can be observed on the market of mobile applications [Gartner2011b], including applications that primarily target consumers (B2C). Affordable smartphones such as the *Huawei Ideos X3*, which started with a price of around 100€ in Germany, or the *Nexus 7 Android*-based tablet ($299 RRP [1]), might make such devices even more accessible for end users. Online markets for mobile applications are growing as well – the numbers of applications (so-called "apps") offered in *Apple*'s, *Google*'s and *Microsoft*'s app stores increased by +9% (+40000), +11% (+38000) and +64% (+19000) between October 2011 and January 2012 [Chip2011]. Applications of popular online platforms such as *Facebook* or *YouTube*, but also applications with a very specific purpose, such as *Amazon Mobile*, *Dropbox* [2] or *RunKeeper* [3], have been installed up to hundreds of millions of times from the stores.

[1] Recommended retail price
[2] Synchronization and sharing of files between multiple devices
[3] Tracking and timing of sport workouts like running

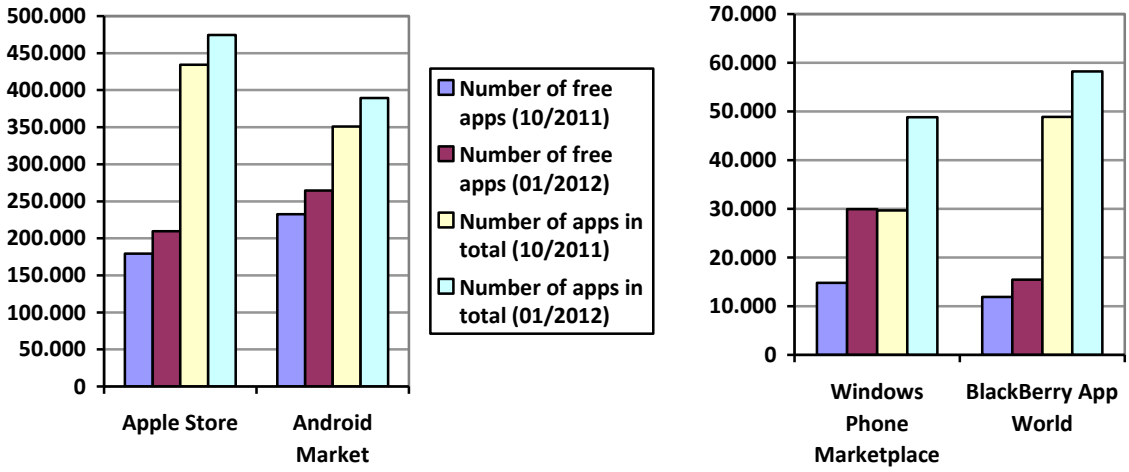


Figure 1: Growth of app stores between 10/2011 and 01/2012 [Chip2011]

All these trends and facts lead to the conclusion that the mobile applications market is a market of the future, and is already a great opportunity for businesses. While the primary goal for a business is of course to improve profit, it is necessary to figure out how this goal can be achieved in terms of efficient development, cost-effective delivery and customer adoption.

Market penetration plays a very important role here – being able to reach more people with an application means more sales or other type of direct or indirect income (e.g. purchases in the application, advertisement), unless the application has a very specific (e.g. device-specific) purpose or the target group is reduced otherwise. Smartphone market share statistics from end of 2011 prove that by only supporting two mobile platforms, namely *iOS* and *Android*, almost 75% of the market is covered [Gartner2012a]. Regarding forecasts, one should also keep an eye on other platforms. For instance, the *Windows Phone* platform is predicted to gain "number 2 rank and more than 20% share in 2015" [IDC2011].

Another factor of success is to keep development costs low. They can be cut down by using fewer resources (e.g. hardware), having cheaper or free software licenses, using efficient tools (e.g. for development and release automation, or development environments) and reducing expenses for personnel – both in terms of timely effort, but also the number of employees involved in application development. Maintenance costs, such as expenses for bug fixing and (professional) support, also have to be considered.

1.1 Motivation

For the development of mobile applications, the costs and expenses named above can already be conquered before and while developing an application, by selecting supported platforms, development tools (which includes licensing), test devices, etc. But also after finishing an application, early project decisions greatly influence further costs. Maintenance costs depend on many factors such as test procedures, documentation quality, adherence to standards and in particular, the size of the source code [vanVliet2008]. The latter, for instance, can be influenced by the number of supported platforms, the programming environment (e.g. tools or the programming language which is used or imposed by the platform SDK), and possibly also the ability of programmers to write lean code.

A business might decide that an application should be developed for multiple platforms, for example iOS and Android in order to allow more customers to use it. In that case, the traditional decision is to develop using the native SDKs of the respective platforms. As mentioned in the introduction, mobile business applications nowadays require short lifecycles. This would essentially mean that an application might get developed by multiple developers for more than one platform at the same time. Many companies, though, do not employ or do not want to afford experts for each platform (see also [Gartner2011b]). *VisionMobile*'s "Developer Economics" report of 2012, in which data from over 1500 developers was gathered, names development and debugging as largest part of an application's development costs (> 55% of the total cost), followed by user experience design (25%) [VisionMobile2012]. It is notable that both points are part of the development for each platform, and as such the actual needed effort (man-hours) might be a multiple of the development time for only a single platform. In my thesis, I also want to find indications that can confirm or disprove this statement. Primarily, though, I want to focus on presenting which promising solutions already exist for cross-platform development, how they compare with one another and how they perform in comparison with vendor-supplied native SDKs. The main criteria for the comparison are features of these solutions (accessing platform and user interface features), performance, reliability, developer support and other criteria like costs and professional support options.

Software frameworks offer basic building blocks and often a predefined architecture that can be used or adapted to build an application. I want to investigate to what extent modern cross-platform development frameworks for mobile devices can help save costs by reducing factors like ease and speed of development, need for expensive licenses, resources and other expenses. As denoted by the criteria above, apart from costs, functionality and technical support for developers plays a major role in my comparison, as well. In the end, I want to draw a conclusion, giving recommendations about the quality, readiness and usefulness of the cross-platform approach as compared with native development.

In a survey of 2010 [vdHelm2010] [4], more than 50% of the participants stated they develop a mobile application individually for each operating system (simultaneously or by porting a finished application to other platforms). Although the survey is not representative, it shows that awareness and usage of cross-platform frameworks is not very high, making it worthwhile to research whether there are reasons for companies to use them.

1.2 Reappearance of an old problem on personal computers

Similar to the rise of mobile applications and increasing popularity of different mobile platforms, the market of personal computers saw a heterogenization of operating systems. Cross-platform development for personal computers never gained high commercial interest, i.e. there are very few vendors developing and offering paid cross-platform solutions, and many commercial software products still only work on a certain family of operating systems. On the other hand, several open-source solutions are available (see below). Developing for multiple operating systems can be cumbersome since the underlying principles such as file system layout, paths, graphics and other hardware APIs differ between them. Various solutions exist to abstract these differences and enable developers to support multiple platforms at once without having to change a significant part of the source code:

- Programming languages shipping with a standard library that abstracts OS differences (e.g. *Java*, *Python*), often executed by a virtual machine
- External libraries with the same purpose (e.g. *libapr*)
- UI frameworks that either reuse native controls (e.g. *wxWidgets*) or display their own type of controls (e.g. *Qt*, *GTK+*)

[4] Note that most participants were based in Germany

- Runtime environments acting as container and executor of cross-platform applications (e.g. *Adobe AIR*, *Adobe Flash*, *Microsoft Silverlight* / *Moonlight*)
- Web-based solutions which are either deployed as web sites or in an application container that displays the web site locally (so-called “site-specific browser”)

The figure below depicts a taxonomy of cross-platform solutions for personal computers (i.e. desktops and laptops), categorized by differences in their features, not their exact architecture. Each of the examples belongs to one of the types mentioned above.

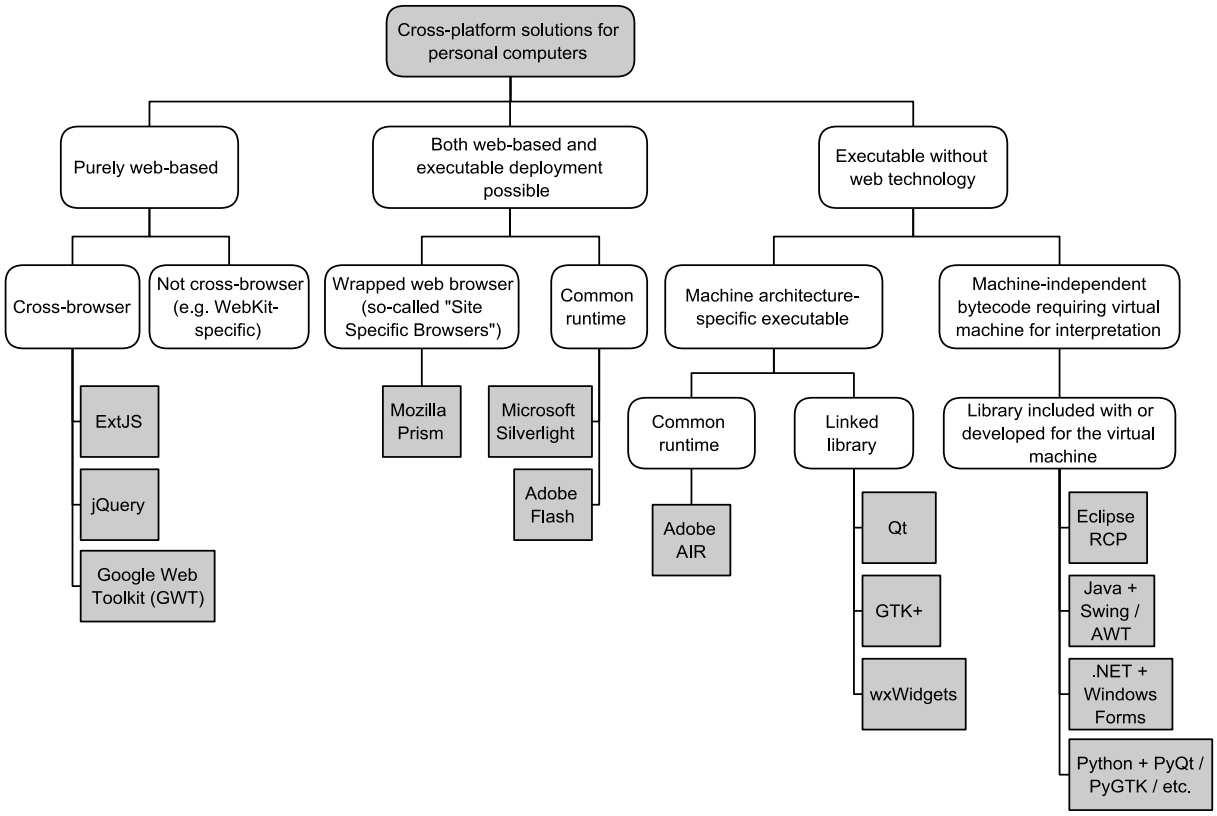


Figure 2: Taxonomy of cross-platform solutions for desktop computers

Developing for multiple platforms is not a new problem. For instance, the *Swing* framework (part of Java’s standard library since early versions), was released in 1997. *GTK+*, a cross-platform UI framework primarily used in a number of Linux distributions, had its first stable version in the year 1998. Both these and other emerging frameworks were developed further and provide good ways for developing software across different platforms. Nevertheless it is important to decide for the right framework depending on the software’s purpose and requirements, which holds true for mobile applications as well. For example, customizability of the user interface should be considered – *wxWidgets*, which uses native UI

components (e.g. buttons, lists), cannot provide as much customization as frameworks that render user interface components themselves. The same consideration is necessary for mobile applications, where an application could follow the operating system's typical design or is customized to the needs of the application.

1.3 Mobile platforms and cross-platform development

Bishop and *Horspool* categorize the term *platform* as a constraint of programming language, operating system or computer (meaning the hardware) [Bishop2006]. For the purpose of my work, I first want to define the term *mobile platform* in detail because its meaning is vital for the understanding of the remaining chapters.

A huge number of different types of mobile devices exist, some with special purposes and some (like smartphones) with a more generic purpose. For example:

- Pure GPS (recording) devices
- GPS-based navigation devices
- Data recorders (heartbeat, blood pressure, running routes)
- Digital compasses
- Digital cameras for photographs and videos
- Calculators
- Portable music players
- Mobile phones (main features: calls and text messages over cellular networks)
- Smartphones (mobile phone that "facilitates access for surfing the web, e-mails and can install and run applications" [Busk2011])
- Tablet computers

Although some of the first eight device types can be programmed (e.g. calculators), they are not capable of running complex, arbitrary software and have high restrictions in computing power, screen size and peripherals – for example, a calculator typically cannot access the Internet. In contrast, smartphones and tablet computers allow many types of software applications to be run (the number of available apps for iOS and Android proves this), they can have the computing power of an average personal computer and screens range from small to large and nowadays offer high quality and high density displays. Furthermore, such modern devices feature a range of helpful built-in sensors and actuators, allowing for a

variety of different applications. Vendors like Apple offer SDKs to create software that can be run on certain devices. Together, the hardware (system plus any built-in, sometimes optional sensors and actuators like GPS or an accelerometer), the operating system and the vendor-provided software SDK and standard libraries thus offer an *application platform*, that is, the base for building software on said mobile devices, without restricting applications to any specific purpose (e.g. navigation).

In the following, the term *mobile devices* shall mean smartphones, tablet computers and similar mobile (i.e. handheld) devices that offer an application platform. A *mobile platform* is defined as the application platform for a certain type or family of mobile device(s) – since only modern phones and tablets are considered, this application platform must support the creation of user interfaces, access to peripherals of the device (sensors and actuators) and, if applicable, features of the underlying operating system, such as file or database access. Vendors may apply restrictions to a platform like minimum hardware requirements.
Note that laptop computers are not considered as mobile devices here because they do not expose extra features as the above device types do (such as a GPS sensors or high-resolution camera), and laptops are typically too large to be “mobile” in the sense of “handheld”.

An example for a mobile platform would be *Android*, which was released in different versions. For instance, an Android 2.3 device must have a screen with at least 2.5 inches diagonal size and at least a resolution of 100 pixels per inch [Google2010a]. The Android SDK is freely available, and leverages a subset of the Java programming language to create applications for devices running the Android operating system. The software API includes functionality to access peripherals like the camera or GPS location.

Just like cross-platform solutions for personal computers presented before, solutions for mobile platforms can be categorized by their features. An overview is depicted in figure 3 below, and will be detailed in chapter 2, together with different types of framework architectures.

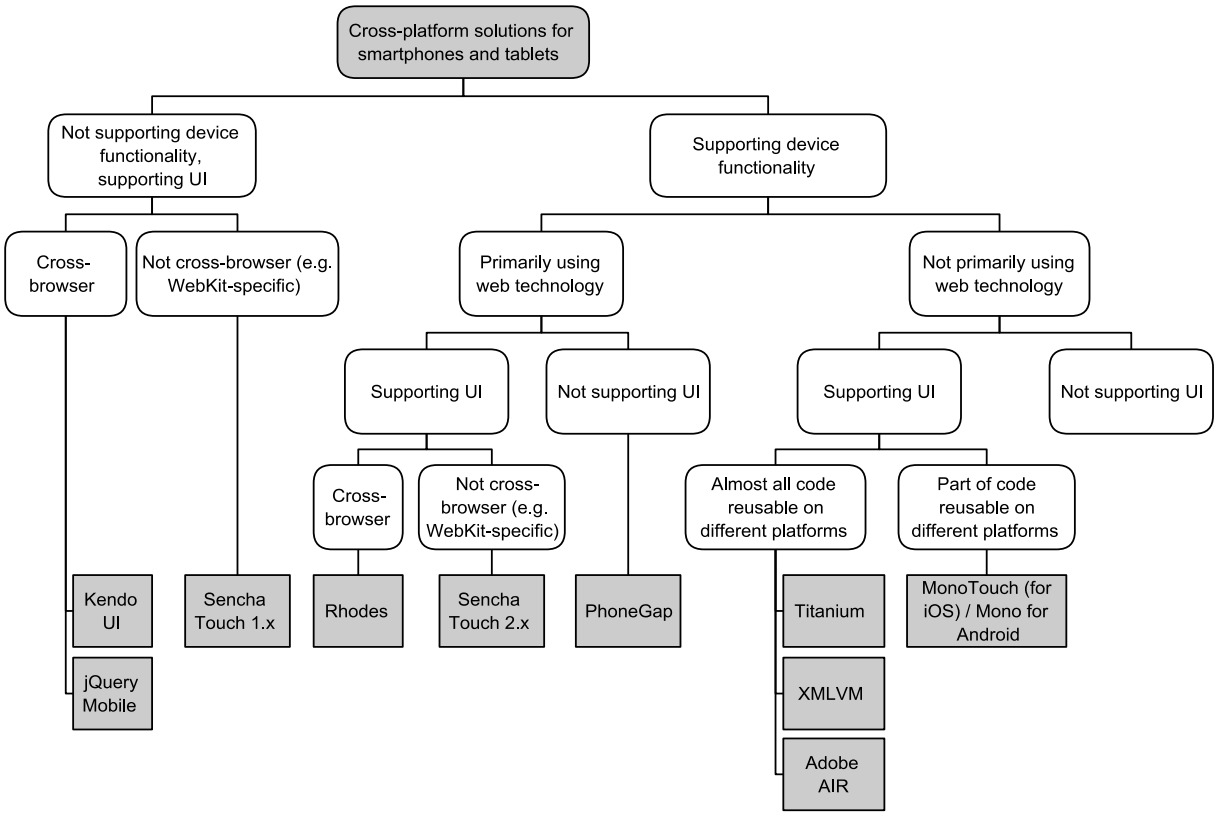


Figure 3: Taxonomy of cross-platform solutions for mobile platforms

A mobile application framework can provide the basis for multiple aspects of an application. Amongst others, the following technical aspects can often be found:

- **Project scaffolding**

  The framework or a supporting IDE plugin creates required project and configuration files, leaving the developer with the task of filling in additional information.

- **Support for device and OS functionality**

  Access to hardware functionality (e.g. camera), contacts, file system, location, etc.

- **User interface design and support**

  Support in UI layout and design can come in many forms. Built-in components (e.g. buttons, tab bar), a visual editor or file-based styling (such as CSS) are possible features.

- **Packaging and deployment**

  Additional tools may allow packaging an application for deployment on a device or emulator, or might allow preparing an application for submission to an app store.

Above features are supported by native SDKs as well, i.e. the collection of libraries, tools and other resources provided by a vendor of a mobile platform such as *Google* (Android platform) or *Apple* (iOS platform). For example, Apple provides the *Xcode* IDE which supports interactive user interface design, an iOS simulator, compiler, debugger, etc. However, such an SDK can only be used for one platform.

Cross-platform frameworks, on the other hand, support more than one platform. This does not necessarily mean that the very same code base and tools can be used to develop on all supported platforms – in most cases, for instance, Apple's Xcode IDE is needed for iOS application development. Yet these frameworks provide means to share a common part of the source code. For example, an Android project consists of the settings file `AndroidManifest.xml` and an application entry point in a Java source file. A cross-platform mobile framework might provide this platform-specific entry point which in turn runs the platform-agnostic code. This is only part of what a framework can provide in order to support several platforms. In fact, most frameworks offer more functionality, like access to the device hardware (e.g. camera) or native APIs (e.g. contacts, location, sending SMS), or user interface support. Moreover, frameworks can differ greatly in the approach of running the application on multiple platforms – for example, it is possible to execute or interpret code in a common programming language, or instead build a web application, using existing technologies like HTML5 and JavaScript that are already supported by many devices. The different approaches are presented in chapter 2.

The main motivation of this thesis is that cross-platform frameworks, compared to native development, can potentially reduce effort and costs as they might lead to smaller (more easily maintainable) source code, higher market penetration with similar effort (because multiple platforms are supported at once), and probably have other advantages. The evaluation of frameworks in chapter 6 shows whether such frameworks can really provide advantages. The above arguments are taken into consideration and measured against what native SDKs support.

1.4 Related work and topic distinction

In April 2011, *Timo Paananen* of the Finnish *JAMK University of Applied Sciences* completed a bachelor's thesis [Paananen2011] that compares several frameworks with the goal to find the optimal solution for a client, a company that is focused on mobile and web development. Other than reasoning about the specific decision for the *Titanium* framework, his work gives general recommendations about which framework is preferred in which situation and what advantages and disadvantages arise from using the non-native development approach.

The frameworks *Titanium*, *PhoneGap*, *Rhodes*, *Adobe AIR* and *OpenPlug ELIPS Studio* were compared. As of the end of 2011, future development of OpenPlug has been stopped by *Alcatel-Lucent*, which leaves four worthwhile frameworks that have been further developed since the conduct of the comparison in the beginning of 2011. For example, *Appcelerator*, the company behind the Titanium framework, acquired *Aptana Inc.* and incorporated their *Eclipse*-based IDE, *Aptana Studio*, as a new supported development environment for Titanium applications, allowing for interactive debugging, the well-known Eclipse environment and other benefits.

This master's thesis compares a similar selection of frameworks, but includes newer versions of the already compared frameworks, and adds other frameworks for consideration. Some projects with new development approaches are considered, for instance (see framework types, p. 15). As an example, new UI frameworks have been developed that simplify native-looking designs of applications for different mobile operating systems – HTML-based development frameworks such as PhoneGap can now be combined with the UI styling capabilities of these UI frameworks.

Regarding methodology, Paananen conducted a comparison based on a demo application whose purpose is to show contacts using the native contacts API. This implicitly includes work on project setup, UI design and of course writing the source code to access contacts, and is thus a reasonable comparison base. Nevertheless this kind of application does not represent a typical, complete business application, for example applications that interact with data or the web (or web services) in some way. In this thesis, one part of the framework

comparison is based on a well-defined application that more closely approximates typical data-driven business applications.

The most distinguishing characteristic of my comparison is the incorporation of native SDKs instead of only studying the features of cross-platform frameworks in comparison with one another. Evaluating cross-platform and native solutions under the same criteria allows me to figure out whether the still very young cross-platform frameworks can be considered ready for production use and can compete in different categories with the vendor-provided SDKs.

Another important point to mention is that the objective scoring model in the bachelor's thesis weighs each feature the same and sums up to a total score. Other properties, such as documentation, license or platform support are weighed the same. In the next section, I will shortly outline my methodology and further distinguish between Paananen's work and this thesis (methodology detailed in chapter 3).

In June 2011, *Gartner* released a research note to help developers and decision makers select the right cross-platform "tool" for mobile applications based on four considerations [Gartner2011a]:

- Commercial issues (e.g. vendor viability, affordability of tool costs and license)
- Functional issues (e.g. support of required platforms and their features, developer support features)
- Skills and staffing (e.g. do the tool's required skills match available developers?, available training)
- Risk management issues (e.g. professional support, sample code, exit strategy if vendor becomes unviable)

Many of the criteria named above are also considered in my evaluation. However, the goal of my work is to provide recommendations for deciding whether to use a cross-platform solution *at all* and to give hints about which of the selected platforms is a good choice for particular types of applications.

1.5 Proceeding

In the next chapter, I will distinguish and present the different types of cross-platform frameworks. Afterwards, I select three cross-platform solutions and compare them with two native development environments (Android and iPhone) in chapter 3. The comparison and evaluation will focus on requirements from different points of view, ranging from functionality and usability to application performance and stability, developer support (tools and professional support), costs and other criteria. I evaluate these categories using a scoring system and assign appropriate weights to each category. Chapter 3 also details on the exact methodology and explains which categories are evaluated. Next, chapter 4 compares the architecture, capabilities and procedure of installation and development for the cross-platform frameworks and native SDKs. Scores in the evaluation are partly based on findings during the implementation of a sample business application (defined in chapter 5) with each of the selected frameworks, and also on known facts about a framework or its vendor. The five solutions (three cross-platform, two native) are then evaluated in chapter 6 using the given categories. Finally, the usefulness of cross-platform solutions is compared to native SDKs in the conclusion (chapter 7), and recommendations are given as to whether adopting a cross-platform framework for development has advantages over native development, and in which cases.

2 Cross-platform and native mobile development

This chapter explains necessary terms and gives an introduction to what cross-platform frameworks are and which potential advantages they offer. Afterwards, in section 2.3, I describe the different types of frameworks with their approach and examples.

2.1 Frameworks

The term *framework* in the context of software development can be defined as follows: "Software frameworks are a software reuse technology that fosters the reuse of entire architectures (as opposed to code fragments) for applications within a certain domain. The framework defines a generic architecture that can be adapted to rapidly instantiate a particular application within that domain. A software framework offers ready-to-use components that encapsulate recurring abstractions within the domain." "It also defines the parts of itself that must be adapted to achieve a specific functionality." ([Pasetti2002] and [Buschmann1996]) Other definitions point out the flow of control in frameworks, which usually follows the *Inversion of Control* principle [5], as opposed to *libraries* [Johnson1988].

These definitions can be applied to the development of mobile applications in the same way. Note that hereinafter, the term *framework* is meant to also comprise accompanying tools, such as helper scripts, a development environment (IDE), an emulator and the like. SDKs provided by vendors of mobile platforms can also mostly be considered frameworks but are often called *native SDKs* in the following for better distinction.

2.2 Cross-platform mobile frameworks and potential advantages

Herein, *native development* for a platform shall mean the use of a usually vendor-provided SDK that is designed for a given platform and usually platform-dependent. *Native applications* is a vague term that is defined in many different ways in literature and by framework vendors. Native applications shall be defined here as applications run by the primary execution environment of a platform, for example executed by the device's processor or a virtual machine. As an example, Google provides the Android SDK, which includes the Android-specific Java-based APIs and tools to compile Java code to platform-

[5] See glossary, example: software registers callback functions with a framework that are triggered on certain events, while library functions are only called directly

specific bytecode since Android offers a virtual machine that differs from a regular Java virtual machine. An application running on that VM is considered a native application [6].

Cross-platform frameworks are frameworks that support multiple platforms, with the same or similar effort involved to create an application on potentially more than one platform *at once* (or *porting* an application to other platforms with very little effort), as compared to creating it for only one platform with the native SDK. This essentially requires that a framework has to provide means to reuse parts of the architecture and source code that are platform-independent.

Potential advantages of cross-platform frameworks for mobile platforms may be that they reduce costs and effort because, compared to native SDKs, they can offer

- **Often a single programming language or a combination of well-known technologies**
  Developers do not need to know several languages, like when developing on each platform separately (iOS: *Objective-C*, Android: *Java*, Windows Phone 7: *C# / .NET*, and so on). They only need to specialize in one language or a small set of technologies, which might result in fewer developers being necessary.
- **Common code base**
  Significant parts of the written code can be reused for all platforms supported by a framework. This means less time has to be invested in the actual programming (if multiple platforms are targeted), and as result probably the need for fewer developers. Less code has to be written and maintained overall, which can greatly reduce complexity and improve the process of finding bugs and correcting them – they only have to be fixed in one place instead of having to reproduce the fix for each platform.
- **High market penetration**
  Frameworks often support four or more popular mobile platforms. Market shares of 2011 show that supporting the four major platforms means that 92% or more of the

[6] Note that a *Native Development Kit* (*NDK*) exists and allows writing an application partially or entirely in C/C++ (https://developer.android.com/tools/sdk/ndk/index.html, accessed: 2012-10-10). Such an application is also considered native. Since using the NDK mostly does not make sense for business applications (it is worthwhile for games, for example) and C/C++ only applications cannot use all Android features without calling the Java API, Android applications based on Java are also considered native.

market is covered ([Gartner2012a] and [IDC2011]). A forecast for 2015 shows that the four major platforms might cover up to 95% of the market, although it is necessary to mention that between 2011 and 2015, the *Windows Phone* platform is predicted to become one of these four platforms, while *Symbian* might lose most of its market share [IDC2011]. Thus, the decision for supported platforms, and as such also the choice for a framework, must be taken carefully and with foresight. As a potential conclusion, frameworks might provide more market penetration, i.e. more supported platforms, possibly with little additional effort.

- **Version independence**

  Many mobile platforms receive regular updates that improve or add features, fix problems or make other minor or major changes. An abstraction layer, like cross-platform frameworks provide, can help cope with different behavior or feature sets in the different platform versions.

2.3 Types of cross-platform frameworks

Different types of frameworks exist with the purpose of cross-platform support. They can be distinguished by their purpose, kind of UI styling (if applicable) and general architecture.

The purpose of a framework can vary: frameworks can include support for building complete applications which mainly includes application logic, user interface and deployment. Some frameworks try to support all of these points, while others are more specialized. For example, there are multiple frameworks that only allow user interface creation, e.g. *jQuery Mobile*, while others explicitly do not include UI creation and styling, e.g. the *PhoneGap* framework.

If a framework supports user interface components and styling, there are two kinds of designs it can support: either it uses native components (or tries to imitate them), blending well with the operating system and other applications on each platform, or strives for a custom and/or uniform look on all platforms. In the first case, though, even if a native look is the goal, a framework might not provide themes for all supported platforms.

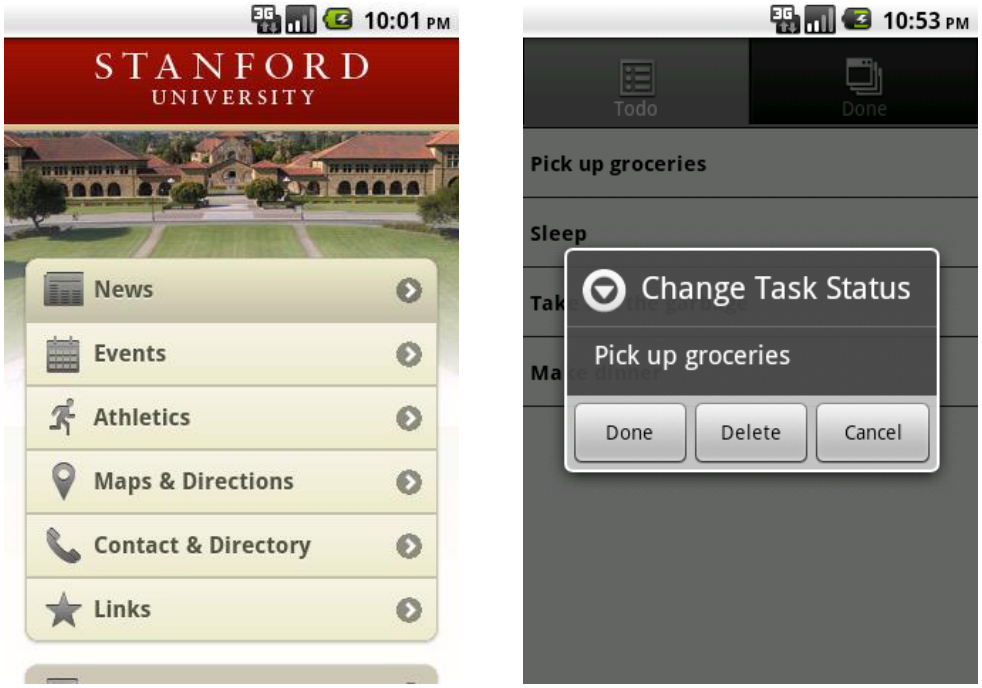


Figure 4: Stanford University's mobile web site (using jQuery Mobile), demo application (using Titanium)

The above figure on the left side shows a web application in Android's browser – without the browsers navigation bar. The *Stanford University* web site [7] uses *jQuery Mobile*, a JavaScript-based framework, with a custom theme in order to give the application a uniform look on all browsers (although the framework supports custom theming and there even exists an iPhone-like theme). On the right side, a demo application [8] of the *Titanium* framework shows that native controls of the Android platform (e.g. the tab bar and a dialog box) are used.

The different types of architectures are explained in the following subsections through descriptions of the approach and types of applications supported by the frameworks. Also, a distinction between pure web applications and mobile applications is made, which is important to decide whether a framework can be called cross-platform *mobile* framework (which are considered in this thesis) instead of only being a *web* framework.

2.3.1 Purely web-based applications and distinction to mobile applications

Pure web frameworks for mobile devices are based on display of HTML, often with included JavaScript, CSS and image files for building application logic, UI styling and effects. Such

[7] http://m.stanford.edu/, accessed: 2012-03-17
[8] https://github.com/appcelerator-developer-relations/Sample.Todo, accessed: 2012-03-17

frameworks typically focus on making applications function on many platforms' browsers or web views (an embeddable UI component that renders HTML, available on many platforms).

The terms *mobile application* (informal: "app") and *web applications* must be distinguished. Web applications are just plain web sites. They can be packaged in a mobile application using a platform's web view capability in order to display them in fullscreen mode or without the browser controls. However, without further extensions, web applications can only access the capabilities of modern web technologies – comprising HTML5, CSS3, JavaScript and extensions that were standardized over the last years, e.g. *W3C/WHATWG* standards like detecting the geographic location of the device, persistent storage using a key-value store or improved 2D drawing (of which some need rights granted by the user when running directly in the browser, e.g. access to the location).

In contrast, *mobile applications* can also access the various features of mobile devices. This includes hardware features like the camera, accelerometer or active GPS usage, data APIs such as access to contacts, the file system or native database implementations, and other features like notifications. Almost all of these features cannot be used using web technologies alone. Nevertheless web frameworks for mobile devices can augment the use of other frameworks, as will be explained in the next subsection about hybrid applications. The other major difference to web applications is that mobile applications are actually installable on mobile devices. Normally, a special package format is used for that purpose in order to put the whole application into one file. This file can be deployed to app stores so that users can easily retrieve the application.

Examples:

- **jQuery Mobile**

  Based on *jQuery* and *jQuery UI*, two related JavaScript libraries, this framework provides a unified user interface especially designed for mobile devices which have different requirements (touch control, small screens). Asynchronous page loading and page transitions are other features. Figure 4 above depicts a screenshot, showing the non-native look.

- **jQTouch**

  Implemented as a plugin for the libraries *jQuery/Zepto*, this framework is focused on UI layout, design (using CSS) and animations on *WebKit*-based browsers, such as on iPhone or Android devices. It also supports page history management. A native-looking theme for iPhone is provided, but not for other platforms.

2.3.2 Hybrid applications primarily using web technology

Because of the previously mentioned shortcomings of solely web-based frameworks in the context of mobile applications, they can be combined with the native functionality of mobile devices. Hybrid application frameworks based on web technology thus support native features and use HTML to actually display the user interface. Existing frameworks use JavaScript as the frontend for calling native functionality, bridging between JavaScript function calls and native APIs.

There are frameworks that only support calling native functionality but do not provide the means for user interface layout or design. For instance, the *PhoneGap* framework offers many native functions, but only offers to use the platform's web view component in order to display HTML-based user interfaces – no UI functionality of any kind is included. Therefore it should be mentioned that such specialized frameworks can be combined with UI-focused frameworks, like the examples of web application frameworks mentioned above.

Examples:

- **PhoneGap** [9]

  PhoneGap uses a platform's web view component to show user interfaces built with HTML and other web technologies, but as mentioned above it does not provide UI styling in any way. Instead, this framework is focused on reducing differences between platforms by offering a uniform JavaScript API for native functionality such as camera, events, contacts, files, notifications and others [PhoneGap2012]. Applications created with this platform can be packaged for app stores – this is also done differently on each platform, by integrating PhoneGap files in a regular application project. This means that for an iOS application, Xcode and as such an Apple computer is necessary.

[9] Base of the project renamed and migrated to an Apache project called "Cordova" (regarding the naming, see http://phonegap.com/2012/03/19/phonegap-cordova-and-what's-in-a-name/, accessed: 2012-03-25)

- **Sencha Touch 2**

  As opposed to PhoneGap, a native package for the supported platforms can be built on *Windows* and *Mac OS X* systems. If the supplementary SDK tools are installed, application packages deployable on Android and iOS can be created. This was introduced in version 2 of the framework, which also added a set of APIs for accessing native functionality (camera, native confirmation dialogs, etc.) through JavaScript [Sencha2012]. Adding own native code is not supported with the SDK's packaging mechanism – in order to achieve that, Sencha Touch can be combined with other frameworks for packaging purposes (e.g. PhoneGap). Primarily, Sencha Touch provides a purely JavaScript-based application structure, generating an HTML user interface from declarations of UI components in the source code.

2.3.3 Hybrid applications with compiled or interpreted code

Apart from frameworks that use the web view's JavaScript engine to run application code, there are frameworks that interpret code themselves. This can be again in JavaScript, but also other programming languages. Thus, the program logic is not executed in a browser environment, but in an interpreter instead. Such frameworks may still offer the use of web technologies, for example to design or program the user interface.

Examples:

- **Titanium**

  This framework, like many others, makes use of JavaScript as development language. The difference is that Titanium puts the JavaScript source code of an application together with a JavaScript interpreter in order to create the full application package. The JavaScript engine built into the platform's web view component does not get used for executing application logic.

  Titanium supports web views for displaying HTML, but the main focus lies in providing native user interface components that are created directly in the source code. The framework offers a uniform JavaScript API for the user interface and handles the specifics of creating and controlling the native UI components [Appcelerator2011].

- **Rhodes**

  Rhodes incorporates a Model-View-Controller (MVC) approach directly in the framework. It uses *Ruby* as programming language for program logic, i.e. models and controllers. User interfaces (the views) are written using web technologies (HTML/CSS/JavaScript), but HTML templates can be used to insert variables or run simple view logic (templates inspired by the web framework Ruby on Rails, similar to JSP files) [Allen2010]. Few native components, like a tab bar, are supported as well.

2.3.4 Other types

Several frameworks exist that do not match the types outlined above. Some examples are mentioned here:

- **XMLVM**

  This is a research project with the goal of translating programming languages into each other. One example utilization is the cross-compilation of an Android application to a native iPhone application by translating its source code and adding a compatibility library that mimics the Android API on the iOS platform [XMLVM]. Therefore, XMLVM provides a very different approach of reusing source code and thus also represents a framework.

- **MoSync**

  MoSync allows developers to write a mobile application with HTML (native UI and device functionality available through bridging of JavaScript calls) or C++. In the latter case, the UI can be created with HTML or alternatively with C++ which again offers two alternatives: using MoSync's own UI framework results in identical look and feel on all supported platforms, while using native UI components is supported as well [MoSync2012]. Altogether, MoSync tries to provide developers with more alternatives and thus belongs to more than one framework category.

- **mobl**

  The programming language *mobl* is designed specifically for mobile applications. It integrates application logic, user interface and data modeling in this one language, however direct use of HTML is possible as well. Out of the source code of an application, the build process generates an HTML-based application with the application logic transformed to JavaScript code [mobl2011].

3 Comparison procedure

In this chapter, I first define what *business applications* and in particular *mobile* business applications are, and describe their common aspects, requirements and restrictions. Section 3.2 details on the methodology used for the comparison and subsequent evaluation of the frameworks. Evaluation criteria are presented and differentiated into five categories. I shortly describe the scoring schema used to assign points for each category, and associate each category with a weight. Finally, frameworks that were selected for comparison are listed and I state reasons and considerations as to why those frameworks were chosen over others.

3.1 Mobile business applications

The comparison of the frameworks is based on aspects of typical business applications. Therefore, the term *(mobile) business applications* and characteristics of such applications must be defined.

According to *von der Neyen*, the term *mobile business application* describes the expansion of business processes by incorporating mobile devices, and thus gaining added value. Such an application can be both for internal and external purposes (staff and customers). A major item that pertains to the term is the exchange of information in order to achieve new purposes or benefits. *Mobile* applications allow location-based and thus personalized services, for instance. Cost reduction and prestige are two reasons why a company would introduce such applications [vdNeyen2011].

Buse defines mobile business as the communication and exchange of information, goods and services via mobile devices. He also mentions that applications can be both internal (B2E) and external (B2B and B2C). The advantages of mobile applications are extended by context-specificity: an application can show different options or offers, based on location (e.g. advertisements), time (e.g. events), user actions and interests (e.g. events of a certain kind). Furthermore, mobile devices are cheaper than personal computers and they are easier to learn [Buse2002].

Hereinafter, a business application is defined as any application that supports a business's commerce or a business process, whether it may be in a B2B, B2C or B2E relation. It is important to note that games and similar applications do not belong to the category of business applications as regarded in this thesis.

Apart from the definition, it is essential to know common features, aspects and requirements of mobile business applications. Depending on the purpose (B2B, B2C, B2E), these might differ. The following aspects are often found in business applications (partially taken from [Anderson2012]):

- User interface providing a simple, structured way of entering and modifying data
- Data validation
- Integration with web services, central data sources, possibly legacy systems
- Data synchronization
- Retrieval of information about products or services
- Authentication and authorization
- Security is crucial because confidential or private information might be handled (e.g. commercial data like orders, customer data, stored login data)
- Accounting and payments of products and services
- Require long-term support and maintenance (e.g. updates)
- User interface layout and styling adapted to corporate identity

Note that not all of these aspects apply to every business application. It greatly depends on the requirements, purpose and complexity of the application.

A *mobile* business application may share the same aspects, but has several differences in the sense of additionally usable features:

- Additional device features can be used, e.g. GPS-based location for personalized advertisement or simplified data input by scanning a product's barcode or QR code with a built-in camera.
- An application can ask for additional permissions to access personal or other helpful data, such as contacts, mails, captured pictures or videos, etc.

- Mobile devices are typically always powered on, allowing background services to run at arbitrary times, for example to receive data updates or notifications for the user.
- New ways of payment, e.g. in-app payment or using *NFC*, are other features that could be useful for certain types of applications.

However, further restrictions may apply as well:

- The often smaller display and resolution force a well-designed, compact and simple user interface.
- In the case of external customers (B2B, B2C), i.e. if the application cannot be pre-installed, an easy installation is required because complex installation and configuration procedures impairs user experience as they can keep users from actually starting to use the application.
- User interfaces' control flow is different because mobile devices are often touch-controlled or have a smaller set of keys as compared to desktop or laptop computers. Complex actions by the user should thus be avoided.

3.2 Methodology

In this section, I will describe how the selected frameworks (see section 3.3) and the two native SDKs (for Android and iOS) are compared and later evaluated with a scoring system.

The procedure for the comparison and evaluation starts with the implementation of a well-defined sample business application with each of the frameworks and native SDKs. With the experiences gained during the implementation, the quality of the resulting applications and information not related to the sample application, each framework/SDK is assigned a score in five main categories. Each category comprises several criteria that are considered for the score. The final score is determined by summing up the category scores, weighted by the importance of each category.

3.2.1 Criteria

Various detailed criteria could be applied in a comparison of cross-platform frameworks. In order to keep a clear overview, all considered points were categorized roughly after the *FURPS+* model for software requirements as defined in [Brügge2004]. This model applies to software products, not specifically to software frameworks, which support developers to

create and deploy software (amongst other advantages). Therefore, the categories were slightly changed. For example, the “U” in FURPS stands for usability, including the ease of learning a system and its documentation. However, the documentation of a framework supports developers who use the framework, while the user experience criterion targets users of the finished application created with the framework, but can as well mean the usability of developer tools (e.g. easy debugging). This ambiguity was addressed with two separate categories for usability features (support for creating a usable interface for end users) and developer support (tools available to developers who use the framework).

The five chosen categories and their counterpart in the FURPS+ model are explained below. An overview including considered subcategories can be found in table 1 (p. 31).

Functionality

This category typically comprises all functional requirements of a software product. For cross-platform frameworks, the available variety and quality of features are considered in this category.

Mobile platforms often include APIs for access to a database (e.g. SQLite), the device state (e.g. network, location) or features that are very specific to the platform’s operating system (application events like pause/resume) or type of devices (hardware buttons such as a menu or back button). The former examples could be abstracted by a common interface in a cross-platform solution, while the latter features might be too specific. Even if some features of a mobile platform cannot be abstracted, a framework might provide a way to access them – one approach would be to allow developers to write and run native code if they need any platform-specific features. Another approach is to include an interface for these features which only functions on the relevant platform. These “native” features are one part of what a framework may support, and can be put in several subcategories:

- Location and network
- Data layer or database access
- Hardware sensors & actuators

- Buttons (hardware buttons, or software buttons provided as a persistent element of the operating system's user interface [10])
- Natively running code, access to all features of the underlying platform's native SDK
- Application lifecycle
    - Events such as start, pause, resume
    - Ability to run a background service or similar
    - Triggering other applications (e.g. map, phone call, mail application)
- Access to data of other applications or provided by the operating system, e.g. contacts

Apart from native functionality, a framework may of course provide own APIs that abstract common features. Important functions are for instance

- Web (service) access, e.g. support for HTTP(S) and SOAP requests, ability to use data formats such as XML or JSON
- Data layer, e.g. an ORM that eases retrieval and storage of data, or an MVC-based code architecture to facilitate separation of data access, content and presentation (which may be important if different screen designs are used on different platforms)
- Other helpful APIs, e.g. logging or push notifications

Usability features

As noted above, the term usability is different between software products and frameworks that only help to create such products.

For software products, usability in the broad sense is defined in *ISO 9241-11* as the "extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use" [ISO1998]. In other words, users must have the ability of completing certain tasks as precisely and completely as possible using the application (effectiveness), with as little "resource" usage as possible such as small time effort (efficiency), and with a positive subjective impression or reaction (satisfaction).

[10] In Android 3.0 and 4.0, Google introduced a fixed layout of three buttons drawn by the operating system on the bottom of the screen (back, home, switch between applications)

With frameworks, the usability category measures the extent to which designers and developers can implement a high usability in a mobile application project. Hence, this category answers the question: how well does a framework support features and concepts that can help to improve usability of the developed application? This includes features offered for creating a user interface: typical elements like buttons, lists, a tab or navigation bar, but also more mobile-specific elements like a paging component for switching between several pages by swiping from left to right, for example. Only features of a framework are considered, not usability in the context of the final application. The figure below depicts the difference again (orange part considered for evaluation of a framework in this category).

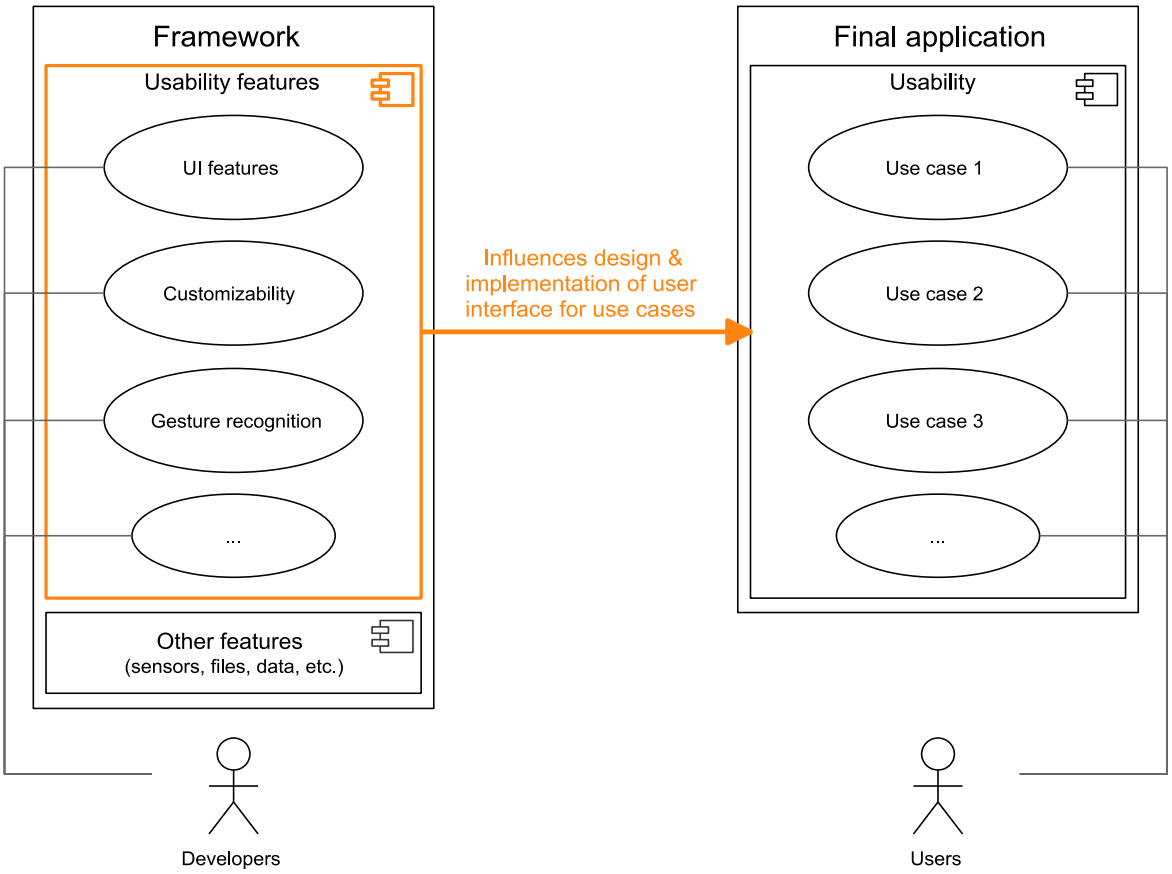


Figure 5: Usability criterion in framework vs. final application

The support for touch or multi-touch gestures is very important because they can improve usability by allowing users to achieve certain actions quicker and in a familiar way, since other applications use these gestures, too. As an example, the “pull down to refresh” pattern can often be found in applications like e-mail clients or *Facebook* – if a list component is scrolled to the very top, and the user shortly swipes down with the finger, an application can recognize this gesture and refresh the list.

Another important criterion is customizability, meaning which attributes of UI components can be changed and to what extent. This includes colors, transparency, font type and size, shape (e.g. round corners). Corporate identity, for example, may prescribe a certain color scheme or custom font type. The better an interface can be accommodated to the intended design, the better its usability can *potentially* be – or in other words, if the necessary customizations and features are not available, some usability features (like gestures and UI patterns based on them) cannot be implemented, or only with high effort.

In conclusion, support for typical and possibly non-standard UI components are the foundation for creating an effective user interface. Their performance, customizability and support for gestures help to make the interface efficient to use. Third, user satisfaction comes from the final design and performance of an application and, consequently, is only indirectly influenced by a framework's features. Note that user interface performance is considered in the "reliability & performance" category below. Mentioned features for user interface creation are evaluated in this category, not in the "functionality" category above.

User experience forms a great part of this category, since UI elements and features alone do not define how good the usability actually is for end users. However, this thesis focuses mainly on the point of view of developers, not on end user experience. Instead, support of features that are necessary to create high usability on mobile devices is considered as a major factor, as explained above.

Thus, the incorporated points of comparison are:

- Basic and advanced UI features and components, customizability, gesture support
- Ability to use native UI controls, e.g. the iOS navigation or tab bar, Android's action bar or Windows Phone's application bar
- Easiness of applying platform-specific UI looks
- Subjective UI impression

Developer support

This category aims at the ease of use that a framework offers to a developer. For example, the documentation completeness and quality plays an important role, but also the development tools provided. Frameworks based on web technology may allow testing a mobile application in a normal browser or simulator, which would give the developers the advantage of modern debugging tools built into browsers (e.g. the WebKit development tools that can display the HTML document tree, CSS rules, loading times, a JavaScript console and other helpful utilities). Other frameworks may provide their own or support for an existing simulator (e.g. iPhone simulator) to run and test applications.

In the comparison chapter, it will become clear that some frameworks may restrict developer tools to, for example, a certain IDE in order to do common tasks (build, framework update, etc.). This hinders developers, who typically have a preference for certain tools or have/want to use an automated build environment. Restrictions of this type negatively influence the score in this category.

Overall, the following points are considered for the developer support category:

- Documentation
- Debuggability and testability
- Tool restrictions
- Learning effort

Reliability & Performance

These categories are defined separately in the FURPS+ model. For simplicity, they are combined here.

A reliable application is able to “perform its required functions under stated conditions for a specified period of time” [IEEE1990]. This is particularly important for mobile applications: since they typically run with very restricted resources and conditions (e.g. available memory, process paused while application is in background), applications should run without crashes even if, for example, a certain operation needs a high amount of memory for a short time. The evaluation chapter will present examples of problems that can arise regarding reliability.

Performance – specifically in the area of mobile applications – is mainly concerned with short response times for user interactions (e.g. because blocking operations are not done in

the background). Thus, an example of a poorly performing application feature could be a user interface that takes several seconds to switch between screens. Of course, general performance criteria for software apply here, too, i.e. any measurable attribute that affects speed, waiting times or accuracy.

As an example, the *Facebook* application for iOS shall be considered. The version available before August 2012 was built using HTML5 for displaying the possibly long list of news. Many users complained that the application is reacting and starting up very slowly, so Facebook decided to re-implement the whole application as native iOS application with several performance-related customizations, allowing the new version to start faster and display/scroll content with less delay [Facebook2012].

Both categories play a big role in customer/user satisfaction. If an application crashes in certain situations or at the first start, or if its user interface is not responsive sometimes, this may be a reason for customers to switch to a competitor's application.

The following points are considered for the reliability and performance category:

- Stability
- Size of packaged/installed apps
- UI and application performance

Deployment, Supportability, Costs

In the *FURPS+* model, supportability is determined by the "ease of changes to the system after deployment" [Brügge2004]. Since app stores are important for distributing applications throughout mobile platforms, easy deployment is another important point to consider, in particular regarding the framework's support for packaging and its compatibility with the restrictions of the stores. For example, Apple's rules forbid execution of code retrieved from a remote source (example use case: self update) [Apple2012a]. The creator of a framework may provide additional services such as continuous or semi-automatic builds, allowing quicker testing and publishing of new versions.

Maintainability is another aspect of this category, comprising several criteria related to the effort of keeping a product operational and making necessary changes to it. Evaluation of these criteria should be based on real applications and change requests over a longer period of time, in order to gain meaningful experiences. Also, the ease of analyzing the source code of a software product, for example by a support engineer who was not one of the original developers, is an important part of maintainability. Therefore, maintainability unfortunately could not be reasonably tested in the time I had available for my thesis.

Supportability also includes the portability to other systems – in this case, other mobile platforms. Experiences porting the sample business application from Android to iOS will be documented in the evaluation. Furthermore, the activity of development of a framework and the viability of its vendor are important, especially if a business wants to use a certain framework in their long-term mobile strategy. Availability of professional support options, their cost and the basic costs of the framework are interesting aspects, as well.

As a last criterion, internationalization is assigned to this category [Brügge2004], that is, support for architecting an application in a way that it can easily be adapted to other languages (translation) and cultures (e.g. different formatting of dates and numbers).

The following criteria are considered:

- Compatibility with app stores
- Simplified or automatic builds
- Activity of framework development
- Variety of supported mobile platforms
- Lack of dependencies (e.g. a runtime environment has to be installed separately to run an application)
- Built-in support for internationalization, i.e. translations and localization (formatting of dates, currency values and other locale-dependent settings)
- Direct costs
- (Professional) Support options and costs

3.2.2 Summary of considered criteria

The table below summarizes the five categories and the criteria that are considered for the comparison:

Functionality	Usability features	Developer support
• Location and network • Data access • Hardware sensors & actuators • Native buttons • Natively running code • Application lifecycle • Other helpful APIs	• Basic and advanced UI features and components, customizability, gesture support • Native user interface functionality • Subjective UI impression • Easiness of applying platform-specific UI looks	• Documentation • Debuggability and testability (e.g. in browser, WebKit tools) • Framework and project setup time • Tool restrictions • Learning effort

Reliability & Performance	Deployment, Supportability, Costs
• Stability • Size of packaged/installed apps • UI and application performance	• Compatibility with app stores • Simplified or automatic builds • Activity of framework development • Variety of supported mobile platforms • Lack of dependencies • Internationalization support • Direct costs • Support options and costs

Table 1: Overview of evaluated categories

3.2.3 Evaluation with scores

In the evaluation described in the following, each framework is rated with regard to the above categories. Each category has a weight between 1 and 5 depending on its importance from the standpoint of a business that wants to develop mobile applications. The reasoning behind assigned weights will be explained below. Scores between 1 and 5 are assigned, resulting in a total weighted score for each framework by multiplying, for each category, the fixed weight by the score, and dividing the sum of these products by the sum of the weights. This puts the final score in the range 1 to 5 as well.

The single scores have the following meaning:

- 1 = not fulfilled
- 2 = poorly fulfilled
- 3 = basic expectations fulfilled
- 4 = more than fulfilled
- 5 = all expectations completely fulfilled or exceeded

The same procedure is applied to the native development environments of the Android and iPhone platforms in order to compare the cross-platform frameworks with native approaches.

Weights are assigned as follows to the categories:

- **Functionality: 4**

  Certain built-in features and support for hardware sensors and actuators can be very important depending on the type of application. Extensibility, e.g. being able to add code that uses native functionality or to use third-party libraries, is even more important.

- **Usability features: 3**

  Usability and user experience in general greatly depends on a good application design. However, in case a framework does not provide extended features such as gesture recognition, some features of the finished application (e.g. “pull to refresh” for lists) are harder to implement. Extensibility of a framework, e.g. using native code to implement certain functionality (also UI-related), is part of the above category and thus the *usability features* category is weighted lower because it mainly regards available base features for UI design and implementation.

- **Developer support: 2**

  While this is an important point, tool restrictions and debuggability play a smaller role with mobile applications because they are typically less complex or require less development time than comparable applications for PCs. For example, mobile applications for employees (B2E) cannot be as complex as Microsoft Excel spreadsheets or large applications with many keyboard shortcuts because operations on mobile devices must be simpler. This leads to less complex user interfaces, less development effort and as such the need for helpful developer tools is not as important.

- **Reliability & Performance: 3**

  The size of applications matters because of limited device storage space, restrictions on the app stores and incurred traffic costs. For example, with a size of more than 50 MB, an application on Apple’s store can only be downloaded over Wi-Fi or the iTunes software on a computer, not over mobile networks [Apple2012b].

Performance is a critical point for customer satisfaction. *Forrester* studies from 2006 and 2009 report that the expectations for web site loading times have increased over the last years, a measure that also applies to applications running directly on a mobile device. Furthermore, slow performance is one of the main reasons why consumers are concerned about using their mobile device for purchases [Forrester2009]. As a conclusion, performance is important for customers to use – and keep using – an application. Nevertheless, new technologies such as multi-core processors are already emerging and will help mitigate performance issues. Also, cross-platform frameworks are relatively new and still have the potential to progress in terms of performance.

Together with reliability, a common and surely important aspect of software, this category is considered of medium importance overall.

- **Deployment, Supportability, Costs: 5**

  Costs and time-to-market are key factors for the success of an application. The mentioned short development cycle, that mobile applications typically have, requires fast delivery of new versions, which can be simplified by automatic build services or built-in packaging or deployment functionality. This category also comprises the fulfillment of app store rules, support for different platforms and how actively a framework is and will be developed. Costs of a framework and available support plans are also evaluated. Therefore, I consider this the most important category because it includes factors that influence the cost, ease and speed of deployment in the long term.

3.2.4 Time measurements

In order to get a rough understanding of development time differences, the time that I needed to implement the sample application was measured for each framework. This does however not provide any statistically significant values because it is influenced by my previous level of experience with the used technologies (e.g. HTML, CSS, JavaScript, Java) and SDKs (Android course at university, but no experience with iPhone SDK). Learning effects may also play a role: as the same application is implemented multiple times, functions or concepts might be reused. Therefore, the results only represent an auxiliary point in my

work and I leave it to the reader whether and how to interpret the measured times. These results can be found in appendix C.

3.3 Selection of considered frameworks

Appendix B shows a non-exhaustive list of available frameworks. Not all of these frameworks could be compared in this thesis for timely reasons, so a pre-selection was made based on the cross-platform approach, observed development activity and size of the developer community, popularity and subjective suitability for the development of business applications.

First of all they were filtered by development activity since I want to figure out whether the cross-platform frameworks are already usable and it makes no sense to evaluate projects that are not yet developed enough or at risk of being abandoned because of too little active involvement of developers. This eliminated mainly small and not very active open source projects like *mobl*, a custom programming language designed specifically to create mobile applications easily, but also projects that still need further development to be usable in production environments, e.g. *XMLVM*, a cross-compilation solution for Android and iPhone applications.

Frameworks which are marketed and primarily used for game development, such as *Corona* or *Marmalade*, were also not considered because their future development may not focus on supporting business applications and their requirements.

Several frameworks only support user interface features such as platform-independent uniform looks or close-to-native look and feel. Since *PhoneGap* was selected for evaluation and only supports device functionality and other APIs like data and file system access, an additional framework had to be selected that adds UI functionality. The *Rhodes* framework was also selected and contains an adapted variant of *jQuery Mobile* for UI purposes, so another UI-only framework was chosen to complement PhoneGap: from the different options, *Sencha Touch* seemed to be the most actively developed solution, backed by the company *Sencha Inc.*, and also provides an architectural approach for applications (MVC) that may help to create a clear code structure and ease development.

In summary, the following frameworks were chosen for the stated reasons:

- **Titanium**

  Even though developed by only one company, the framework offers a unique approach (developing native applications using JavaScript as only language, not HTML-based) and a large number of developers seem to have adapted it – judging from activity in the developer forum, marketplace [11] and the fact that the vendor Appcelerator offers its own conference and certifications for Titanium developers.

- **Rhodes**

  With an HTML-based approach and distinction between backend code (Ruby) and frontend code (HTML/JavaScript), Rhodes provides a new way to develop mobile applications, targeted primarily at developers who have existing web development experience. The open source MIT license and freely available online build service *RhoHub* were major arguments for the framework.

- **PhoneGap and Sencha Touch**

  Very active development by multiple companies [Apache2012], large open source involvement of the community, clear API design and extensibility through native plugins were the determining factors for the selection of PhoneGap. In contrast to the two frameworks above, it uses a different approach by allowing developers to use the tools they would normally use for native development on a single platform (e.g. *Eclipse* for Android, *Xcode* for iOS development). As noted above, the UI and application framework *Sencha Touch* was chosen to complement PhoneGap, which only supports HTML display and device features, but not user interface creation.

For timely reasons, other interesting frameworks could not be tested, as the evaluation involves comparison with the native SDKs of Android and iPhone (i.e. iOS SDK). The *MoSync* framework, for example, offers a similar application architecture as PhoneGap by wrapping an HTML application in a mobile application project, but additionally offers native development using C++ as alternative. Evaluating web-based solutions like *Application Craft* or *AppFurnace* would have been interesting as well.

[11] Web-based store provided by Appcelerator where developers can offer/sell modules and additions for the Titanium framework, https://marketplace.appcelerator.com/, accessed: 2012-02-12

4 Comparison of the frameworks

In this chapter, each of the selected cross-platform solutions and native SDKs is shortly described together with an explanation of its architecture. The installation process, creation of an application project and a typical development and testing cycle for Android are outlined (assuming the Android SDK is installed already).

4.1 Titanium

4.1.1 Overview and architecture

Titanium is a framework developed by the company *Appcelerator Inc.* The latest major version at the time of writing is Titanium 2.0 as released in April 2012. It leverages JavaScript as the main development language, but does not primarily use HTML or other web technologies. As a side note, HTML inclusion using an embeddable web view component is supported. The architecture consists of two major parts: a standalone JavaScript interpreter executes the application code and the Titanium library provides device functionality like location services, file system access, native UI components and others, abstracting away many (but not all) differences between different mobile platforms. An application can use the Titanium library from anywhere in the source code. Calls to methods or properties of the global object `Ti` are passed to the native implementation for the respective platform. This bridging facility also allows native modules to be added and used in a project from within the JavaScript code. Compilation of an installable application package is done differently on each platform. For example on iOS, the JavaScript code is inlined as string in a generated Objective-C file, which is then used by the interpreter *JavaScriptCore* [Whinnery2012]. It is then bundled together with a JavaScript interpreter and the Titanium library in order to form the complete application package. The below figure outlines the architecture of a Titanium-based application.

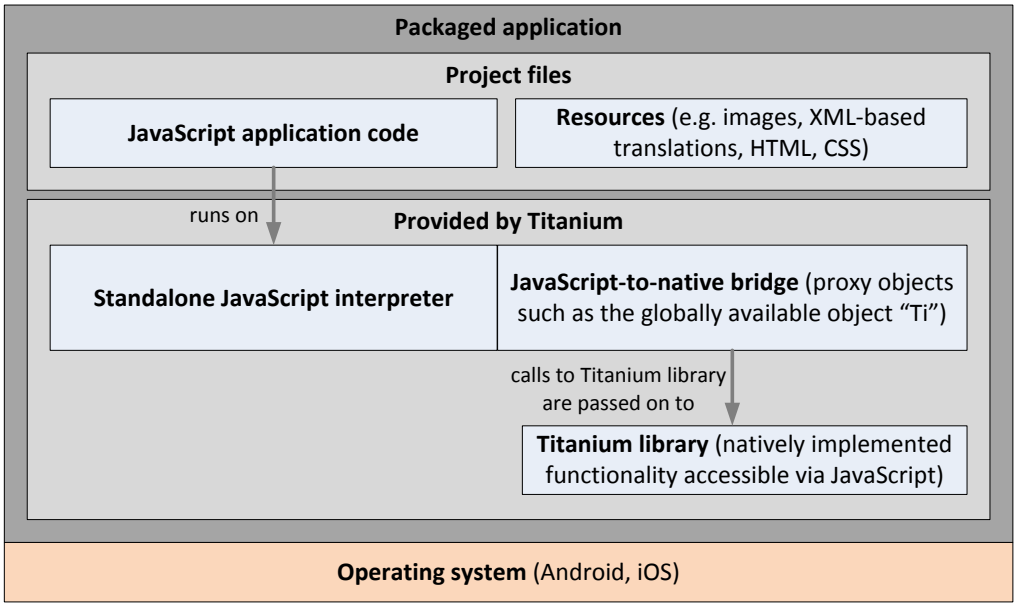


Figure 6: Architecture of a Titanium-based application

4.1.2 Installation and development procedure

Appcelerator provides *Titanium Studio* as recommended IDE for developing applications with the Titanium framework. It is available from the company's web site for Windows, Mac OS X and Linux after signing up for a user account. On the first run, the current SDK version is downloaded, extracted and set up automatically for use with the IDE. As next steps, it allows to set up paths of required SDKs (in this case the Android SDK) and creation of a new project. At last, a run configuration as known from the Eclipse IDE, on which Titanium Studio is based, can be created and started in order to run the Android emulator, for example. As an alternative to the IDE, Titanium provides the command line tool `titanium.py` that offers basic functionality for a project (creating a project and running it on a device or emulator). The easiest way to retrieve the SDK is to let Titanium Studio download it automatically into a predefined location on the local system. Alternatively, one can build it from the source code (open source). Appcelerator also offers a web page with continuous builds of the SDK [12].

[12] http://build.appcelerator.net, accessed: 2012-08-01

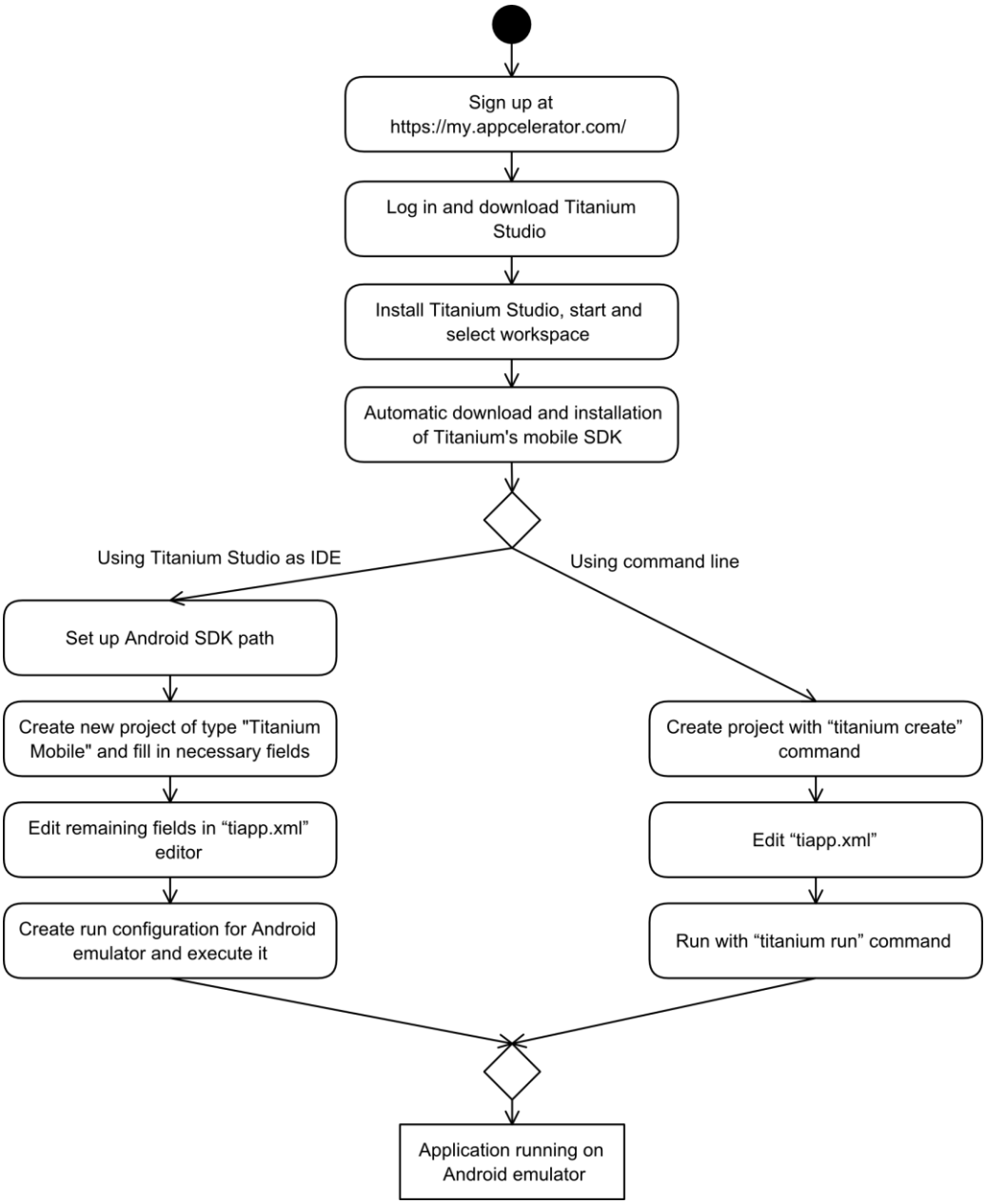


Figure 7: Procedure of installation and first application run using Titanium

The typical development cycle is fairly simple. While an application is running in the Android emulator or on a device, changes to the source code are not automatically reloaded, so the application has to be repackaged and started again for changes to take effect. Uncaught errors in the application lead to a dialog window that shows what kind of error occurred and where. For example, exceptions are shown in this way – one can then decide whether to close the application or to continue, leaving the application running. Breakpoints are possible in debugging mode, including modification of variable values.

4.2 Rhodes

4.2.1 Overview and architecture

The framework *Rhodes* was developed by the company *Rhomobile* which was acquired by *Motorola Solutions* in October 2011. Since major version 2 (published in 2010), the code is open source under the MIT license. The latest version at the time of writing was 3.3.2. In the subsequent version, several features like barcode recognition, near field communication, signature capture and automatic database encryption were removed and are now available in the so-called *RhoMobile Suite*, requiring a different license [Motorola2012].

The client-server model is used as basic architecture. More exactly, the user interface of a Rhodes application is shown entirely by a web view component that renders HTML content and evaluates CSS styling and JavaScript code just like the device's normal web browser would. Views are written using *ERB* (*Embedded Ruby*), a common templating system that is also used in Ruby-based web frameworks such as *Ruby on Rails*. The application's backend contains models and controllers (based on the MVC architecture) written in the Ruby programming language. Any controller action can be accessed from the HTML-based views by accessing the respective URL. The framework includes a simple web server that is embedded in created applications and handles the translation of URL accesses to Ruby method calls in a controller, and returns the view that is created as a result by the controller. Controllers do not have to return a new view, but can also just run a task in the background without rendering an HTML page. They can directly interact with the user interface by sending JavaScript code which is directly executed by the web view.

Device capabilities can be accessed through the Rhodes library, and thus only from backend code (controllers written in Ruby, not from views [13]). Device-specific functionality is implemented as native Ruby extension and thus can be called directly. A reduced Ruby standard library is included with the Ruby virtual machine that is bundled with an application. As a result, common functionality such as file access does not require the implementation of a separate API in the Rhodes framework.

[13] In a subsequent version, it seems that some functionality can also be accessed in HTML and JavaScript.

Additionally, Rhodes provides functionality for accessing, storing and synchronizing data. A simple ORM called *Rhom* is included, allowing for simple model definitions either with or without a fixed scheme of properties and types. Typical CRUD operations, pagination and also raw SQL queries are possible. The *RhoSync* component allows optional synchronization of models with a remote backend – since it is based on a very simple adapter architecture, it can be used with any backend such as key-value stores, web services, etc.

The following figure gives an overview of the above architecture description for an application based on the Rhodes framework:

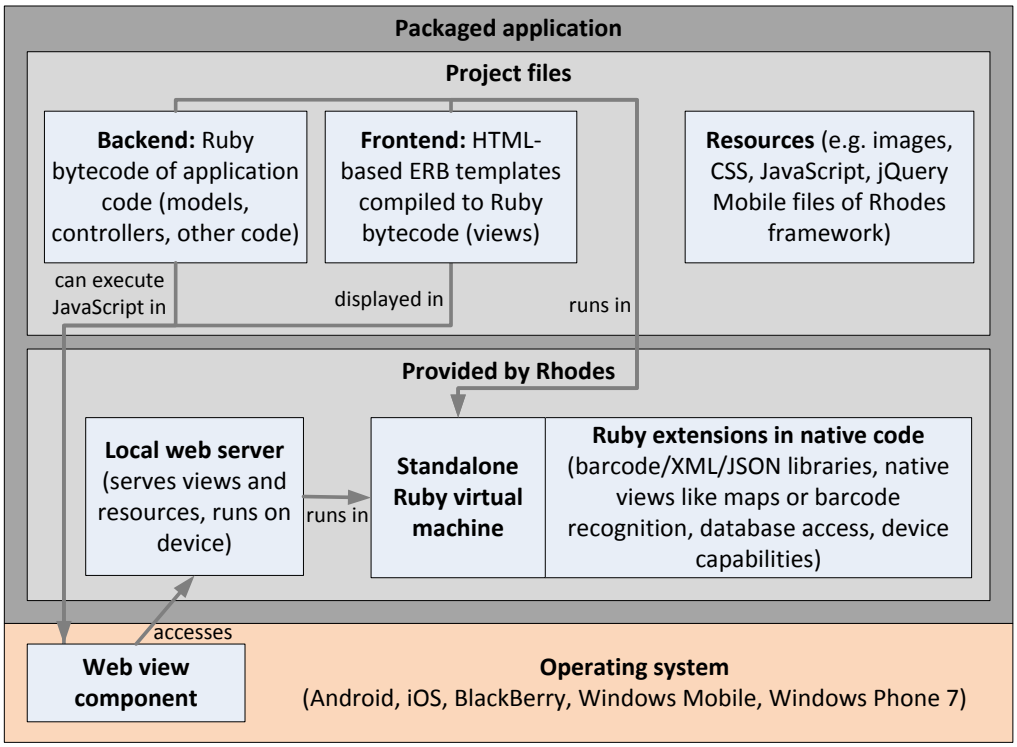


Figure 8: Architecture of a Rhodes-based application

4.2.2 Installation and development procedure

The company Rhomobile developed *RhoStudio*, an Eclipse-based IDE that also includes all necessary dependencies in its installer, i.e. *Ruby*, *Rhodes*, *RhoSimulator* and a compiler chain to compile additional Ruby libraries. RhoStudio was integrated into Motorola Solution's *RhoMobile Suite* [14]. This description is about the version that was previously available from Rhomobile's web site [15]. Rhodes applications can be created using the integrated project

[14] Freely available at http://www.motorola.com/Business/US-EN/RhoMobile+Suite/Downloads, accessed: 2012-06-13
[15] http://www.rhomobile.com/products/rhostudio/, accessed: 2012-03-29

template. *RhoSimulator* works on Windows and Mac OS X, and allows simulation of an application and easy debugging and inspection of the HTML views. Apart from RhoSimulator, RhoStudio allows running the application inside an emulator or directly on a device.

After installation of the IDE and other dependencies, the `rhodes-setup` tool must be invoked manually and lets the user enter necessary information and paths to e.g. the Android NDK (*Native Development Kit*) which can be used to incorporate code in Android applications that runs directly on the device hardware. In the case of Android development, *libruby* (the virtual machine for running Ruby code) is compiled for the ARM processor architecture once (using the Android NDK) before an application is compiled in RhoStudio.

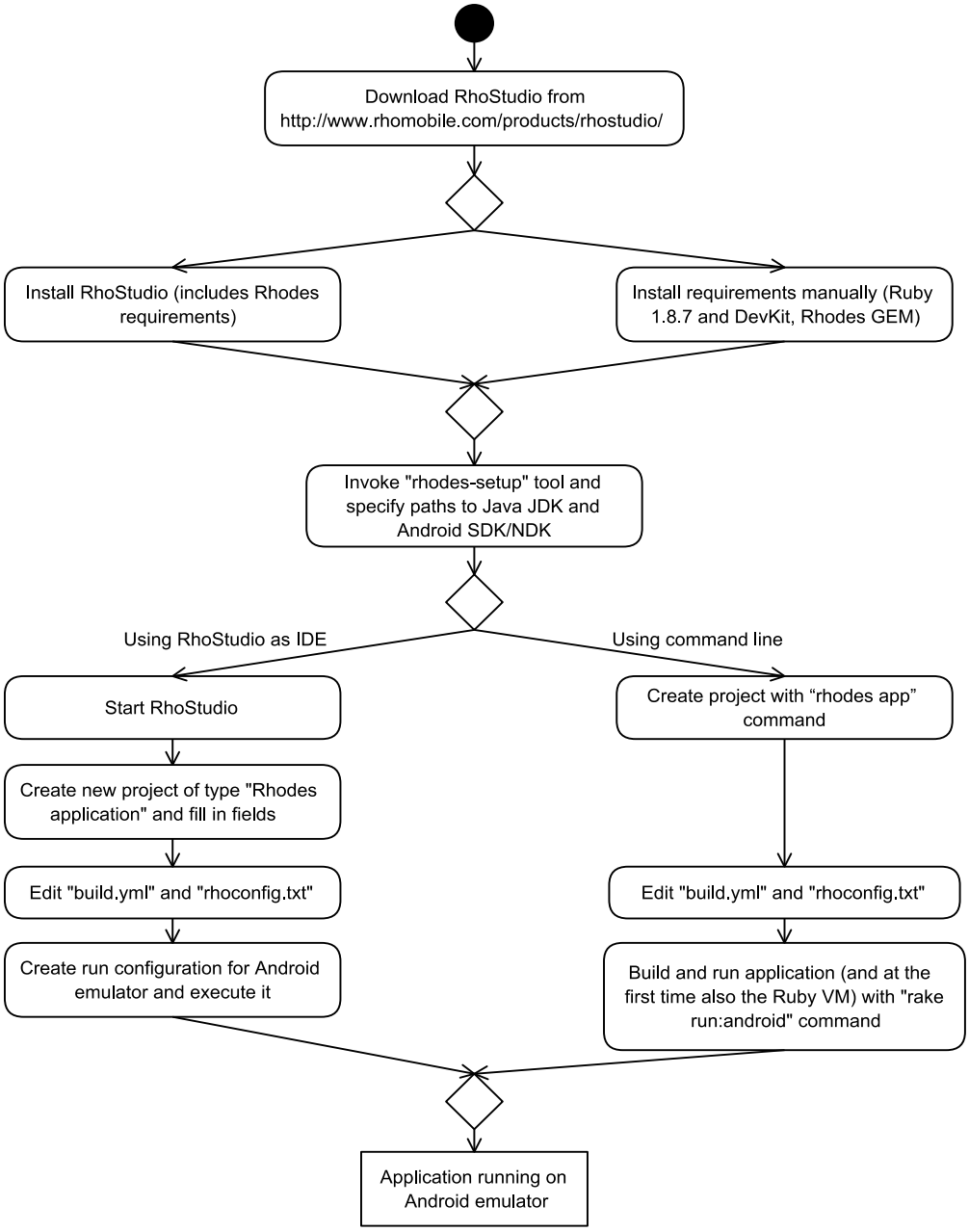


Figure 9: Procedure of installation and first application run using Rhodes

Application development can be done with RhoSimulator which has the advantage of starting and running faster than the Android emulator. However it cannot simulate all device functionality and might behave differently in some cases, for example it might seem that a particular Ruby method or module is available with Rhodes, but it actually is not because the standard library of the Ruby installation that comes with RhoStudio has more modules than the standard library packaged for installation on a device. RhoSimulator starts up with very little delay and allows debugging using separate windows for the log file and a "Web Inspector" that offers the very same functionality as found in the developer tools of the *Google Chrome* web browser (HTML/CSS inspection, network requests, loading time, script console, etc.). RhoStudio allows Ruby code debugging if the application runs in RhoSimulator (not in the Android emulator), allows breakpoints and variable/expression watches, but no changes can be made to variables. Uncaught exceptions are automatically logged (including the call stack) and lead to a HTTP 500 response of the currently loading page.

4.3 PhoneGap and Sencha Touch 2

4.3.1 Overview and architecture

As mentioned in the section about the selection of frameworks, these are essentially two separate frameworks, combined because a mobile application using device features and nontrivial UI cannot be developed with only one of them.

PhoneGap

PhoneGap is an open source project initially developed by the company *Nitobi* which was acquired by *Adobe Systems* in 2011. It allows development of mobile applications using web technologies by providing an interface to a full-screen web view component. HTML pages are loaded from the application's files. Additionally, PhoneGap provides APIs for device functionality through a JavaScript-to-native bridge and native code can call back JavaScript functions inside the web view as well. This bridge is implemented differently on supported platforms. For example on iOS, calls from JavaScript to native functionality is done by setting the web view's location (`document.location`) to a special URI [MacFayden2011] (e.g. `gap://sessionId@com.phonegap.camera/getPicture`). The native part of PhoneGap can then intercept this change, stop the location change and open the camera screen in this example. Developers can extend PhoneGap with native plugins by implementing a simple interface which also differs across the platforms – results can be returned synchronously or

asynchronously (using callbacks to JavaScript functions). No user interface functionality of any sort is included, so PhoneGap could be seen as a wrapper for mobile applications that use HTML and other web technologies for displaying their user interface.

As a side note, PhoneGap was donated as open source project to the *Apache Software Foundation* and its name changed to *Apache Cordova*. The name PhoneGap now refers to the main distribution of Cordova [16].

Features that are not natively supported on a certain platform are mimicked by PhoneGap's own implementation. For instance, if the *W3C Web SQL Database* specification is not implemented by a platform, PhoneGap's own implementation is used. Close adherence to standards (like *W3C* standards related to HTML5) enables developers to use the same interface on all platforms.

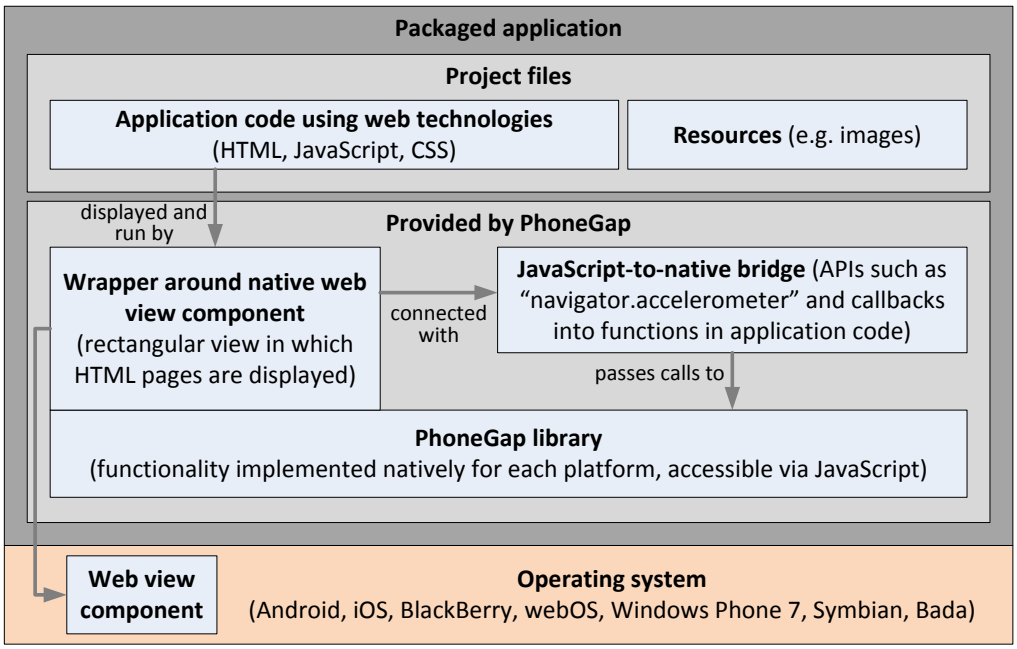


Figure 10: Architecture of a PhoneGap-based application

[16] Cf. http://phonegap.com/2012/03/19/phonegap-cordova-and-what's-in-a-name/, accessed: 2012-03-25

Sencha Touch

The company *Sencha Inc.* develops *Sencha Touch* primarily as web framework, but also – as the name suggests – for touch-based applications. In contrast to PhoneGap, Sencha Touch barely offers any device functionality. Access to camera, location, acceleration sensor and device/network information was made available in version 2, but is only supported if Sencha's native packaging feature is used which creates a similar wrapper like PhoneGap.

Sencha Touch supports user interface development and data retrieval and persistence. The framework is designed in a way that fosters the MVC approach: the data layer is represented by models that define data fields and optionally types, and "stores" which define where and how model instances are stored and retrieved. *Local storage* (from the HTML5 *Web Storage* standard) is one supported way for persisting data. Views and controllers are defined in separate classes.

Regarding the architecture, an application usually consists of a single HTML page that loads the application's JavaScript code. Routes, i.e. URL mappings, are used to determine which controller class handles a URL. For example, a location of `index.html#document/5` would display a certain document. This enables separation of the application's functionality and lazy loading of the controller/view code. Lazy loading, i.e. loading a class implementation only when it is needed, is a built-in feature of the framework.

4.3.2 Installation and development procedure

The following diagram shows the separate installation procedure for PhoneGap and Sencha Touch. During development, I automated the build process using the Eclipse IDE – since this setup was a one-time effort and has never been documented before, it is not part of the setup/development time measurement. Basically, Sencha Touch compiles the web part of the application (HTML, JavaScript and CSS files) together, and the result must be copied to a folder of the Android project where the web view component can find it. The build automation setup is not relevant here. An article about it can be found on my web site [17].

[17] http://andidog.de/blog/2012/06/packaging-a-sencha-touch-2-application-with-phonegap-for-android/, accessed: 2012-06-08

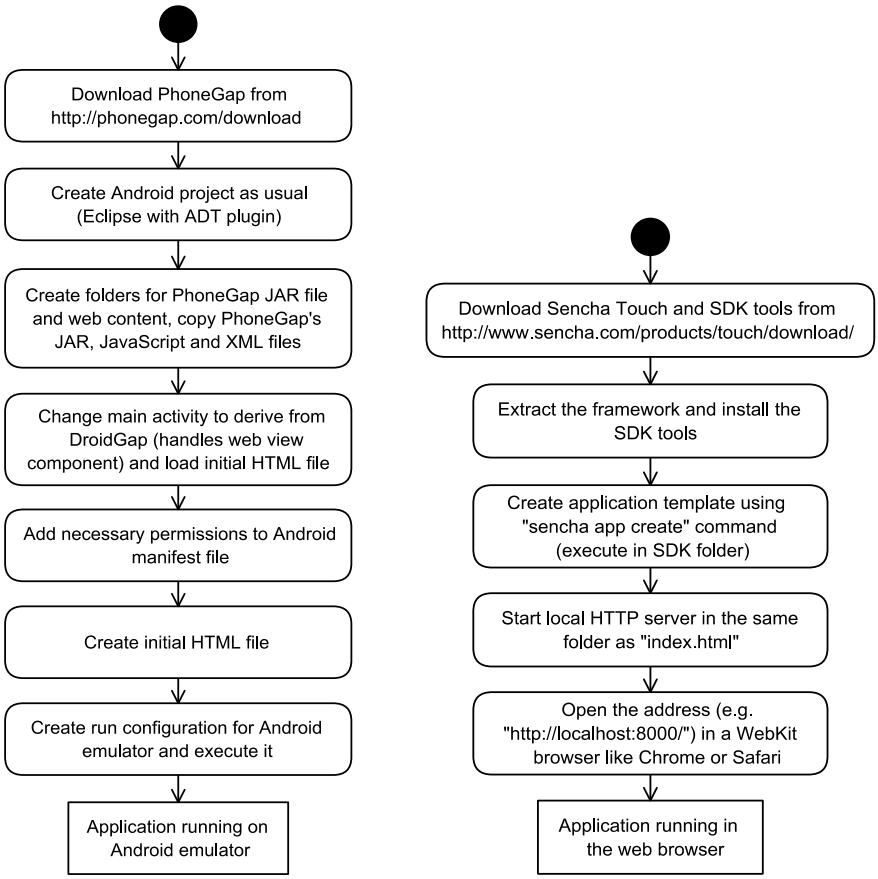


Figure 11: Procedure of installation and first application run using PhoneGap and Sencha Touch, respectively

The development cycle is slightly different between tests in a browser and on a device/emulator. Browser testing is of course not possible if device features such as the camera are used, so in order to enable testing in the browser, the application should check if those features are available at all. In order to view the application in a browser, only a local web server must be started, serving the HTML, JavaScript and other files. Any changes in the code thus only require a page reload. For device or emulator-based development, the web part of the application and the native project must be combined. With Android, for instance, the web part must be copied to the `assets/www` folder from which PhoneGap's web view implementation will serve the web page. As mentioned above, the Sencha Touch build tool allows collecting all necessary files in one folder, and together with some additional steps, the Sencha Touch part (web page) and the PhoneGap part (native web view for displaying the page) can be combined automatically whenever the developer wants to start the application on a device or emulator. Parts of the Sencha Touch framework which are not needed are left out automatically by the build tool in order to save space and avoid higher loading times caused by unused code.

4.4 Android

4.4.1 Overview and architecture

The *Android* platform and SDK is developed by the *Open Handset Alliance*, with *Google* as primary software contributor. The latest update as of August 2012 is Android 4.1.1 ("Jelly Bean"). Most code runs on the virtual machine *Dalvik*. Java is the usual programming language, and a large subset of the Java standard library is supported, together with many APIs provided by the Android class library itself. Native libraries that directly run on ARM processors can be compiled with the Android NDK (native development kit).

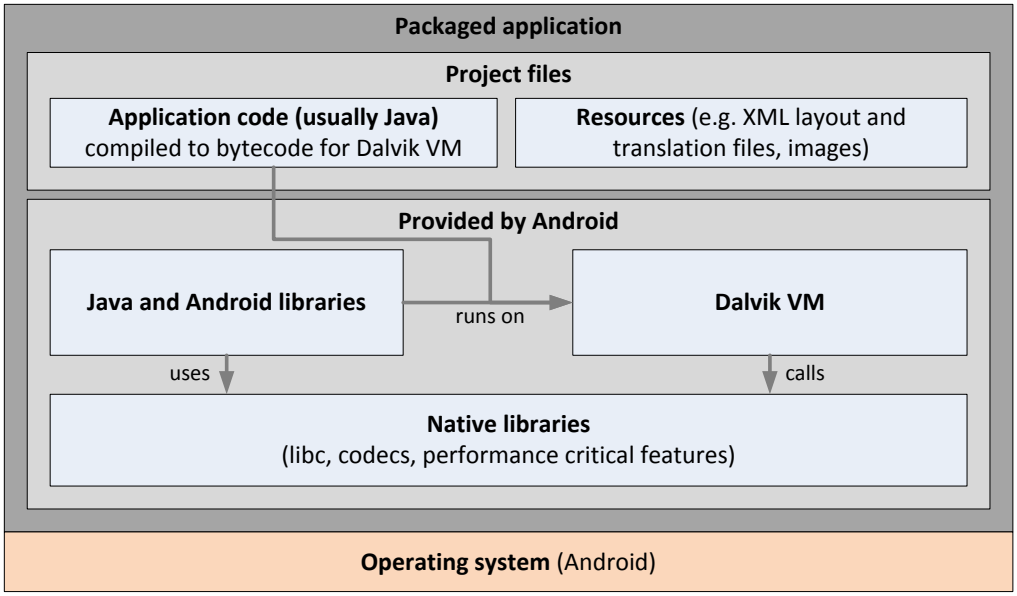


Figure 12: Architecture of an Android-based application

4.4.2 Installation and development procedure

Development of Android applications is often done using *Eclipse* and the *Android Development Tools* (*ADT*) plugin. This plugin allows project setup, graphical UI layouting and design, editing of Android-specific XML files (translations, configuration, layouts, etc.), creating application packages, running/debugging a project on the Android emulator or a real device, and other features.

In order to create a simple project using Eclipse, one first has to download the Android SDK tools which in turn take care of downloading the latest SDK and emulator. After installing the Android plugin for Eclipse and configuring the path to the Android SDK, a new project can be created by entering information such as the package name or the supported SDK versions.

The application can then be started with a new “run configuration”, choosing to deploy to an emulator or a device connected via USB.

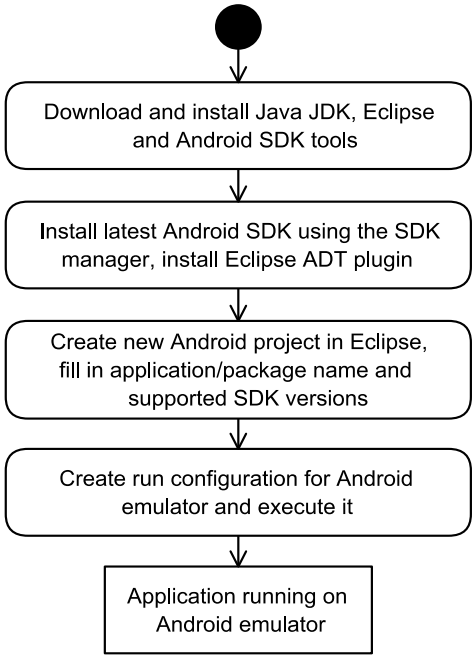


Figure 13: Procedure of installation and first application run using Android

A typical development cycle looks as follows: After making changes to the source code, the running application must be restarted. The Java code is compiled to Dalvik-compatible bytecode and an application package is generated, signed with a testing certificate and uploaded to the device or emulator. The emulator has to be started once which may take some time. Eclipse offers two run modes – with and without debugging. In debug mode, breakpoints can be set (also with conditions, e.g. “halt if x > 0”) and variable values can be inspected and changed when the application was halted at a breakpoint. Unexpected errors lead to the application being quit and a message box appears reading “Application has stopped unexpectedly” – if the cause was a Java exception, the call stack is logged automatically and can be seen in the “LogCat” perspective of the Android plugin in Eclipse.

The Android SDK tools also provide additional programs for development purposes: for example a manager for emulators (there can be many emulators of different Android versions) and the *Dalvik Debug Monitor* (*DDMS*) that can display the log of any running emulator or connected device, download and upload files to its file system and change settings (e.g. simulated geolocation, network availability and speed, incoming test phone calls and SMS).

4.5 iOS SDK

4.5.1 Overview and architecture

Apple's *iPhone* devices run on the *iOS* operating system which is also used for other mobile devices from Apple, such as the *iPad*. Therefore, most considerations for iPhone apply to all devices that run on iOS because an application can use mostly the same Apple-provided and external libraries and functions.

The iOS SDK is provided by Apple and includes [Apple2011]

- Bundles of core functionality resources, so-called "framework" files (typically consists of shared library, header files and other required resources). They can be added separately to a project if required. One example is the *UIKit* library that offers interfaces for creating, drawing and controlling user interfaces.
- *Xcode* and accompanying tools for debugging and performance analysis
- iOS simulator that simulates iPhone and iPad devices
- Documentation (library reference and higher-level articles about concepts and tools)

Xcode is the only officially supported IDE for iOS development and no professional alternatives for native development seem to exist. Since Xcode and the rest of the iOS SDK is only made available for installation on the *Mac OS X* operating system, developers are restricted to use Apple-branded computers for iOS application development. It should be noted that Xcode includes automatic build tools that can compile Xcode projects from the command line, and as such it is possible to set up a central server for compiling and signing iOS applications.
Note: *PhoneGap Build* (see also p. 97) is one service offering iOS builds. The Sencha Touch native packager claims to be able to create *IPA* files (signed iOS application packages) on both Windows and Mac OS X, thus not requiring Xcode [Sencha2012].

The user interface of an iOS application is typically built with separate views and associated view controllers. Views and their relationship between each other and with the view controller, such as transitions and events (e.g. clicks) can be designed interactively in Xcode. Following the MVC pattern, it is also possible to define models with their fields and

relationships, and use them in conjunction with the *Core Data* library that acts similar to a simple ORM between models and a data store (e.g. *SQLite 3*).

Apple describes iOS with a layered architecture: The *Cocoa Touch* layer as highest layer provides iOS/touch-specific functionality, i.e. mainly the *UIKit* library. Lower layers provide general features like graphics and audio, data, location and sensor access, while the lowest accessible layer provides OS-level features such as file access and networking. All layers expose interfaces that can be used for developing applications.

Most parts of the SDK are written and accessible in the *Objective-C* programming language. As a superset of the C language, third party libraries can be linked with an application and pure C/C++ source code can be mixed with Objective-C. Together with version updates of iOS, normally also a new version of the SDK is released that contains interfaces for added or changed functionality. The dynamic character of the Objective-C programming language allows selective use of new features, for example by testing whether a class implements a certain method. All iOS applications that use Objective-C are compiled to a native executable (i.e. ARM instructions on the iPhone) and linked with a minimal runtime that handles the dynamic language features such as dynamic typing. Memory management is based on counting references to objects and releasing their memory when no more references exist. *Automatic Reference Counting* (*ARC*) is an optional feature of the Objective-C compiler that automatically inserts memory management calls (e.g. retain or release a reference to an object) where necessary [Apple2012d]. It reduces the complexity of memory management and safeguards developers from common mistakes like memory leaks (unreleased memory), incorrect reference counts or memory access errors caused by early release of an object.

Apart from Objective-C, other programming languages, including languages that run on a virtual machine, can be used and are allowed for applications distributed on the app store if they do not fetch code from a remote source [Apple2012a]. For example, the company Xamarin offers the product *MonoTouch* for iOS that allows the use of C# as programming language, based on *Mono*, the open implementation of the *.NET* framework. Other presented frameworks such as Rhodes or Titanium package applications together with a

virtual machine implemented in C, for instance, able to run the bytecode generated from the respective programming language (Rhodes: Ruby, Titanium: JavaScript).

The figure below shows the architecture of a typical iOS application:

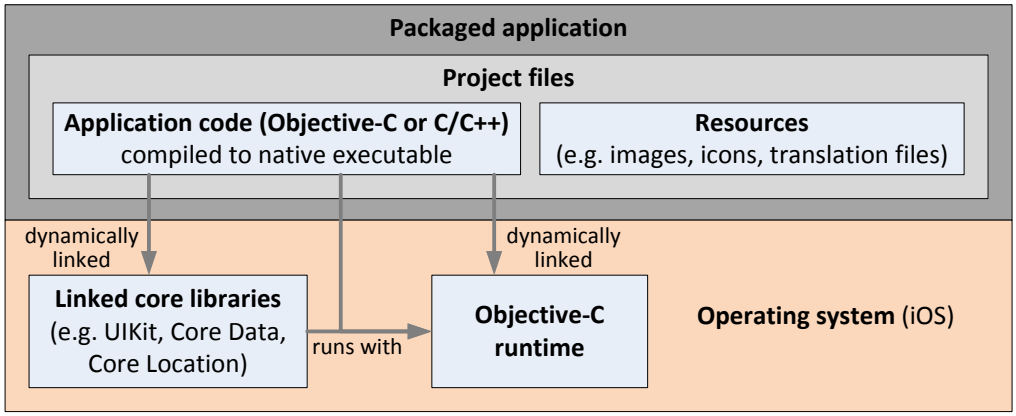


Figure 14: Architecture of an iOS application

4.5.2 Installation and development procedure

Apple offers the iOS SDK bundled together with Xcode as development IDE. The latest version can be installed from the App Store on recent versions of Mac OS X – Xcode 4.4 (latest version as of August 2012) requires Mac OS X 10.7.4 or newer. The latest iOS simulator is installed automatically, older versions (for testing on previous versions of iOS) and device debugging support for older iOS versions can be downloaded in Xcode.

For a new project, the application name and company identifier, project options and a project template must be selected. The options include automatic reference counting (see previous subsection), the use of “storyboards” for interactive design of screens and their interactions/transitions, and addition of a unit testing target to the project. Several project templates, for example an application with a master view (list) and a detail view, can be used as starting point. Running a newly created application will execute it on the simulator without further setup steps.

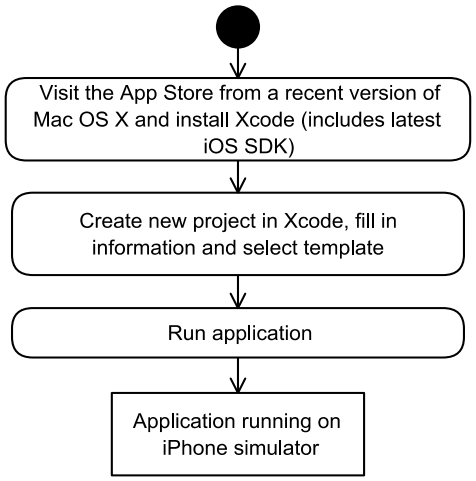


Figure 15: Procedure of installation and first iPhone application run using Xcode and iOS SDK

Development using Xcode is simple: after changing code and selecting to run the application again, it is automatically re-deployed to the simulator. A project by default has debug and release mode targets. Debug mode is selected by default and allows the use of breakpoints, value inspection and manual commands in the debugger console, e.g. for showing complex values like dictionaries or running application code for debugging purposes. Xcode also shows application log entries in the same window as the debugger console.

A connected device can also be used to test applications. Therefore, a valid "provisioning profile" must be created in Apple's member center for developers who have purchased and are enrolled in the *iOS Developer Program*. Application packages are signed for security reasons, so a developer certificate and private key must be created and then associated with one or more test devices. The target iOS version will be changed automatically if the build target differs from the version that is installed on the device. Xcode can then run and debug the application in the same way as with the simulator.

5 Sample mobile business application

In this chapter, I will introduce the sample application that I used to gather insights, experience and facts about the five presented application development solutions, regarding some of the criteria mentioned before (p. 23ff). Afterwards, I outline my experience with the frameworks in chapter 6 and evaluate them – partly based on the implementation of the sample application, but also according to known facts and capabilities of each framework.

Since results of development time and effort, application usability and the experience from a developer's perspective are vastly based on the development of the sample application, it is crucial to obtain comparable results. Hence, the sample application must be well-defined so that it can be implemented in a similar way with each framework. This section will detail on how the application shall be implemented. The application's functions are split up into visible (UI) features and functionality that runs without user interaction (e.g. web service queries). Both are described in separate sections after the idea and purpose of the application is presented. At the end of this chapter, a short overview of the system architecture is presented.

5.1 Idea and purpose

The application's main purpose is to let users order printed photographs over the Internet using their mobile phone. As phone manufacturers have been improving the quality of built-in camera sensors over the last years, sending captured pictures from a mobile device already makes sense. Relating this idea to the definition and advantages of mobile business applications (see p. 21ff), photo printing shops or counters in supermarkets could benefit from such an application by extending their business to mobile users. Customers gain the convenience of sending pictures from home or anywhere, without having to bring a physical medium to the store. The store itself saves time because pictures come over a common channel, and it is not necessary anymore to extract files from different media types (USB sticks, CDs, SD cards). A small store might not want to invest in development of an application – instead, it will be assumed that the application targets a chain of stores that shares this ordering service. Consequently, the application allows the selection of a store in which the ordered photographs will be printed. This application shall be called "MobiPrint".

A publicly accessible, HTTP-based web service and its exposed actions are pre-defined and must be used by the sample application.

Since the framework comparison should yield results on how well-suited the frameworks are for mobile business applications in particular, the application should cover many aspects of a typical mobile business application (see p. 21ff). The following characteristics are covered, regarding the overall system architecture and deployment:

- **Integration with web services and data synchronization**

  A web service – based on the *REST* principle and the *JSON* data format – is used by the application to retrieve information about old and active orders, and to send order information and pictures selected by the user.

- **Additional device features**

  Location services like GPS shall be used to find the closest store. Thus, if a location is available, the application can preselect the correct store where the printed photographs can be fetched as soon as the order is finished. Else, the previously selected store shall be used – this requires persistence possibilities (e.g. using the file system).

  The camera could as well be used to immediately add captured pictures to an order, but this is probably a feature that would barely be used because the application is expected to only be used once the user wants to make an order.

- **Additional data sources on the mobile device**

  Access to captured pictures is required in order to let the user select which pictures should be printed. Whether a specific "photo gallery" API is provided, or the application has to scan the file system itself, is not dictated here since this will influence the score in the functionality category.

5.2 Functional requirements of background processes

This and the following section detail the functional requirements, obeying the principles of their specification as explained by *Brügge*. In this case, requirements are dictated by the goal of the framework comparison, not by a customer. The description must be in particular complete, verifiable and unambiguous [Brügge2004], so that the application can be developed in a mostly identical and thus comparable way with each framework. Since the

frameworks and also the mobile platforms can differ greatly in de-facto user interface standards, patterns and layouting, restrictions to the UI implementation are only made where necessary. Requirements stated in this section are meant to describe the application completely, hence no “extra features” are implemented or considered in order to keep the development time in a reasonable timeframe.
As another hint for this and the next section about functional requirements, it must be stated that the word “should” denotes a feature that must be implemented if the required functionality is made available (by the device or framework). If this is not the case, it may have a negative influence on a score in the evaluation.

As this chapter only describes a sample application, not a full application, some restrictions are made to the complexity of MobiPrint in order to keep development time on a reasonable level:

- No authentication, authorization, request forgery protection or user distinction is enforced by the web service and thus there is no need for the application to store or send any user information. As a result, the service only manages and returns a single list of orders, without managing the relation to users. Nevertheless, the UI specification contains a screen that asks for username and password, for the sake of evaluating the availability, look and ease of use of the relevant form components.
- Every picture can only be added once, the user cannot choose how many printed copies he wants to have.
- SSL security is not used with the web service. However, missing support for secure connections would result in a deduction in the evaluation because it is a highly important feature for many business applications.
- No detailed error information or error code is returned by the web service in case of expected errors, instead an automatically generated error page and a reasonable HTTP status code is used.

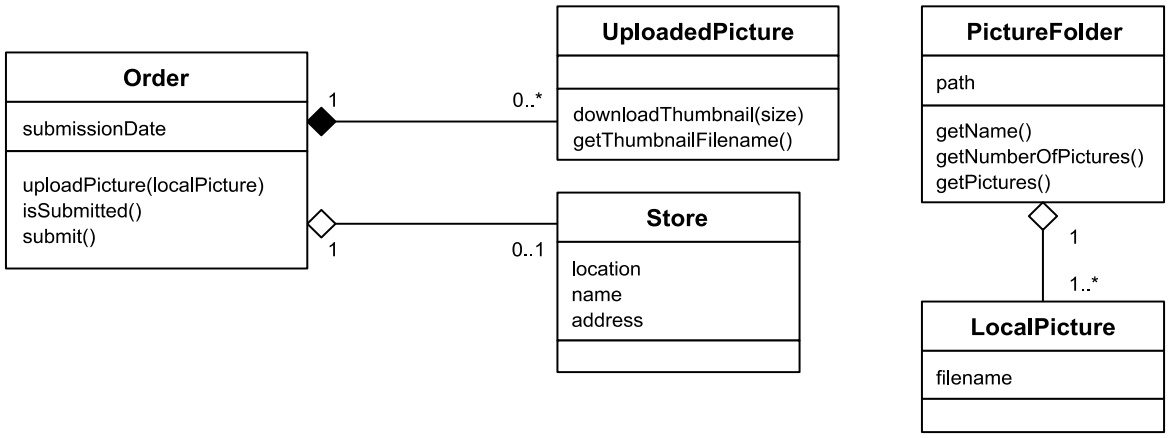


Figure 16: Analysis Object Model of the entities handled by the sample application

The above figure gives an overview of entities handled in the sample application and for communication with the web service:

- An order can contain many pictures which are stored by the web service (*UploadedPicture*). Each order may be in "submitted" or "open" state. Submitted orders are called "old orders" and are assigned a store where the user can pick up his printed pictures.
- For uploaded pictures, the mobile application can download thumbnails in a specified size for efficient display. They are cached locally as files (*getThumbnailFilename*).
- Stores have a name and address that are displayed in the application. The *location* attribute is utilized by the web service to find stores close to the user's location.
- Objects of type *PictureFolder* represent folders on the device's file system that contain pictures (*LocalPicture*). The methods for retrieving name and number of pictures are needed to show a list of picture folders to the user.

In the following, the necessary web service related functionality is described, for which I go more into detail on the actual definition of entities. As mentioned before, the web service is a fixed component of the system and can be assumed to work as defined. It uses REST-inspired access to resources and JSON as data format for responses (in the case of success). Details of the web service interface can be found in appendix A.

The application must retrieve information for display on the screen; this comprises order details (e.g. ID for each picture in the order, plus an ID for the order itself), picture thumbnails in a specified size and the list of orders. That data is fetched from the web service.

The application must be able to retrieve a list of old orders and the current order from the web service. An order consists of:

- A unique order ID, a positive integer number (assumed to be $< 2^{31}$)
- Date of submission as string, or `null` if the order was not submitted yet (i.e. is the current order). The string is formatted after the recommendations in *ISO 8601* which includes the time zone but no second fragment, e.g. "2012-04-14T23:19:00+01:00" for 11:19 p.m. on April 14th, 2012, in the time zone UTC+1. The application must be able to parse the date and time of such strings and display them.
- List of included picture IDs of the same integer type as the order ID
- Store ID (same integer type) or `null` if order was not submitted yet

With the provided picture IDs, a thumbnail of a specified size can be downloaded from the web service. Furthermore, the application also has the functionality of submitting pictures to the web service which automatically adds them to the current order.

The application allows the user to search for nearby stores for picking up the ordered pictures. An entered address or automatically determined latitude/longitude location (see description of order submission screen in the next section) can be sent to the web service which returns information about up to five close stores.

An order can be submitted, or in other words confirmed by the customer to be printed at a store, when the user confirms it and if at least one picture was added to it and the order is not yet submitted. Submission assigns a date and the selected store ID to the current order. These attributes can be used to determine whether an order is an old order or the current one – only the current order has no submission date assigned.

Apart from functionality related to making requests to the web service, the following features of the application must be considered.

At the first display of the screen that shows the list of orders (see next section), and in regular intervals, the application must retrieve the up-to-date list of orders and display it. The interval is defined as 60 seconds for testing purposes. If nothing changed between the cached and the up-to-date list, the user interface must not be altered or redrawn.

When an order is displayed, it is necessary to also show contained pictures as thumbnails (described in the next section). Therefore, thumbnails in an appropriate resolution must be downloaded from the web service. All thumbnails should be cached according to the web service's HTTP cache control headers. As a side note, the web service actually always returns "Expires: Sun, 17-Jan-2038 19:14:07 GMT", i.e. allows "infinite" caching. The requested size of the thumbnail shall be roughly the width or height of the device's display in pixels, choosing the greater value (restricted by the web service's parameter range, i.e. sizes from 5 to 500).

In order to present the user with a list of folders which contain pictures that can be selected for printing, the application must also be able to scan the mobile device's file system, e.g. the SD card, for JPEG pictures. A recursive search for filenames with the extension ".jpg" (case-insensitive) is enough to fulfill this purpose. If both a folder and its child folder contain pictures, they are treated as two distinct folders. In case a picture gallery API or similar functionality exists and allows automatic listing of pictures, it should be used.

5.3 Screens and functional requirements of visible features

MobiPrint has four main functionalities visible to the user:

- Listing old orders
- Adding pictures from the mobile device
- Showing the details of the current or any old order
- Submitting an order

The following diagram depicts the basic use cases for the customer. Any use cases of the printing store are ignored here and only the mobile application is considered, not the web service.

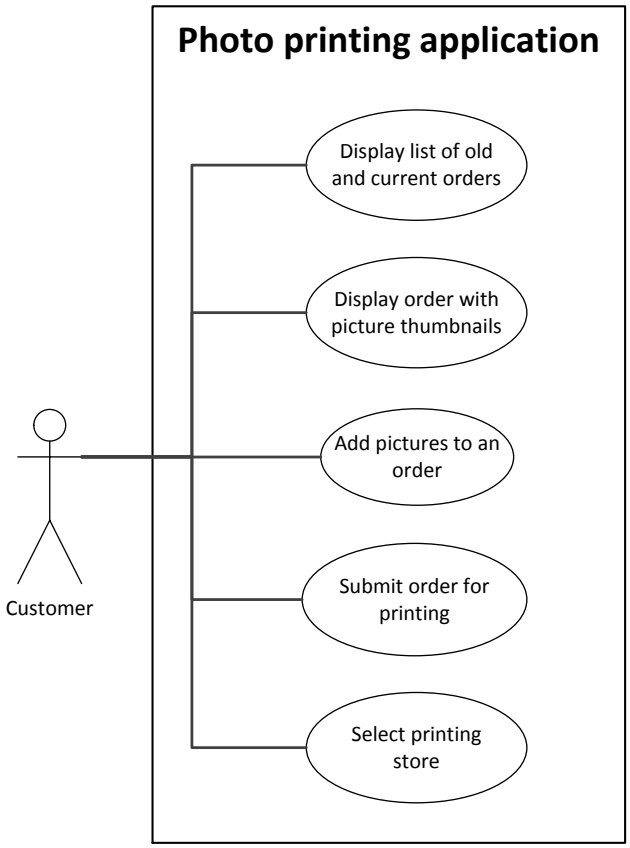


Figure 17: Use case diagram for the photo printing sample application

The application is split into multiple screens, each one (or more) belonging to one of these functionalities. The screens are described below. It must be possible to navigate easily between functionalities (two taps or fewer). In the following mockup figures (designed similar to iPhone user interfaces), this is implemented as a tab bar, for example.

The “old orders” screen shows the total number of old (i.e. previously submitted) orders. Each of them is displayed in a list, including the date of submission in the format “Tuesday, April 10th 2012” (superscript formatting optional) and the number of pictures contained in the order. The user can navigate to the detailed order screen from each list item. After the list of old orders was updated (loaded from the web service), it must be cached so that it can be displayed even if the device is offline. Figure 18 shows an example mockup for the old orders screen.

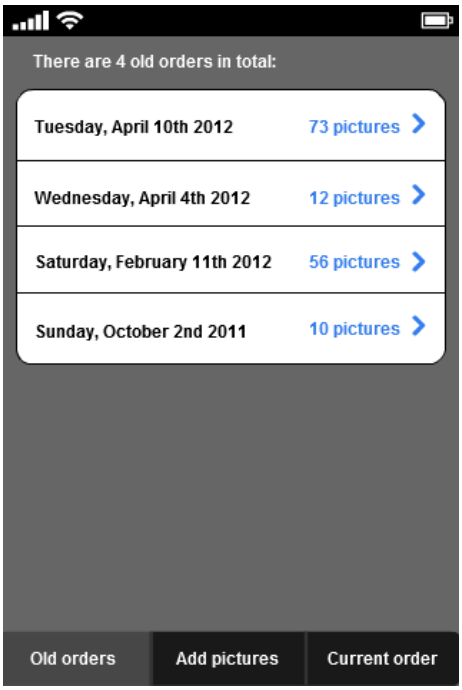


Figure 18: Mockup for “Old orders” screen

The application can display any old order or the current order in a separate screen (see figure 19 below). For all pictures that were added to an order, a matching thumbnail image is shown. The thumbnails are arranged as 9x9 or larger grid that should be automatically resized if the screen orientation changes. Each thumbnail should be scaled proportionally from the original picture and its displayed size (longer side) must be at least 1 cm on a mobile device. Since a thumbnail can only be downloaded from the web service once a picture was added to the order, the application displays the local picture file of uploading pictures as long as it did not retrieve a thumbnail from the web service. Together with each thumbnail, the state of the picture is shown (either “Uploading”, “Uploaded” or “Printed”) together with a matching symbol (image files given for each symbol). If the current order is

shown, an additional button labeled “Submit for printing” is displayed which switches to the order submission screen.

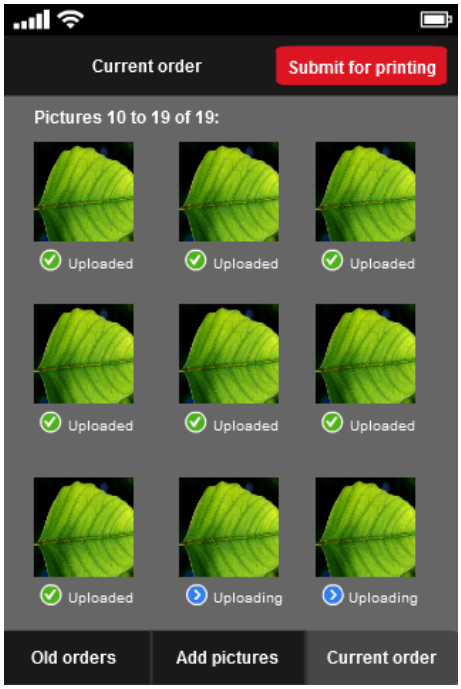


Figure 19: Mockup for “Order detail” screen (example for current order)

The “add pictures” functionality consists of two screens (see figure 20 below): In the first screen, all scanned folders that contain pictures are listed with the folder’s name and the number of contained pictures. The user can navigate to the pictures screen from each item. That screen shows all pictures in the selected folder. Each picture is displayed using the full width of the device in its current orientation (reasonable margins of less than 1 cm allowed). If necessary, larger pictures must be downscaled to that width, while smaller pictures should be displayed centered and in their original size. Below each picture, the user can select and deselect whether the picture shall be added to the current order. At last, the screen must include a button labeled “Add to order” – when clicked, the selected pictures are added to the current order and the application switches to the detailed order screen, showing the current order. The switch is done immediately, and the pictures should be uploaded in the background, with the state “Uploading” shown as mentioned before.

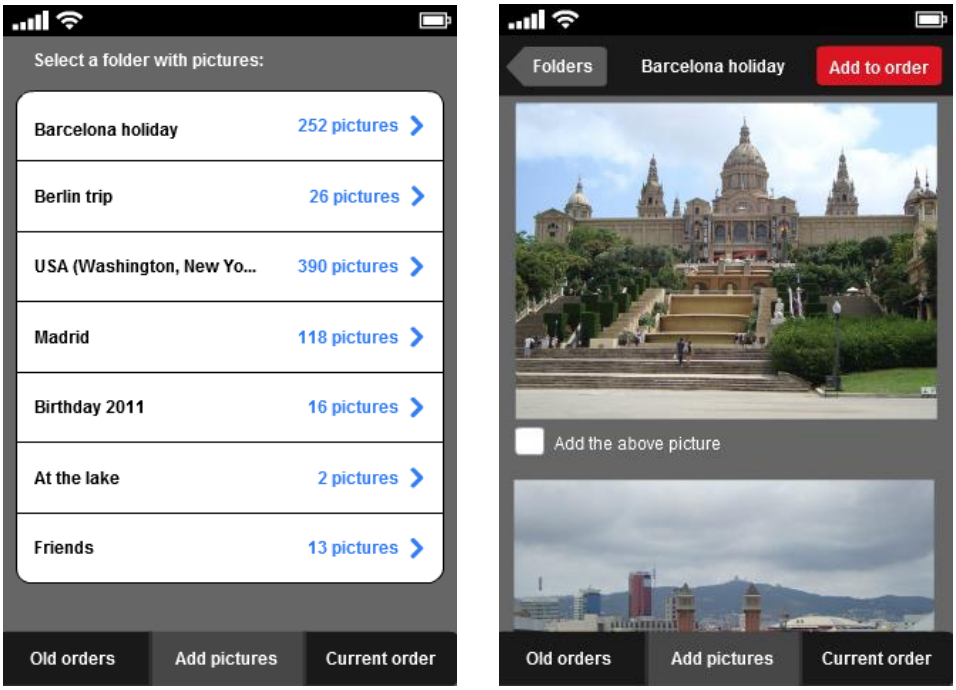


Figure 20: Mockup of "Add pictures" screens

At last, the order submission screen (see figure 21 below) shows the number of pictures contained in the current order, a selection of the pick-up location (where printed pictures can be collected), a form for the username and password and a button labeled "Submit order", including a confirmation checkbox that must be ticked before the order can be submitted.

The pick-up location should be automatically determined by the available location services of the device. GPS shall be the preferred option. If the GPS module is not enabled, the application should use other mechanisms as available, e.g. detecting the position by WLAN hotspots in the environment [18]. If no location is available, the previously entered address must be used automatically (requires storing this setting permanently). The detected address is filled in automatically in the corresponding search field. Whenever this field was changed (manually or automatically), the web service is queried for stores near the given address and the results are shown in a list, containing the name and address of each store. If the device has no Internet access or another failure occurs, a short message or a message box is displayed instead of the list of stores. The user can select exactly one of the proposed stores. If and only if exactly one store is selected, both username and password are entered and the order confirmation checkbox is ticked, the "Submit order" button can be used to

[18] Often uses Google's location services (https://support.google.com/maps/bin/answer.py?answer=1725632, accessed: 2012-04-17) or similar databases

send the order to the web service. In case of success, a message box or similar notification is shown, and the application switches to the screen with the list of old orders, which shall be updated at that time. Should a failure occur when submitting the order, the same kind of message box is shown to the user, reporting an error (detailed error description not needed) – the screen is not switched in this case.

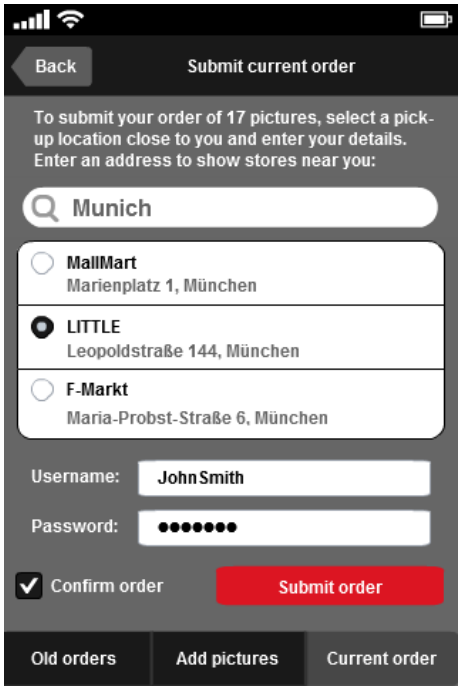


Figure 21: Mockup of "Submit order" screen with three suggested stores

5.4 Non-functional requirements

A few non-functional requirements apply to the implementation of the sample application. The following conditions must be met (categorized according to the FURPS+ model):

- **Reliability:** The application should not crash. Any expected or unexpected errors must be shown as notification, message box or other kind of displayed or logged error message.
- **Performance:** Switching between screens must not take longer than one second even on slow devices. This is tested with a *Huawei Ideos X3*, an Android 2.3 mobile phone with low-end specifications (600 MHz, 256 MB RAM, 320x480 display resolution).
- **Usability:** All displayed text must be readable. This is defined as the letter 'W' having a height of at least 0.14 cm. For example, this would be a height of about 9 pixels on an *iPhone 3* display (163 ppi) or 10 pixels on the display of the Huawei device mentioned above (180 ppi).

- **Usability:** The user must be able to navigate easily between functionalities (as defined in section 5.3), i.e. with two taps or fewer.
- **Usability:** "Look and feel" of the application should be close to the native platform look if the framework supports it.
- **Supportability:** The language of the application must be English. Strings that are shown in the user interface must be used in a way that they can be easily translated to other languages (translation and internationalization support influences the score for the supportability category).
- **Supportability:** The application must run and fulfill all above requirements on the Android emulator (any Android 2.x version) and the test device *Huawei Ideos X3* (Android 2.3). In the case of the iPhone SDK, the application must run on the iPhone 5.1 simulator and on the test device, an *iPhone 3GS*.

5.5 System architecture and web service

The following diagram shows how the whole system is deployed and which physical services are used by the mobile application. Since the web service is considered as already existing component, it is not relevant for the comparison and its exact implementation is not presented.

For communication, a *REST* approach was chosen, i.e. read access to entities (like orders) is available through HTTP *GET* requests, while picture uploads require a *PUT* request, for example. The *JSON* data format is utilized to represent entities because it is a simple yet concise format and parsing is possible with many libraries and thus should not be a problem for the sample application implementation. As a minor notice, during development the web service was hosted at a fixed address on a private virtual server on the Internet.

On the mobile device, the application may access the file system to scan for pictures, store and read configuration files, or to cache thumbnails. Location tracking features like GPS are used to find stores that are close to the current position. The Internet connection is used to connect to the web service in order to retrieve the list of orders, upload pictures, etc.

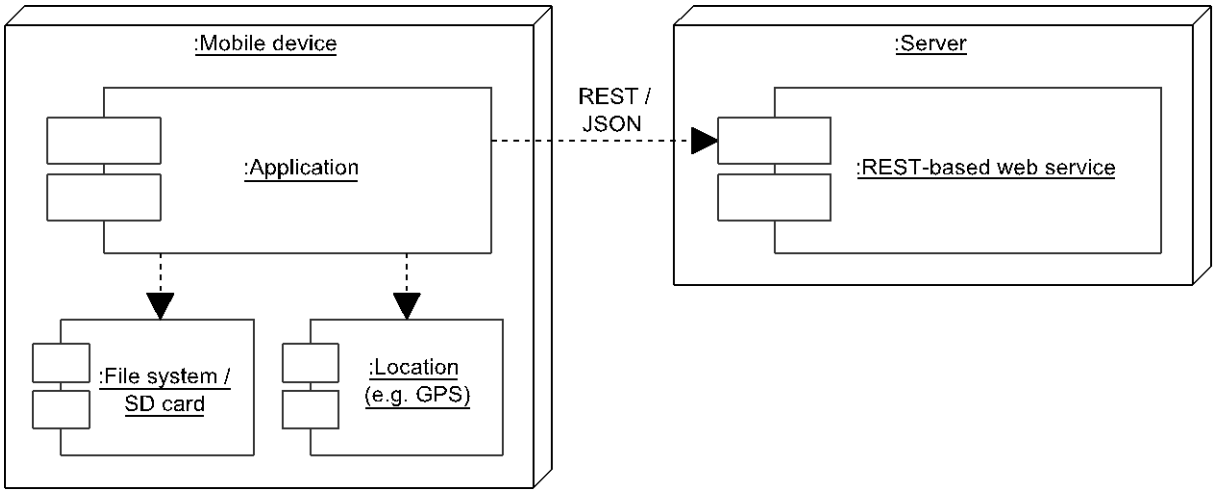


Figure 22: Deployment of the system for the sample application

5.6 Application summary

Relating the above application definition to the intent of this thesis, it should be noted why this kind of sample application was contrived for the evaluation of frameworks. As mentioned before, business applications – and in particular the mobile variants – have certain requirements. With MobiPrint, the comparison is based on a sample application that tests a wide range of functionality and capabilities of the frameworks:

- **Device and framework functionality**
  - Location discovery (GPS and other)
  - File system access
  - Web service requests via HTTP, including binary file upload
  - Caching
  - Screen orientation changes
- **UI functionality**
  - Tab bar or similar navigation method
  - Different layouts and components in the screens (grid layout, form elements and layout, lists, buttons, checkboxes)
  - Performance of navigation and user interface in general
- **Cross-platform support** (does not apply to the implementations based on the native SDKs of Android and iPhone)
  - Application must run on both Android and iPhone test devices
  - Support for applying native look of a platform

Moreover, MobiPrint is fairly complex for a sample application and thus also requires clean coding practices from the developer, such as splitting up source files where possible instead of creating a single source code file. Although not all criteria (cf. pages 23ff) can be tested with such a small and simplified application, many development-related criteria can be evaluated. The remaining criteria, for example compatibility with app stores, professional support options or activity of framework development, are assessed and compared as well, but not based on experiences from implementing MobiPrint.

5.7 Work packages

With the MobiPrint application now fully defined, the next step is to divide the application into work packages and put them in a reasonable order. This simplifies the development process and shortens time to completion. Another benefit of advance planning is that the time for defining the work packages does not add up to the measured development time of the application (see p. 33), making results slightly more comparable.

Implementation is pursued in the following order, if possible:

- Installation of framework
- Creation of “Hello World” project, assignment of project name “MobiPrint”, ready-made icon, license text and application description, starting the application on emulator. Time measurement does not include creating or applying splash/loading screen pictures because this feature might not be available for every framework.
- Setup of screen navigation component (e.g. tab bar). The user shall already be able to switch between the (temporarily empty) three screens “old orders”, “add pictures” and “current order”. If the framework or its IDE already comes with a matching template for navigation, it may be used, resulting in smaller setup time.
- Setup of internationalization (translations only)
- Development of old orders screen
    - Screen design
    - Single retrieval of the order list from the web service
    - Population and display of list items (includes parsing of submission date string and display in the format “Tuesday, April 10th 2012”)

  - Caching of the list and storing the last update time (e.g. in a configuration file), screen changed to use the cached list
  - Regular retrieval of the order list and display if the list changed
- Development of order detail screen
  - Screen design
  - Retrieval of thumbnails as background task (appropriate thumbnail size must be chosen according to display)
  - Caching of thumbnails
  - Way to store state of pictures that are currently uploading, display of correct state for each picture
  - Display of cached thumbnails, fallback to local picture if thumbnail was not downloaded yet
- Development of “add pictures” screens
  - Design of the screens and navigation between them
  - Scanning of file system or picture gallery
  - Display of folder list on first screen
  - Display of picture list on second screen
  - Background task functionality for picture uploads
- Development of order submission screen
  - Screen design
  - Location discovery, storing and reading the previous location from the configuration
  - Retrieval of information about nearby stores from the web service and display in list
  - Order submission
- Final testing and bug fixing: This includes going through all requirements and checking whether each is implemented correctly, and verifying that the application works on the test device(s).

The time needed for each of the major and minor steps above is measured exact to the minute, however only the total time will be mentioned in the appendix for reference. Unrelated times, such as for downloading installation files of a framework, are neglected.

6 Evaluation

This chapter represents the main work of this thesis. The three selected cross-platform solutions and the Android and iPhone native SDKs are evaluated one-by-one in separate sections, using experiences from the implementation of the sample application described in the previous chapter, and from facts and sources found through further research about the five solutions.

In each section, I will first explain my overall impression and experience during the implementation of the sample application *MobiPrint*. Next, I mention the total time that I required to develop the application, without further comments for the reasons stated before (cf. page 33). Design and technical decisions specific to the implementation of *MobiPrint* with a framework are outlined in a separate subsection, which also contains an example screenshot. Finally, each of the five solutions is evaluated with a separate score in each category, together with an explanation and reasoning for the chosen score, based on whether the respective criteria were fulfilled (overview on page 31).

An overview of the results of the evaluation scores can be found in chapter 7, in which I interpret the results and give recommendations regarding whether a cross-platform framework is worthwhile, how cross-platform solutions compare to native development, and other aspects.

A short note on the implementation: I first implemented the sample application using the three cross-platform solutions and only then ported them to work on iPhone devices. Since I planned to show folders containing pictures by scanning the SD card's file system, it was implemented this way and worked well on Android. Only later I noticed that iOS runs applications in a protected environment ("sandbox") so that they are not able to access the whole file system. When porting these three implementations to iOS, I decided not to port the selection of pictures for timely reasons. This was purely my own technical design mistake but is a good point to mention because obviously also cross-platform applications are influenced by operating system level restrictions.

6.1 Titanium

6.1.1 Experience

I started developing the sample application without prior knowledge of Titanium's API and offered functionality, but with some previous experience programming in JavaScript for web development. Since Titanium does not use HTML (web views are possible however), the process of creating an application with this framework had to be learned from scratch, using the available documentation, answers on the official developer forum and screencasts that were freely available after registration on the Appcelerator web site. Any information related to the sample application implementation refers to Titanium in version 2.0.1 (unless specified otherwise).

Implementing the sample application was generally very straightforward due to the simple architecture of a Titanium project: since MobiPrint is not overly complex, I implemented each screen and background operation (e.g. uploading pictures) in a separate file. No architecture pattern such as MVC is enforced. Interactions between screens and/or operations, such as refreshing the list of orders after the current order was submitted, are greatly simplified by the event system that offers a publish/subscribe pattern. In said example, the screen displaying old orders could simply listen to the event "force-order-list-update" which is sent by the order submission screen. This has the advantage that a screen that wants to interact with another screen only sends out an event and does not have to store an object instance of the other screen – in case the other screen is not loaded yet, the event will not be handled. For example, this can speed up the application start because classes (here: screens) can be loosely coupled and do not have to be instantiated immediately.

Memory issues were one problem that occurred. Since uploading files have to be shown in the current order screen, thumbnails should be created to save memory and avoid crashes. But downscaling picture files led to memory errors as will be described below. Instead, the original, possibly large pictures are displayed directly while uploading. In this case, the pictures could not be displayed with the correct aspect ratio [19] anymore because setting one

[19] Ratio between width and height, "correct aspect ratio" means width and height are scaled by the same factor so that the picture does not look squashed

side of the display rectangle to a fixed size did not work as expected – the picture was displayed larger than the fixed width and still with a wrong aspect ratio. Setting both width and height correctly was also not possible because the original picture dimensions would have to be known, and reading these dimensions from the picture files already causes said memory errors. Apart from this, other UI and layouting issues were discovered, such as not being able to make a table have exactly the height it needs to show all rows. A later minor version update of the Titanium SDK (2.0.1 to 2.1.0) even caused different defaults, e.g. the table row height changed from a platform-specific height (e.g. 40 pixels on the Android test device) to only take as much space as necessary.

I tested the application on iOS only after it was finished for Android. In retrospect, it would have been better to test early on both platforms because both user interface and other functionality can behave differently (even if not documented) and the effort to make the application work on iOS may have been smaller. However, I must say that the amount of time spent for finding and correcting these differences was not very high. Most of the effort for porting pertained to UI layouting differences, e.g. layouting that displayed correctly on Android but needed additional sizing parameters on iOS.

Altogether I must say that working with Titanium was very easy: the JavaScript language in combination with the mostly well-designed API and the event system make small applications easy to implement and the documentation can be considered very complete. I did not like that the APIs are not always truly cross-platform and often no attempt is made to provide a common interface for functionality that is very similar on both supported platforms (Android and iOS). More details are described in the per-category evaluation below.

6.1.2 Sample application implementation details

MobiPrint has a fairly simple structure and consequently needs no complex source code architecture, especially since JavaScript code is supposed to be fully reusable with the Titanium framework (except for platform-specific functions). Therefore, I decided to follow a simple approach as given in the default project template for a tabbed application: the main entry point, i.e. the first developer-defined JavaScript code run by the interpreter, decides whether the device has a large (tablet-sized) or small (phone-sized) screen and creates a

different main view, respectively. This is a good preset because with larger screens, additional navigation controls could be shown permanently, such as a side bar. For the purpose of evaluation, only phone screens were considered and this decision was removed.

Titanium includes a function that loads a module and returns it as an object. Such modules must conform to the *CommonJS* module specification whose main purpose is to let a module return exported functionality (e.g. a single class) in an object instead of setting a global variable (which may cause naming conflicts). Apart from the advantage that no global variables are involved, this mechanism allows loading modules only when they are needed (lazy loading). If only part of the program code is needed at first, application startup may be accelerated because less memory and time is needed for parsing and executing JavaScript code.

The selected project template puts each screen implementation in a separate file (module). I followed the same pattern for aforementioned simplicity reasons. Communication between screens, such as transitions, tab selection changes or updates of displayed data is done with Titanium's event handling mechanism that offers a publish/subscribe approach. One or more functions can be defined to be called in case a certain named event occurs. Events can be triggered from anywhere in the code, and may pass further information to the callback functions. As an example, an event called "update-thumbnail" is used by the MobiPrint implementation to let order screens know when a thumbnail was successfully downloaded and the placeholder image can thus be replaced by the thumbnail image from the file system.

As another hint on implementation details, it is notable that through the use of JavaScript, many libraries that are typically used for web development can also be used with Titanium unless they are specifically web-related (using DOM manipulation, for instance). The *moment.js* library [20] is used to parse date strings coming from the web service (*ISO 8601* format) and to calculate the difference between dates.

[20] http://momentjs.com/, accessed: 2012-04-19

The following screenshot shows the finished implementation on an Android device and the iPhone simulator:

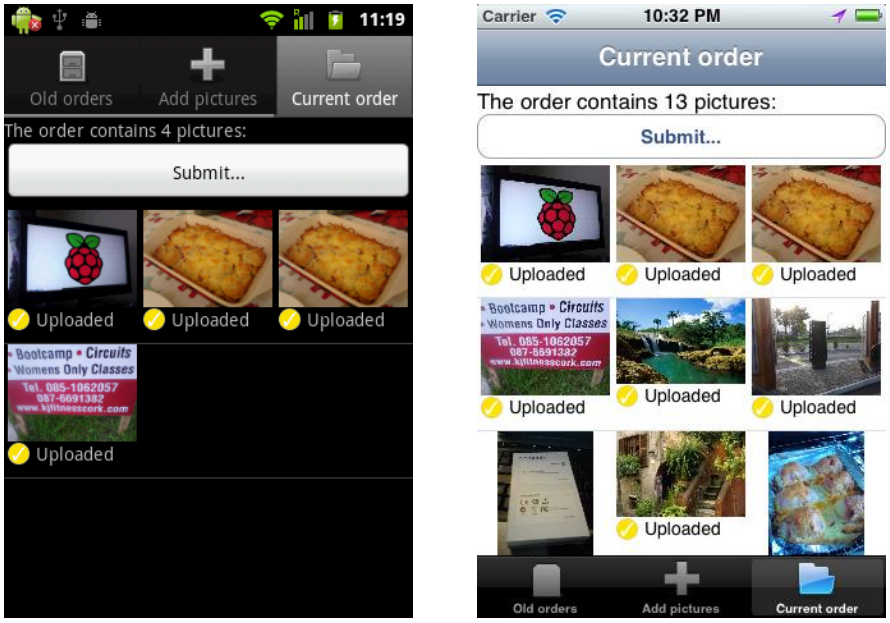


Figure 23: Screenshot of Titanium-based implementation of MobiPrint on Android/iPhone

6.1.3 Category scores

Functionality

The Titanium API offers a variety of non-UI functions. Most of the functionality typically used in mobile applications is available, such as location tracking, data and file access, an asynchronous HTTP client, JSON/XML parsers or retrieval of device information (e.g. screen size). Hardware sensors and actuators, except for NFC, are available. APIs are well designed and separated into modules that are part of the global object named `Ti` (e.g. `Ti.Accelerometer`).

Unfortunately, the framework design does not attempt to abstract platform differences for features that are platform-specific. For example, the `Ti.Geolocation` module has a function `setPurpose` used to define a short text that iOS displays to the user when asking for the permission to use the current location. This is not supported on other platforms like Android, the only other mobile device platform supported by Titanium. A developer might expect that this function is always implemented – but doing nothing on Android. However, the function does not exist on Android and calling it will result in an exception (undefined function). One could instead assign the property (e.g. `purpose = "Navigation"`), but then any future renames to the function/property will not be noticed because properties of

JavaScript objects can be set even if they do not yet exist. In contrast, other features that are only available on a single platform are located in separate submodules like `Ti.App.iOS.BackgroundService` (for running a service when the application is in the background) or `Ti.Geolocation.Android` (Android-specific location API). Examples like this prove that the API design is slightly inconsistent in several places.
Since Titanium does not offer common interfaces for some functionality, developing truly cross-platform is not always possible. For instance, both Android and iOS allow to run a service even if the application itself is running in the background. An exemplary purpose would be the regular check for new notifications or e-mails. Titanium supports this for both platforms, but developers must use two different modules `Ti.Android.Service` and `Ti.App.iOS.BackgroundService` instead of a common interface.

Regarding the quality of the framework implementation, it should be noted that most non-UI functions seem ready for production use and only few minor issues occurred. Especially porting the sample application to work on iOS revealed some implementation differences. As an example, the function `Ti.App.Properties.setList` persists a JavaScript array for a certain key in a key-value store. On Android, storing an array containing the current time and an array of orders (one JavaScript object each) worked fine, but the iOS variant is implemented in a way that the array cannot contain a nested array. The documentation does not state any platform differences for this function.

Extending an application with native code is very simple: Titanium Studio offers a special module project type as template for a platform-specific module. The compiled output (e.g. a JAR file for Android) and a configuration file represent a Titanium module. They can be included via the project settings in Titanium Studio and their interface can be imported in the JavaScript code with a call to the `require` function. Appcelerator also offers a market for free and paid modules at https://marketplace.appcelerator.com/ [21]. Features like barcode scanning, payments or UI design templates can be purchased (or downloaded for free) and added with little effort.

[21] Accessed: 2012-02-12

In addition to expected features, the framework also supports Appcelerator's *Application Cloud Services* which can be used for free to a certain extent (250000 API calls per month [22]). They include push notifications, key-value data storage, integration with a Facebook account, sending of e-mails and other services.

Concluding the score for functionality, Titanium has almost all necessary functionality including access to device capabilities. For data access, simple key-value storage and SQLite support are available, but without any higher level abstraction such as an ORM (only raw SQL queries). Native extensions, available modules (e.g. barcode scanning) and supplementary APIs offered by Titanium (like cloud-based services) are a good addition for being able to build many types of applications. Because of the fact that many interfaces are not truly cross-platform and sometimes different implementations for the supported platforms (Android and iOS) are necessary, a score of 3 points is assigned.

Usability features

The framework relies on native user interface components, i.e. components that are also used by native applications, and offers common types: buttons, lists, labels, text and form fields, etc. It also offers several components that are only available on iOS, such as a "cover flow" view (animated and swipe-controlled view for scrolling through images). Embedding HTML pages using a web view is also supported.

Titanium offers a layouting system with several different coordinate units (e.g. pixels, density-independent pixels, percentages, centimeters) and sizing strategies applicable for width and height of a component (automatic, necessary size, fill parent, relative to parent borders). The user interface is generated in JavaScript code and Titanium Studio includes no visual designer [23]. This proved as good and simple approach for constructing the sample application's screens, except for issues with getting correct sizes in certain cases (see below). The missing visual design feature may however become problematic for implementing complex user interfaces (not tested, so this is only an assumption). Almost all UI components are customizable in terms of colors, font, size etc.

[22] http://www.appcelerator.com/plans-pricing, accessed 2012-08-22

[23] The third party software ForgedUI (http://www.forgedui.com/, accessed: 2012-08-22) claims to integrate directly with Titanium Studio, allows visual UI editing and generates code automatically (compatible with the Titanium framework). I did not test this software.

UI functionality seems to have even more platform differences than described in the functionality section above. Since MobiPrint was only ported to iOS after it was finished for Android, some issues arose, mostly with layouting. The most complex screen, for submitting an order, consists of a search bar, a dynamically sized list for selection of a store and a form for filling in username/password and confirming the order. Many of the components that were displayed correctly on Android did not show up on the iPhone. The solution was to add appropriate width and height attributes. This means that the default behavior is not the same on the two platforms. Several other layout and UI-related issues occurred during development and the port to iOS – for example, a table could not be configured to use exactly the necessary height for all rows, or both the default screen background color and text color for a label is black on iOS (resulting in invisible text).

A small number of features are also still in development. One example from the documentation is the search bar component: "On Android, there are several issues with the current implementation: The cancel button does not work. It does not clear the search bar text or close the onscreen keyboard." [Appcelerator2012b]

In summary, Titanium offers a simple and sophisticated approach for constructing and layouting screens. Even though platform differences can create layouting problems, this would have been mitigated through early testing on both Android and iOS, and with more knowledge about required parameters of the layouting system. Since native components can be used directly, the subjective impression is very good and no extra effort is involved to design an application with a native look. Gestures like swipes can be easily detected by adding an event listener for the respective gesture – this enables customization of views and extended features like pull-to-refresh. Therefore, the score for usability features is 4 points.

Developer support

The framework is typically downloaded by installing *Titanium Studio* which automatically downloads the latest framework version. The IDE is based on *Aptana Studio* (in turn based on *Eclipse*) and as such familiar for many developers. Titanium Studio requires signing in with an account on http://my.appcelerator.com/ [24] before displaying the IDE. Since the

[24] Accessed: 2012-05-25

application and the UI is completely built in JavaScript code (apart from XML configuration and translation files), any editor suffices.

Regarding debugging capabilities, Titanium Studio supports all expected functionality: setting breakpoints in the JavaScript code, inspecting and changing values during runtime and an integrated view for the log output of both the build process and application.

The documentation is available online and updated with each release of the framework. It includes an API reference, guides (setup, example applications, native extensions, Titanium Studio), tutorial screencasts and presentations. During development of MobiPrint, the API reference was most helpful, since it offers a complete documentation of Titanium's APIs, many of them coming with example code where necessary. For each class, all properties, methods and events (for UI components, e.g. click or swipe event) are listed and explained shortly.
However, descriptions of methods and their parameters are often very short and in several cases missing needed information. For example, the description of the function `Ti.Filesystem.createTempFile` states that it creates a temporary file and returns the file handle, but does not mention where on the file system the file is created and when and if it will be deleted automatically (e.g. immediately after being closed, after a certain time, only manually, determined by the operating system). Other cases exist, such as wrong or incompletely described parameter or return values and types – the return type of `Ti.Filesystem.File.getDirectoryListing` is only described as array of strings, not what the strings represent, for instance (they are in fact paths relative to the provided directory).

With the easy installation, basic debugging support, mostly complete documentation and small learning effort (only one programming language), Titanium earns 4 points in this category.

Reliability & Performance

Titanium relies only on the JavaScript language and packages applications together with a fast interpreter. The subjective performance difference to native applications is only slightly noticeable. Effects and transitions are mostly smooth and the startup time of the application is acceptable even on the slow Android test device. The problem with JavaScript is that

multithreading is not supported and thus long-running background tasks should be avoided or put into a separate background service. If this is not done right, the user interface will not react and the operating system may tell the user that the application is not responding and offer to stop it. Titanium offers asynchronous HTTP request handling, for example, in order to avoid such issues – the file access API is synchronous, however.

During development, only few repeatable crashes were encountered. One was due to normal use of the API – updating width/height of an image at the same time – and was solved by just changing the values separately. Another problem was memory: while a picture is uploaded, it should be shown on the current order screen. Displaying many full size pictures would read all of them into memory, so a thumbnail should be created instead. On Android, even if the uploading picture files were read one after another, converted into thumbnails and their object references invalidated, the application crashed because of too little memory. As a result, the pictures could not be shrunk and the large files are displayed directly (which did not cause a crash in my tests). This did not happen on iOS because of higher limits in application memory. Titanium needs functions to handle memory management better – in this case, memory-efficient loading of thumbnails and conversion of memory errors to JavaScript exceptions (instead of letting the application crash) would be helpful. Manually triggering JavaScript garbage collection could be another option.

Titanium provides almost native speed and deserves 4 points for the performance rating. On Android, the installed MobiPrint application needed 1.39 MB, which is the largest package size of all five tested solutions, but surely acceptable. Mentioned memory issues are specific to the sample application to a certain extent but should be considered, depending on the application type, before deciding for this framework. Because of these problems and the nontrivial memory management in general, which is missing functions like efficient loading of pictures or creation of thumbnails, 2 points are assigned for reliability. Therefore, an average of only 3 points is assigned for the whole category.

Deployment, Supportability, Costs

Two mobile platforms are fully supported by Titanium – *Android* and *iOS*. Additionally, HTML-based web applications can be generated from a project [25]. Support for *BlackBerry* devices is included in Titanium Studio but not mentioned as supported by Titanium. In a press release from May 2012, Appcelerator stated the *BlackBerry 10* platform would be supported in the future [Appcelerator2012a]. Applications created with this framework are accepted on both Google's and Apple's app stores.

The source code for Titanium is open source and freely available under the *Apache License 2.0*. From the change history, the framework seems to be only developed by the company Appcelerator with close to no contributions from external developers. Releases of minor version updates are frequent and development is very active with typically more than 50 commits per week. For professional support from Appcelerator, different levels of support are offered [Appcelerator2012c] with different prices on a per-application basis. The "Standard" plan includes web and chat support at 2 day response time from 8am to 5pm during weekdays. Also, the number of allowed calls to cloud API and storage space is double as compared to the free plan. Two other plans exist, both including phone support (also on weekends), shorter response time (1 day / 8 hours) and advisory time. All three paid plans allow developers to access "Enterprise Extensions". Both prices and a list of these extensions are not available on the web site, and Appcelerator did not respond to enquiries about this information.

The Titanium framework also has a built-in analytics feature that collects usage information which can later be analyzed, e.g. number and location of users or which features in an application are the most used. Analytics are free up to a certain amount of "data points" (not specified further).

In addition to direct support, Appcelerator also provides an online forum in which all registered users can participate. Because Titanium is open source and generally free to use, many developers are using the framework and cause a high amount of questions, resulting in few questions being answered at all and with a working solution. Some questions are

[25] Not tested, irrelevant because web applications do not support all device functionality

taken on by Appcelerator's developers themselves which makes the question more likely to be answered.

Automated building of projects is unfortunately not consolidated – for both platforms, a separate Python-based build script exists, but there is no documentation on how to use them. A third party solution for managing installed Titanium SDK versions and building projects exists [26] (not tested).

Another important point of supportability is internationalization support. In this aspect, Titanium offers a built-in function for simple translations that uses a key-to-value mapping from an XML file of the respective language. Extended features such as pluralization [27] are not supported. Since Titanium works with most JavaScript libraries that do not rely on web functionality (e.g. DOM manipulation), third-party libraries could be used for this purpose. For example, the library *moment.js* was used in the sample application for parsing date strings.

In summary, Titanium is open source and both the framework and Titanium Studio are offered for free and basic support via the online forum is free as well. The most popular platforms – Android and iOS – are supported and another may be added soon. Also, the framework is under active development. The only drawback are automatic builds which are not offered in a consolidated and well-documented way. Hence, a score of 4 is assigned because most expectations are fulfilled.

6.1.4 Conclusion

Titanium is freely available, easy to install and offers an integrated development environment with Titanium Studio. Concerning the variety of features, Titanium fulfills most requirements to device functionality, user interface creation and other APIs like simple data storage or asynchronous HTTP requests. The API is mostly well-designed and documentation is nearly complete with missing information in few cases. Extensibility is possible both with native extension modules and external JavaScript libraries. With the use of only one programming language and very little configuration, the learning effort can be considered

[26] http://russfrank.us/2012/04/25/how-not-to-use-titanium-studio/, accessed: 2012-08-19
[27] See glossary, translation of singular/plural forms of a word which can differ greatly between languages

fairly low. Help is available for free in the online forum (however solutions can be rare for very specific questions), and paid professional support is offered by Appcelerator as well.

Supporting more than one platform can be difficult because the framework is not designed to be fully cross-platform in terms of offering abstract interfaces. Instead, developers have to make distinctions between platforms in the code. Different behaviors, particularly regarding user interface layouting and components, can raise issues when porting an application. Therefore, early testing with all desired platforms is recommended. Even embedded web components are supported, enabling development of fully or partially web-based applications that are able to use native features.

Altogether, Titanium fulfills many expectations. Through the very active development, it may be possible that mentioned issues are addressed in the future – more platforms are planned to be added (e.g. BlackBerry 10) and the framework will have to behave more consistently on different platforms. The final score, resulting from dividing the sum of weighted scores of the five categories by the sum of the weights, is *3.6*.

Titanium	*Weight*	*Score*
Functionality	4	3
Usability features	3	4
Developer support	2	4
Reliability & Performance	3	3
Deployment, Supportability, Costs	5	4
	Rounded final score:	3.6

Table 2: Evaluation results for Titanium framework

6.2 Rhodes

6.2.1 Experience

For developing MobiPrint, version 3.3.2 of the Rhodes framework was used. It was necessary for me to learn the *Ruby* programming language from zero and to understand the application architecture. An application is basically a web application and Rhodes offers a web server and framework for serving requests. The difference to a normal web application is that the server is running locally and in addition, code running on the server side can inject client code (JavaScript run in the web component) or redirect at any time, meaning it always has access to the frontend. Only backend code (Ruby) has access to device functionality. This untypical approach can be unclear to web developers: the backend is not actually physically remote and more assumptions can be made. For instance, no network connection errors can occur because all HTTP requests are sent to the local server that is embedded in the application.

A simple Rhodes application consists of a single HTML file that includes the *jQuery Mobile* library (written in JavaScript). Rhodes leverages jQuery Mobile to load screens by sending HTTP requests, inserting the result into the DOM tree and optionally applying a transition effect like fading out the previous page and fading in the new one. The HTTP requests go to the local server which, depending on the URL, calls the respective method of the controller class. That method can return a filled HTML template – such templates are based on *ERB* (*Embedded Ruby*), best known from the web framework *Ruby on Rails*. Many other popular web frameworks use templates, too. Therefore, web developers should get accustomed with this approach easily. jQuery Mobile also offers page styling with special HTML attributes, for example `<a data-role="button">Back</a>` would apply a button-like appearance as soon as the page is loaded. More details will be explained in the rating for the usability features category.

As mentioned above, the split between backend and frontend code can be hard to understand, and I consider both the architecture and the necessary learning effort for two programming languages slightly problematic – Ruby is used for implementing controllers and JavaScript has to be used in screens with any application that does not simply display static data. Since only backend code (Ruby) can access device functionality or any native extensions, dynamic updates always require the backend to serialize data, for example in

JSON format, and the frontend code to parse the data and display it. This is an unnecessary extra step.

But there is also an advantage: Rhodes applications are multithreaded, i.e. execution of JavaScript code in the web component and Ruby code in the backend are independent. Long running background operations like data synchronization with a web service are thus not a problem [Allen2010].

In summary, my experience with Rhodes was mediocre regarding Ruby and backend development, while working with HTML/ERB templates and jQuery Mobile to design screens was very easy. Since the official simulator (*RhoSimulator*) does not support device capabilities like location tracking, these features were tested on the Android emulator. In contrast to the very quick startup speed of RhoSimulator, packaging and installing the sample application for Android took very long each time and debugging Ruby code is unfortunately only possible in RhoSimulator. Rhodes is very well suited for cross-platform development: the port to iOS only needed changes in the user interface because iPhone devices do not have a built-in back button like Android devices. No other changes were necessary, speaking for the good implementation of the Rhodes API and possibly for the choice of Ruby as a mostly platform-independent language. Unfortunately, not all requirements of MobiPrint could be fulfilled (explained below).

6.2.2 Sample application implementation details

Rhodes offers a simple MVC-like application architecture. Controllers are implemented in Ruby code and called when requests are sent to the local web server. The URL path defines which method is called, e.g. `/app/Order/show?id=5` would call the `show` method of the controller defined in a Ruby file in the directory `app/Order`. The sample application only has one controller, including actions for listing orders, showing order details and the order submission screen, for example.

Regarding models, a simple ORM called *Rhom* is included with the framework and can optionally be used. It stores data in SQLite or a similar database depending on the platform, and allows either schemaless definition of a model, i.e. any number of properties of any type can be added to a model instance, or fixed schemas. I decided to use the former variant for simplicity. Unfortunately, this led to several issues: the ORM automatically creates an ID field and appropriate values, so an own ID field must be defined if required (which I assume is the

case in many real data synchronization scenarios). Furthermore, all properties seemed to be converted to strings and the special value `nil` in the Ruby language was converted to an empty string. This was counterintuitive and led to freshly created order objects having a list of picture IDs, while objects read from the database had that list as string, making comparisons problematic. In the end, the ORM approach was replaced by a simple JSON file to store orders.

Rhodes provides several ways to pass data to a view that is already displayed: the HTML page may send an AJAX request via JavaScript and process the result, the controller can asynchronously evaluate a JavaScript expression (usually a function call with passed data) in the web view component or simply refresh the page. Since the list of orders only changes when the user submits another one or adds pictures, the third option was chosen to refresh the list screen whenever the orders changed.

As will be described in the following, some requirements could not be implemented. For example, no pictures can be displayed on the "add pictures" screen but the user may still upload the selected ones. Also, instead of a HTTP *PUT* request to upload a picture, the *POST* verb had to be used because Rhodes does not allow other verbs when uploading data in "multipart/form-data" form [28].

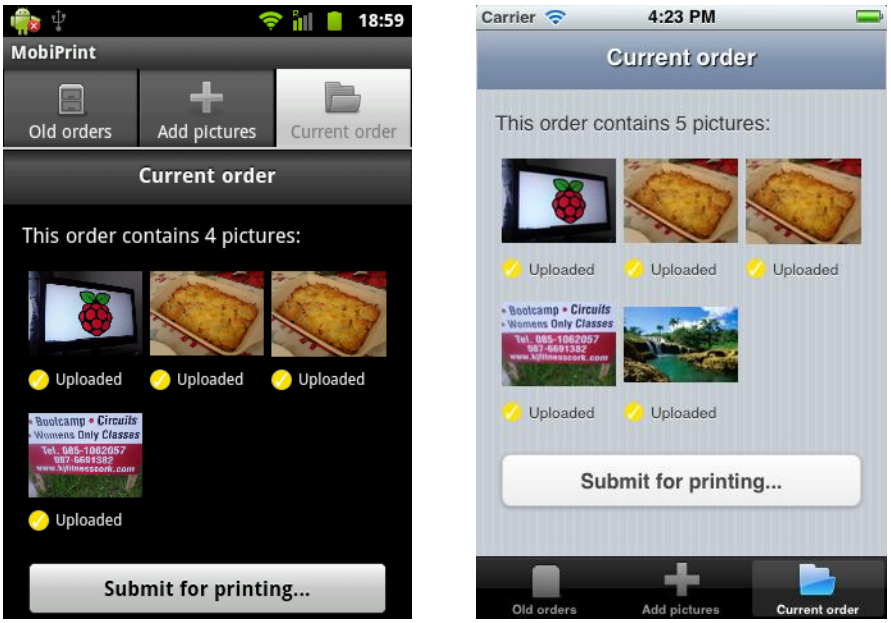


Figure 24: Screenshot of Rhodes-based implementation of MobiPrint on Android/iPhone

[28] Data encoding typically used to transfer binary data or form field values over HTTP, see glossary

6.2.3 Category scores

Functionality

Typically required device functionality is available in Rhodes, such as camera, location tracking and extended features like barcode scanning and on-screen signature capturing [29]. However, some features are not supported, like tracking of the heading (compass), accelerometer, application lifecycle events (pause/resume) and presses of hardware buttons. Since lifecycle events are not captured, implementing a service that still runs when the application is suspended is not possible without a native extension or changing the framework.

Extending an application with native code is possible either through "native extensions" or by adding existing libraries for the Ruby language (they may include native code). Native extensions can be created with a command line tool that comes with Rhodes and generates the necessary source code files and build scripts for each supported platform. This process is well-documented and enables one extension to be built for multiple platforms – the source code will still be platform-dependent, of course, but the exposed Ruby interface is the same.

As mentioned above, the included ORM *Rhom* has several shortcomings and is not considered a helpful addition to the framework. Issues with the built-in HTTP client were also explained above but can probably be solved with small bug fixes in the framework implementation.
Regarding other helpful APIs, Rhodes supports various features such as NFC, profiling of different places that are potential bottlenecks (time needed for rendering templates, executing controller methods, loading initial HTML page, etc.), reading from an attached card reader and others.

Rhodes receives only 2 points for the functionality score because of various unsupported but important features (e.g. application lifecycle events, hardware buttons), the issues with the included ORM and unfulfilled requirements in the sample application.

[29] After *Rhomobile* was acquired by *Motorola Solutions*, both features (also NFC support) were removed from the open source version and are now only available with a *RhoElements* license [Motorola2012]

Usability features

Regarding user interface functionality, the rating pertains mostly to the *jQuery Mobile* library. However, a modified and slightly outdated version of this library is included in the version of Rhodes that I used for the sample application, causing small problems that may no longer be present in the latest release of jQuery Mobile. For example, the page height is defined incorrectly and thus shows a white border, or list items may randomly stay selected when the list is scrolled. The library offers great features for mobile web sites: as mentioned before, it can asynchronously load local or remote web pages into the DOM tree and then transition to it with an effect. Many standard components such as buttons, lists with customizable content or form elements like fields and switches are supported. All of them are declared by HTML attributes such as `data-role="button"` and jQuery Mobile makes the necessary changes (CSS classes, DOM manipulation) as soon as a page is loaded. Also, theming is supported, allowing developers to customize colors, borders, sizes etc. A free online editor for themes is available [30]. Gestures such as swipes can be recognized simply by binding a callback function to a component.

The looks of the finished sample application are not really satisfying on Android, but quite good (similar to native look) on the iPhone. This can be vastly improved by replacing jQuery Mobile with the latest version.

In addition to HTML styling capabilities, Rhodes supports the native navigation bar of iOS and tab bar on Android and iOS (probably other platforms, not documented), both with an API that is very easy to use. In the HTML-based views, native styling could be applied with some effort by changing the respective jQuery Mobile theme. Rhodes already comes with themes for Android, iPhone and other platforms – of the first two, only the iPhone theme looks very good, while the Android theme seems to be incomplete and unmaintained. It is also possible to define different views for each platform by simply renaming the template file, e.g. `index.bb.erb` will be used instead of `index.erb` on the *BlackBerry* platform.

After all, Rhodes supports the basic UI components, allows implementation of extended features with gesture recognition, and can be customized with jQuery Mobile themes or

[30] http://jquerymobile.com/themeroller/, accessed: 2012-08-24

own CSS rules. The subjective impression, particularly on Android, could be better and is considered to be improvable with a newer version of jQuery Mobile. Some basic native components are supported and CSS themes that are close to native looks (especially on iPhone) are included for several platforms. Hence, Rhodes fulfills the basic requirements and most expectations in cross-platform support and per-platform customizations. The score in this category is 3 points.

Developer support

The framework and the IDE *RhoStudio* are offered through a single installer that contains all dependencies and is available for *Windows* and *Mac OS X*. Use of RhoStudio is not enforced since all requirements can also be installed separately and *Rake*, the de-facto standard build system for Ruby, is a command-line alternative for building, running and packaging a Rhodes application.

RhoStudio comes with a simulator called *RhoSimulator* that supports basic testing but misses simulation of device functionality like determining the current location. It includes the developer console of *WebKit* (as known from browsers like Google Chrome) that helps debug the web part of the application – the DOM tree, CSS rules, resource loading times, the JavaScript log console and other views are available for inspection. Backend code written in Ruby can be debugged in RhoStudio, but only using RhoSimulator. Breakpoints and inspection of variables are supported, but their values are not changeable. The simulator is preferred for repeated testing because it starts very quickly and shows the application log in a separate window (log entries from Ruby code). In contrast, builds for Android took quite long even for small changes because many files are recompiled.

Documentation for the framework is available online and gives a mostly complete overview of available functions of each module or feature. From changes within the year 2012, it seems that since *Motorola Solutions* acquired the original framework vendor *Rhomobile*, the quality and completeness became even better than before. Guides for the most important topics exist, such as building an application from the command line, creating the user interface, accessing device capabilities, data persistence with the ORM, utilizing the simulator, etc. Another important point is that the creation of native extensions is very well-documented, with examples for all supported platforms. However, example programs (apart

from an application that showcases most functionality [31]) and project templates were not available at the time of writing.

The learning effort for Rhodes is high from employing both Ruby as backend language and web techniques for the frontend. Together with the fact that debugging Ruby code is only really possible on RhoSimulator, and the lack of good examples, 3 points are assigned in this category.

Reliability & Performance

The Ruby library is compiled with the respective compiler for each platform. On Android, for instance, this results in a truly native library for the ARM architecture. Nevertheless, application startup and page loading times are very slow: the non-functional requirements of the sample application (p. 62) define a maximum time of one second for screen switches. On the Android test device, one of the slowest Android 2.3 devices, loading often takes several seconds. Even on the faster *iPhone 3GS*, page display is noticeably delayed at times, for example the background image may be loaded more than a second after other content in the "current order" screen. This also causes transition effects between pages to render less smoothly. An obvious reason for the slow performance could not be found – most probably the overhead of the split backend/frontend architecture or the built-in minimal HTTP server implementation are reasons for the observed performance problems.

The installed Android build of the sample application only took 320 KB of space. During development and testing, no crashes of any kind were encountered on both Android and iPhone. Yet, the poor performance results in a score of only 2 points.

Deployment, Supportability, Costs

Rhodes supports the platforms *Android*, *iOS*, *Windows Mobile*, *Windows CE*, *Windows Phone* and *BlackBerry* and thus covers the most important platforms (by popularity).

The open source part of the framework is licensed under the *MIT* license, while some extensions require a *RhoElements* license from Motorola. No information about professional support or pricing could be gathered because Motorola Solutions did not respond to

[31] https://github.com/rhomobile/rhodes-system-api-samples, accessed: 2012-04-26

enquiries. From the commit history in the source code repository of Rhodes, the framework seems to be actively developed with 20-30 commits per week in average. Open source contributions, however, often took months to be noticed and accepted.
As a free support option, a Google group is available for questions and discussions; 2 of 3 questions that I posted during development of MobiPrint remained without a matching solution. Traffic in this group is high and most questions receive several responses (as of August 2012).

Automated builds of Rhodes applications are possible in two ways. The Ruby build tool *Rake* can be used for most needed operations, including building, running and packaging the application. There also exists an online service called *RhoHub* that has been developed and supported by Rhomobile since 2009. It is available for free, including for commercial purposes. Only applications containing highly sensitive data (like social security numbers) are not allowed. The web site claims that professional support for RhoHub is available with paid plans.
RhoHub offers a Git repository for each application project that is protected by public key authentication and not publically accessible. If required settings are filled in (e.g. developer certificate and application provisioning profile for iOS builds), the latest revision of the application can be built for selected platforms. Afterwards, the resulting package files can be downloaded. Android, iOS, BlackBerry and Windows Mobile are supported as of August 2012.

Regarding internationalization, the framework comes with a simple wrapper for the Ruby library *localization_simplified* [32]. Setup is as easy as creating a separate file for each language that contains simple string translations, then it can be used directly in HTML templates. An example is included with the project template. Unfortunately, no extended features like pluralization are supported. This functionality could be replaced with a more sophisticated library for Ruby (using Ruby functions is possible within the HTML templates).

Rhodes applications have been accepted in app stores for years [Rhomobile2010] and still are, considering that no contrary messages were posted in the Google group for Rhodes.

[32] http://rubyforge.org/projects/l10n-simplified/, accessed: 2012-08-24

Development is active but open source contributions are rare and accepted very late. The freely available build service RhoHub is surely one of the most outstanding offers coming with this framework. Overall, Rhodes deserves 4 points in this category.

6.2.4 Conclusion

Rhodes tries to provide a minimal web framework which in my experience can complicate development more than necessary because an additional step for encoding and parsing data is involved and direct function calls must be replaced by HTTP requests. Advantages like multithreading – which would not be possible only with JavaScript (designed as running only with a single thread at once) – balance the disadvantages. Web developers may find the architecture more intuitive since Rhodes resembles the web framework *Ruby on Rails*. However, the resulting application performance is very poor even in simple views (as can be found in the sample application). Stability, the little effort for porting an application to another platform and the free build service *RhoHub* are main benefits of this framework. Altogether, the final result calculated from the weighted category scores is *2.9*.

Rhodes	*Weight*	*Score*
Functionality	4	2
Usability features	3	3
Developer support	2	3
Reliability & Performance	3	2
Deployment, Supportability, Costs	5	4
	Rounded final score:	2.9

Table 3: Evaluation results for Rhodes framework

6.3 PhoneGap and Sencha Touch

Just like the *Rhodes* framework is a combination of Ruby libraries for device and OS-level functionality and *jQuery Mobile* for the user interface, the choice of *PhoneGap* and *Sencha Touch* is a similar mix: PhoneGap supports device functionality and offers a wrapper for the platform's implementation of a web view that can display HTML pages. In case of the sample application, the purpose of Sencha Touch includes user interface creation, access to a web service and storage of retrieved data, all done with web technology (mostly JavaScript). The conclusion of this section will give separate hints on both parts of this framework combination.

6.3.1 Experience

All described experiences related to the sample application implementation refer to the versions PhoneGap 2.0.0 and Sencha Touch 2.0.1.1.

In contrast to the often used web application architecture with different HTML documents served for different actions/screens, Sencha Touch offers a different, single page approach: the major part of the application is written in JavaScript, but not embedded in HTML pages. Instead, a hierarchical system of view components and layouts declares the user interface, and Sencha Touch handles the creation of HTML elements in the DOM. I first had to become familiar with this very different approach because I already had previous experience with web development – for example, to update displayed data dynamically, one can access component objects (using the Sencha Touch API) instead of having to manipulate the DOM directly. Altogether, creating the user interface part of MobiPrint was thus fairly simple.

One big advantage was that testing could be conducted in a web browser except for cases that require device features via PhoneGap. With small distinctions in the code (e.g. *if* testing in web browser, use test data, *otherwise* scan real file system for pictures), most parts of the development could be completed without running the mobile application on the emulator or device. PhoneGap tries to implement its APIs according to web standards if possible, hence certain features work directly in modern browsers (e.g. geolocation).

Learning Sencha Touch was not easy at first as only a small number of guides and complete example applications was available at the time, presumably because Sencha Touch 2, the

major version I used, was only released in February 2012. For example, the orders screen should only show old orders, but information about both current and old orders are persisted in a "store" – in this case utilizing the simple key-value *local storage* standard (support built into Sencha Touch). With the documentation, I did not find out how to filter the list to only show old orders, and subclassing the store with a filter resulted in items being duplicated in the storage. I ended up writing a workaround, an additional "store" that only contains old orders, which is certainly not the best solution.

As mentioned in section 4.3.2 (p. 44) about the setup of PhoneGap with Sencha Touch as Android project, I integrated the Sencha Touch build tool with Eclipse: When the build begins, the web application part is recompiled and copied to the respective folder of the Android project from which the main HTML page is loaded by PhoneGap's wrapper for the web view component. Since the build tool for Sencha Touch does not return an error code if problems occur (e.g. JavaScript syntax errors), another custom script was necessary so that problems would show up in the Eclipse IDE.

In retrospect, accessing device functionality and file system with PhoneGap was very easy due to the simple API design and documentation with complete examples. With Sencha Touch, a few issues appeared but overall, the impression during development was good – especially user interface creation is straightforward once the basic concepts are understood. The strict separation with the MVC approach helped in keeping the overview and resulted in a nicely structured application.

6.3.2 Sample application implementation details

Sencha Touch fosters a strictly separated MVC application architecture. The three data structures of the sample application (orders, nearby stores, local folders containing pictures) are all defined as model and either connected to HTML5 *local storage* or temporarily persisted in memory by a simple "proxy" definition that determines how a store loads and saves models.

The JavaScript file of PhoneGap is included via the dependency loading mechanism of Sencha Touch: the file `app.json` lists all external dependencies that will be loaded when the application entry point is called, i.e. with the display of the main HTML page.

Since neither framework supports internationalization, such as translations, in any way, an own minimal JavaScript translation module was introduced for this requirement. Sencha Touch loads JavaScript classes as soon as they are needed. This means the three screens of MobiPrint, which are directly accessible through the tabs, are loaded even before the first entry point (the `launch` function) that a developer can use. Views are implemented declaratively with JavaScript, that is, the whole structure of a screen including e.g. layout, style and text labels are contained in one JavaScript file. Thus, as soon as such a file is loaded, the translation function can already be called. But the language for translations is only set in mentioned application entry point, so any previous calls to the translation function must work even though no language has been defined yet. Any external JavaScript library could have been used for internationalization purposes if this aspect is considered.

The following screenshot shows the application running on the Android 4.0 emulator and the iPhone simulator:

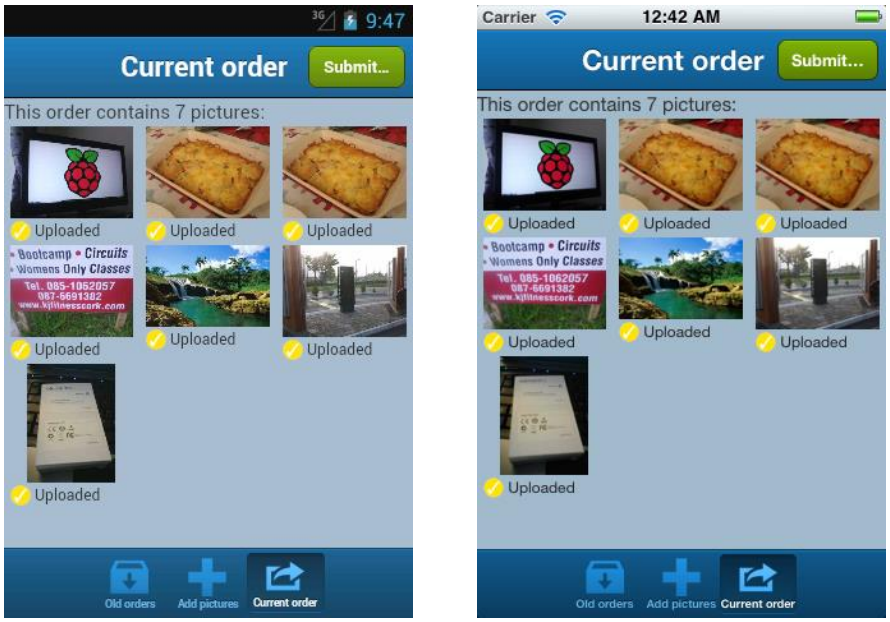


Figure 25: Screenshot of PhoneGap / Sencha Touch based implementation of MobiPrint on Android/iPhone

6.3.3 Category scores

Functionality

Since Sencha Touch version 2, the framework offers access to a small number of device features, like the camera. It automatically selects an available implementation which can be either PhoneGap or the interface offered only if the native packaging tool of Sencha Touch was used to package the whole application for running on a device. As only very few device features are supported through this interface of Sencha Touch, the decision was made in favor of PhoneGap's packaging approach, that is, creating a normal Android project and including the PhoneGap JAR file, for example (similar for iOS development: an Xcode project is used). This means that all PhoneGap APIs are available.

They include almost all expected device functionality (e.g. accelerometer, camera, network connection state) and other features like access to contacts, different types of notifications (e.g. vibration, message box, sound), file system access and storage. The latter two, and also others like location tracking, are implemented according to web standards – if the web component built into the operating system does not support one of these features, PhoneGap offers its own implementation, allowing the same interface to be used on all supported platforms.

Possibly long-running functions are designed to be run asynchronously, so they take two callback functions as arguments which are called on success or failures, respectively. This can be problematic in certain cases. For example, the sample application must scan the file system for folders containing picture files. With the asynchronous *File API* (a web standard), it is complicated to find out when and if all directories were scanned recursively because the function `DirectoryReader.readEntries` returns the file list of a directory asynchronously, i.e. at a later, unpredictable time.

More native functionality can be added very easily to an application through PhoneGap plugins. For instance, plugins for reading NFC tags, iOS push notifications or barcode scanning are available for certain platforms in the official plugin repository [33]. Own plugins can be implemented as well, following a simple platform-specific interface. This interface has been altered several times [34] which means the API is not fully stabilized yet. As an example

[33] https://github.com/phonegap/phonegap-plugins, accessed: 2012-08-27
[34] Observed in PhoneGap versions up to 2.1.0

for this interface, on iOS, methods of the plugin class (written in Objective-C) with the right signature can be called "directly" through the `cordova.exec` function (positional and named arguments are possible). Only the way of calling these native functions is different between the platforms, but hidden behind the platform-specific implementation of the `cordova.exec` function.

Additionally, PhoneGap supports many events, like hardware button presses, pause/resumption of the application or network status changes. An application can register callback functions to receive such events. The framework has the same issue as Rhodes in that only the HTTP *POST* verb is available for file uploads in "multipart/form-data" encoding.

Sencha Touch offers mostly UI functionality which is explained in the next category. Its MVC structure gives a good separation of concerns in the application source code. Even data storage implementations are included, such as the web standard *local storage*. It has to be noted that storing data can lead to small issues. For example, since local storage is a key-value mapping, IDs of a certain collection of model instances are saved under one key (e.g. "orders=3,5") and the instances are stored in JSON form under separate keys (e.g. "orders-5={id: 5, submissionDate: …}"). During development, it happened that the list of IDs had duplicate entries, such as "1,2,4,2" which could be problematic – this should not be possible in any case. Altogether though, both frameworks allow easy development and together offer expected features. Therefore, a score of 3 points is assigned for functionality because basic expectations are completely fulfilled and additional functionality is available through plugins, however many of them are only supported by a single, external developer and the plugin interface was not stable yet at the time of writing.

Usability features

In this category, mostly Sencha Touch features are rated since PhoneGap does not offer user interface functionality. However, it supports calling native code through plugins, allowing the use of native components. By default, the web view component displaying the HTML page takes up the whole screen, so the layout may need to be changed in case native components are added (e.g. tab bar). Several plugins exist that add support and a JavaScript interface for such components, like the navigation bar and tab bar on iOS – an application can then create and change these components from JavaScript code within the HTML page.

Considering Sencha Touch, the framework offers a hierarchical model, i.e. a parent component can contain one or more children components. Typical components like buttons, form fields or lists can be arranged using different layouts such as tabs, a carousel (switches to different view on swipe to left or right) or just a plain rectangle with different options (only use necessary height, take all space; layout children horizontally/vertically). A view is defined completely in JavaScript code. The layout is usually defined inside the view class by a hierarchical structure of JavaScript objects describing the type and options of a component, e.g. `{xtype: "button", text: "OK"}` would declare a simple button labeled "OK" which is converted to corresponding HTML elements when the view is displayed. The problem with the implementation of the hierarchical approach is that many nested levels of HTML document elements are created and using a large number of CSS classes. The latter allows individual designs of every type of component, but changes to the whole look may take long because many CSS rules have to be modified. Sencha Touch does not come with a native-like look on any platform, so if platform-specific UI looks are desired, the effort of changing the CSS rules could be quite high. Themes for Android, BlackBerry and iOS are included with the framework, but they mostly change color values and thus are not close to the native look.
The content of components can be any HTML code, leaving the possibility for own design and formatting. The framework also supports several transition effects when switching between views and supports detecting gestures such as swipes or taps.

In summary, Sencha Touch provides the basic functionality expected from a framework mainly focused at (mobile) user interfaces, including detection of gestures, transition effects and a sophisticated layouting system. The subjective impression is very good and consistent throughout the available components. However, applying native or platform-specific looks may require significant effort. Therefore, not much more than the basic requirements are fulfilled, leading to a score of 3 points for usability features.

Developer support

PhoneGap provides a full reference documentation online, with examples for all APIs. Guides for several topics, including project setup (platform-dependent), upgrading from old versions or developing plugins.
Sencha Touch also provides free online documentation with a complete API reference and screencasts about several topics and important concepts (components, layouts, MVC

approach, etc.) – at the time when I implemented MobiPrint, only few guides existed for version 2.x but the number has been growing since, now including explanations for most core concepts and helpful, complete example applications.

As PhoneGap packaging requires platform-specific projects, tool restrictions for those platforms apply, e.g. Xcode is necessary to create an iOS application based on PhoneGap. The native code, however, usually does not need to be altered (unless own native code is added) and so any editor can be used for the web part of the application, in this case the Sencha Touch part.

Regarding debugging, no extra tools are offered with either framework. For native code (PhoneGap project or native plugins), tools for the respective platform may be available. For the web part (Sencha Touch), testing can be done in a browser. On an emulator or device, special tools are available. *weinre* [35] is part of *Apache Cordova* (the open source part of PhoneGap) and can be integrated into the main HTML file, providing a client that connects to a debugging server which is also included. A developer can then view several debugging controls – the *WebKit* debugging tools (DOM/CSS inspection and manipulation, loading times, loaded resources, web databases, etc.) – by accessing this server from a browser. External tools like *iWebInspector* for the iOS simulator are similar in functionality. The browser-based emulator *Ripple* [36] supports displaying a web application in different screen resolutions and to forge certain values like the geolocation. It can also emulate PhoneGap functionality up to the version 2.0 (as of September 2012).
With testing in a browser, developers should know that HTTP requests to external hosts are only restricted by a whitelist of accessible domains (enforced by PhoneGap), not by cross-origin restrictions [37] – but in a modern web browser, *CORS* rules are applied, meaning that any web services used by the application should respond to HTTP *OPTIONS* requests if browser testing is desired.

[35] https://people.apache.org/~pmuellr/weinre/, accessed 2012-08-26
[36] http://ripple.tinyhippos.com, accessed 2012-08-13 and 2012-09-25
[37] See *CORS* in glossary

For Sencha Touch, the product *Sencha Architect* is sold as separate software and offers an IDE supporting visual UI creation and preview, code editing and packaging for mobile platforms (Android and iOS). Prices start at $399 for a one-developer license.

Both frameworks are well documented and helpful examples are available. Debugging is possible with a browser, for which many existing inspection tools are freely available, although testing and debugging on a device requires some effort. The learning effort is acceptable, and only one programming language (JavaScript) is really necessary for each framework unless own native plugins are developed for PhoneGap. The result in this category is a score of 4 points.

Reliability & Performance

On the Android test device, one of the slowest Android 2.3 phones, loading time and screen switches are very slow, taking up to multiple seconds. With the iPhone 3GS, performance was fair with smaller but noticeable delays. Application startup is also slower than with a native application (up to several seconds), most probably because many parts of the Sencha Touch JavaScript library are loaded. The problem of highly nested view hierarchies (see usability features category above) could be the reason for the poor performance. In the sample application, text labels in the list of old orders are at level 30 (`<html>` tag counting as level 1) and in most levels, several CSS classes are applied. A plain PhoneGap application with an empty HTML page (Sencha Touch not included) starts very fast, so aforementioned performance issues are mainly related to Sencha Touch.

No crashes of any kind were encountered from both frameworks, except for an Android-specific problem that lets the application crash in certain cases when PhoneGap's JavaScript bridge for making native calls is used. This seems to apply only to the Android emulator 2.3.3 and very few devices of the same or older OS versions [OHA2010a], and could be a problem if older devices are targeted with an application.

The sample application built with PhoneGap and Sencha Touch 2 had the smallest size of all five implementations with an installed size of only 188 KB on Android. Nevertheless, performance issues and the problem with potential crashes on Android 2.3 result in a score of only 2 points for the framework.

Deployment, Supportability, Costs

The two frameworks must be distinguished in this category. PhoneGap is free, open source under the *Apache License 2.0* and supports the platforms *Android*, *BlackBerry*, *iOS*, *Symbian*, *WebOS*, *Windows Phone* and *Bada*. *Adobe Systems* purchased the original developer company *Nitobi* in 2011 and since contributed the source code to the *Apache Software Foundation*, with the Apache project now named *Cordova*. Developers from multiple companies actively contribute to the project, including *Adobe*, *IBM*, *Microsoft*, *RIM* and *ICS* [Apache2012]. Professional support is offered by Adobe in form of different support options and prices. All plans include online trainings, a knowledge base, chat office hours and a forum. The plans differ in the number of included support cases and advisory hours, availability and response time. More expensive plans support a higher number of developers, add e-mail support, bug fixes, support with PhoneGap plugins and even remote debugging. The price range goes from a 1-developer plan ($24.95/month or $249.99/year) to a "Corporate" plan for 20 developers ($1995/month or $19999/year) and higher with the "Enterprise" plan (unlimited number of developers, price only on request) [Adobe2012b].

The PhoneGap framework itself does not support automated builds because different project types are required depending on the platform. Thus the usual tools for these projects can be used. In addition, an online service called *PhoneGap Build* is available. Similar to *RhoHub* for the Rhodes framework (see page 87), it can build an application for several platforms (all named platforms except *Bada*) if the required settings are made. The source code of the application, i.e. the `index.html` and other files, can be pulled from a public Git repository or uploaded manually. Platform-specific project settings or additions cannot be altered. The service is still in development (in beta testing at the time of evaluation [38]) and Adobe plans more features (such as integration of PhoneGap plugins) and addition of paid plans [Adobe2012a].

As free and public support alternatives, a community web site for questions and ideas and a separate, quite active Google group exist for questions and discussions, and the framework has gained enough attention so that developers can find good help on general discussion sites like *StackOverflow*. Also, "community release notes" were introduced in May 2012,

[38] Out of beta testing in September 2012 (see http://phonegap.com/blog/2012/09/24/phonegap-build-is-launched/, accessed: 2012-09-25), changes not tested or documented here

allowing everyone to submit and accumulate PhoneGap-related events, articles and other resources in a public Git repository.

Sencha Touch is developed by the company *Sencha Inc.* and developers can opt between several licenses: *GPL* version 3, two free commercial licenses (one of them for embedded devices, not relevant here) and a fourth, paid license if Sencha Touch is incorporated in the creation of own SDKs or developer tools. No public code repository exists, so no information about development activity is given. The public online forum, which includes a sub-forum for bug reports, shows active involvement of Sencha employees in triaging and resolving bugs and other problems that users incur. Professional support is sold in packs instead of time-based contracts. For instance, the 1-developer pack for $299 includes premium forum support (private forum), maximum initial response time of three days and 40 “credits” (equaling 4 support tickets). The most expensive offer is the 20-developer pack for $4995 that additionally includes telephone support, emergency bug fixes and a reduced initial response time of two days – 750 credits are included, equaling 75 support tickets or 15 hours of telephone support.

The framework only supports *WebKit*-based browsers and thus only mobile platforms whose web view component uses WebKit. *Android*, *iOS* and *BlackBerry* are supported officially. Internationalization support is unfortunately missing from the provided functionality.

Both PhoneGap and Sencha Touch are actively developed and mobile applications built with either framework are accepted in app stores if the general guidelines are obeyed. PhoneGap seems to have gathered a large community which has already contributed different native plugins, for example. Various options for professional support are available in different price ranges. Considering PhoneGap only, many mobile platforms are supported. Porting the sample application to iOS only required the creation of an Xcode project and inclusion of the iOS-specific JavaScript file of PhoneGap – the rest of the application was left unchanged. This leaves a very good impression of supportability for different platforms and therefore 5 points would be well-deserved in this category. The missing built-in internationalization support in both frameworks and the fact that Sencha Touch only works with WebKit-based browsers leads to a deduction, and thus a final score of 4 points.

6.3.4 Conclusion

PhoneGap is freely available and has a complete documentation with full examples, helpful guides and a large community that can help with problems and actively contributes to development. Except for the asynchronous approach of the file API, no issues arose during implementation of MobiPrint – all features and integration of native functionality via plugins are very easy. Since the framework is only designed for the purpose of accessing device data/functionality and displaying a web view component as primary screen, it can be combined with any web framework (not only Sencha Touch) or just custom HTML pages, and thus gives much freedom to developers.

Sencha Touch uses a single HTML page approach for the application architecture, consisting of mainly JavaScript code, in contrast to the HTML template files in the Rhodes framework. With the hierarchical component system and several layouting and sizing options, the user interface of a web application can be built easily through code. Also, a paid IDE is offered (*Sencha Architect*) which should simplify this task and help with Sencha Touch development even more. Unfortunately, the framework has had performance issues with the sample application, especially on slow devices. Furthermore, it only supports WebKit-based browsers, making it difficult to turn a mobile application into an application served as web site on the Internet because popular browsers like Firefox or Internet Explorer do not render Sencha Touch applications as intended.

The final score for this combination of frameworks is *3.2*, but would have been higher if a more performant web framework for mobile applications was combined with PhoneGap. A list of such frameworks (mostly UI-only, like *jQuery Mobile* or *Sencha Touch*) can be found in appendix B.

PhoneGap and Sencha Touch	*Weight*	*Score*
Functionality	4	3
Usability features	3	3
Developer support	2	4
Reliability & Performance	3	2
Deployment, Supportability, Costs	5	4
	Rounded final score:	3.2

Table 4: Evaluation results for framework combination of PhoneGap with Sencha Touch

6.4 Android

6.4.1 Experience

From a university course, I have gained some previous experience programming Android applications, but have never created a more complex application with multiple screens. Creating and understanding project creation and concepts from Android version 2.x again was therefore quite easy for me. In addition, I was already knowledgeable about the Java programming language from several projects. Experiences about the sample application development refer to Android SDK version 10 (for the Android operating system 2.3 and newer).

A lot of functionality, including a large subset of the Java standard library, is included in the SDK, and the design of additional APIs is mostly very good, with small exceptions. For example, JSON parsing and serialization is supported. Since a value in the JSON format can have different types (e.g. string, number, special `null` value), one method exists to retrieve a value for each type. Strangely, `null` is not handled like Java's `null` value and instead, the `getString` method of a JSON object would return a string `"null"` in case of a `null` value. Thus, one would have to use the separate `isNull` method whenever such a value can occur.

The most problematic part of the Android SDK is the emulator. Its advantage is that all previous Android versions can be emulated. However, several emulator-specific issues exist that were reported but still not solved, especially with older versions. For example, the Android 2.3 emulator does not handle orientation changes correctly [OHA2010b]. Another example was mentioned in the reliability category of the PhoneGap section (p. 96).

Working with the native SDK was mostly very straightforward. For example, built-in support for background tasks helped to separate concerns, e.g. the orders screen from retrieving information about orders from the web service. The documentation with many examples, and the fact that solutions to many common problems can be found on the Internet, helped very much with developing MobiPrint for Android.

6.4.2 Sample application implementation details

I implemented each screen in a separate so-called “activity” class that handles updates and interaction with the components. Data storage was solved by writing JSON files to the application’s cache directory for storing information about orders and nearby stores. Android also offers simple key-value storage (`SharedPreferences`) which could have been used here, but typically data that can be retrieved again, for instance from a web service, should be stored in the cache directory that can be deleted any time to free up more space. A high-level wrapper for the built-in SQLite database support would have been helpful (such as an ORM), but is not included with the Android SDK.

For downloading thumbnails, I tried to use the HTTP cache available since Android 3.2 since the sample application requirements include handling HTTP cache headers (like the expiry date). Unfortunately, the implementation does not expose the filename of cached content and so the thumbnails could only be displayed if the cached content was saved by myself to a separate file, resulting in thumbnails taking up twice the space – once from the HTTP cache and another time from being saved by the application in order to be accessed by filename. Downloaded thumbnails are stored directly in the cache directory and the HTTP header for the expiry date is handled manually (obviously not the best solution).

Just like with the *Titanium* framework, I experienced memory problems with large pictures. These issues are documented below in the reliability category.

For picture selection, the image gallery API could have been used, but since it did not show all folders on the SD card that contain picture files, I decided to use the same approach as with the three previous solutions: scanning the SD card recursively for such folders.

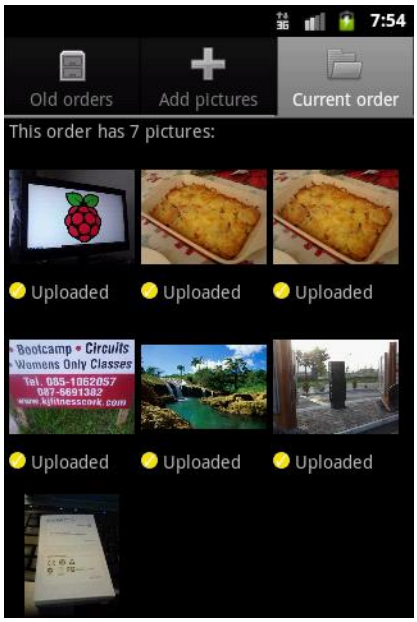


Figure 26: Screenshot of native Android implementation of MobiPrint

6.4.3 Category scores

Functionality

Android devices may support many types of built-in sensors and actuators [Google2010a]. For all supported hardware types, an API to access it – and to find out whether it is available – exists. A per-application permission model asks users to confirm the use of certain functionality (e.g. use of the current location). If a permission is not requested for an application, it cannot use the respective API. In this aspect, the design is slightly inconsistent: in most cases, accessing an API without permission triggers a Java exception. If the Internet access permission is missing, however, an `UnknownHostException` is thrown which is misleading. As mentioned, Android includes a subset of the Java standard library and thus offers much more functionality than required. Furthermore, the *Google Cloud Messaging* API is offered free of charge and allows a server to send short messages to an application via Google's servers, such as notifications, even if the application is not running at the moment.

Multithreading is natively supported and fostered. With the latest Android release, blocking operations (like network access) are not allowed in the special user interface thread but have to be run in the background. The aforementioned built-in class for asynchronous tasks makes this very easy by running one method in the background and then passing the result to a method which is run in the UI thread and may modify UI components if necessary.

As platform-specific SDK, the Android SDK supports all features that a device with the Android operating system could offer. Additional functionality can be added through other Java projects (e.g. packaged as JAR file) if no unsupported classes are used, or by special "Android library projects", a project type made available in the Android plugin for Eclipse and capable of including Android-specific files such as XML layout or translation files. Therefore, missing functionality like a database abstraction library for SQLite could be added easily. Ultimately, Android gains 4 points in this category for the well-designed APIs with many features and good extensibility.

Usability features

Android offers many user interface components. Typically, they are defined through XML-based layout files, but can also be generated through code. Many attributes, like colors or fonts can be customized. Different layouting variants are available, using a hierarchical approach that positions children components within their parent (e.g. linear horizontal or vertical layout). Gestures can be detected by a transparent overlay view or manual interception of touch events. Animations, such as transitions between screens, are also a built-in feature. Apart from normal UI components, display of HTML inside a web view component is implemented and usable as well, based on *WebKit*.

The Android UI is easy to use and its looks have been improved over many versions of Android. Fragmentation, i.e. differences between the versions, can be problematic to program for both new and old versions. For example, the "action bar" component, a bar at the top of the screen that allows navigation between screens or action buttons for the current screen, was introduced in Android 3.0, replacing the options menu of previous versions that was triggered by the menu button. Android provides a support library that offers features not available in older versions. The action bar, however, cannot be integrated with this library – only the old approach of showing an options menu instead. An external compatibility implementation [39] is needed to use the new component with old Android versions.

[39] Third party libraries exist for this purpose, e.g. *ActionBarSherlock*

The subjective impression of Android's user interface is great, especially with the more recent versions. Applications that are not customized fit together with the rest of the operating system. In order to apply the same criteria as with the cross-platform frameworks, it must be stated here that the effort of applying native looks is evaluated as well – as native SDK, Android of course has zero effort and supports all natively available components. Even if version fragmentation can be problematic, the official compatibility library supports older Android versions with the UI approach available in those versions. Together with the variety of built-in components and the simple layouting, Android deserves 5 points in this category.

Developer support

The SDK download comes with a simple installer that includes the necessary tools for downloading APIs, emulators, documentation and other resources for selected Android versions. It runs on *Windows*, *Linux* and *Mac OS X*. If so desired, a plugin for *Eclipse* (*ADT, Android Development Tools*) can be installed separately. Apart from project templates and creation, application debugging, packaging and other features, it includes a visual UI editor which is not very easy to use – however manually writing screen layouts in XML might not be a problem for an experienced developer. The debugger allows breakpoints, value inspection and changes during runtime and shows the log output from the emulator or device.

A new Android SDK version release always ships with the respective Android emulator. Since Android has had many releases over the last years, many APIs were added and de-facto standard UI patterns changed. Therefore, testing with old Android versions on the emulator is very helpful for checking backward compatibility of an application. Changes in the Java source code can be deployed very quickly, allowing for a quick change-and-test cycle. The emulator's downsides are its slow performance and some issues of old versions as mentioned before (p. 100).

In addition to the emulator, the DDMS (Dalvik Debug Monitor Server) program is included with the SDK tools and allows to access emulators and devices: Logs and the file system can be viewed, files can be retrieved and added, and screenshots can be made. Simulation of slow/offline mobile network or a physical location and other settings are possible.

Documentation is available online on a dedicated web site for developers and includes API reference, guides, blog articles and various tutorials covering many aspects of an application. APIs that were added in later Android versions are clearly marked as such. Important classes come with examples in most cases.
Several Google groups and an IRC channel exist for discussions and problems with development. Bug reports can be issued online as well. Apart from that, it can be said that most Android problems can be solved quickly through existing solutions on the Internet, due to the very large developer community.

As a conclusion, the Android project fulfills most expectations in this category. With Java as programming language and XML (e.g. for layouts and translations), the required learning effort is acceptable. The score for developer support is 4 points.

Reliability & Performance

Regarding UI performance and reaction time, Android is very fast and thus superior to the cross-platform frameworks in this aspect. Multithreading support decouples blocking operations from the user interface. The sample application runs very stable – the only issue with reliability during development was manual memory management of pictures, similar as with the *Titanium* framework. Pictures had to be downscaled instead of loading them into memory in their full resolution. Android offers methods to load a downscaled picture without loading the full picture into application memory. In case many pictures are displayed on a screen, a more complicated approach may be required because of per-application memory limits (for example, keeping only currently visible pictures in memory).

The installed MobiPrint application takes up 576 KB [40] of space which is acceptable. With the previously evaluated combination of *PhoneGap* and *Sencha Touch*, the installed size (188 KB) was much smaller because the Android implementation is written in Java and the generated bytecode takes up more space than JavaScript source code.

Altogether, the native implementation proved to be faster than any of the above cross-platform frameworks and the sample application is running very stable. Manual memory

[40] Without splash screen (image size is 46 KB for test device's resolution) because that functionality is not a built-in feature of Android and was not required for the sample application

management, however, can render the implementation more complex, depending on the type and requirements of the application. Therefore, only 4 points are assigned instead of the maximum score.

Deployment, Supportability, Costs

In contrast to the cross-platform frameworks, the Android SDK only supports Android-based devices. The SDK and tools are freely available and released under open source licenses. *Google* and partners of the *Open Handset Alliance* are actively developing the operating system, SDK and new devices (including phones and tablets). The developer community is very large, and even customized variants of Android and ports to other architectures exist (Android for x86).

Automatic builds of Android projects are often done with one of the Java build tools *Ant* or *Maven* which make the setup quite simple. Builds with Ant are described in the official documentation.

Sophisticated internationalization support is a built-in feature of the Android SDK. Translations are stored in separate XML files for each locale and support important features like ordered arguments [41] and pluralization. In the user interface, translated texts can be referenced in the XML-based layout file or through code. Furthermore, all resource files, such as UI layout files or pictures, can be varied based on locale, screen resolution and other variables.

Concerning support options, no official professional developer support seems to be available from *Google* or the *Open Handset Alliance*. Instead, questions can be asked in online support forums (Google groups) which are highly frequented with new questions and do not always provide many answers. Chat channels and so-called “office hours” are also available, the latter mostly targeting discussion of a different topic every week. Since the developer community is very large, problems can also be asked at many different web sites and solutions for typical issues can be found with a simple Internet search.

[41] If a text contains more than one variable part, assigning a name or number to the variables is important because sentence order may be different between languages.

The Android project is very actively developed and new versions and updates are released regularly. All expectations of features in this category, like automatic build support and internationalization, were exceedingly fulfilled. But since the Android SDK is platform-specific and direct professional support does not seem to be available, only 4 points are assigned here.

6.4.4 Conclusion

Android's SDK and tools are freely available and a development environment can be installed very quickly and easily. The official plugin for Eclipse supports developers in debugging and deploying applications. Regarding functionality, all device features that can be available in Android-based devices are accessible and many helpful APIs are included, although with the exception of a high-level abstraction for database access. Since most Java libraries can be used, this is not actually a problem.

Learning how to develop Android applications is fairly easy, since only one programming language and simple XML files are used, and the documentation offers many tutorials. Unfortunately, direct professional support seems not available. Many online forums and web sites allow questions about development in many languages and solutions to common problems can be found in many places.

With the great native speed and multithreading support, Android is well capable of running complex or graphics-intensive applications and applications even work well on slow devices such as my test device. Only very few problematic points were found in the evaluated categories, leaving Android with a result of *4.2* points as the final, weighted score.

Android SDK	*Weight*	*Score*
Functionality	4	4
Usability features	3	5
Developer support	2	4
Reliability & Performance	3	4
Deployment, Supportability, Costs	5	4
	Rounded final score:	4.2

Table 5: Evaluation results for Android SDK

6.5 iOS SDK

6.5.1 Experience

Since the sample application was only developed and tested on the iPhone, only experiences relevant for iPhone development are documented here. For instance, issues of porting an iPhone application to iPad devices are not part of this evaluation. Version 5.1 of the iOS SDK was evaluated.

In order to develop the sample application for iPhone, I had to learn the programming language *Objective-C* and usage of *Mac OS X* and the IDE *Xcode* because I did not have any prior knowledge. With the official documentation, especially Apple's introduction document for Objective-C [42], I could start with development quickly. Xcode turned out to be fairly simple and offered the right tools, for example visual editing of screens and their connections ("storyboards"), or the quick deployment to the simulator.

As for the UI, I dislike the approach of laying out components with absolute coordinates instead of using automatic layouting mechanisms. Absolute positioning is however not really an issue because resizing options are available that change a component's margin automatically on screen size changes (e.g. orientation changes). With dynamic creation of UI components however (through code), setting absolute coordinates is more complicated than having an automatic layout component (e.g. vertical layouting) and adding the new component as a child. This is more of an application-specific and subjective issue because I decided to implement two screens in a way that components (for showing pictures) are added at runtime – in most cases this may not be necessary and the next major version of the SDK (for iOS 6) includes automatic layouting features, anyway [Apple2012e].

The API of the iOS SDK and the underlying *Objective-C Foundation Framework* is mostly easy to use and functions and parameters are clearly and consistently named and thus I often did not have to consider the documentation or other sources to solve small problems.

[42] https://developer.apple.com/library/mac/documentation/cocoa/conceptual/objectivec/objc.pdf, accessed: 2012-07-08

Apart from the requirement of an Apple computer for development and thus to become familiar with their operating system Mac OS X and the development environment Xcode, development for iOS was mostly straightforward and easy to learn. Various web sites and the official documentation helped a lot in solving incurred problems. The only major issue during MobiPrint's implementation for iPhone was *Core Data*, a framework handling the definition, persistence and loading of object-based models that supports different data storage backends (e.g. SQLite).

6.5.2 Sample application implementation details

The typical architecture of simple iOS applications was applied for MobiPrint: all screens were designed using Xcode's visual editor for user interface screens and transitions between them, stored in a single "storyboard" file. The SDK includes a view controller class `UINavigationController` that handles navigation between screens and automatically adds the back button in the "navigation bar" at the top of the screen which is necessary because iPhone devices do not have a hardware back button.

For functionality not found in the iOS SDK, third party libraries were included, such as a JSON library to parse web service responses and *ASIHTTPRequest* [43], a library for enhanced HTTP support like "multipart/form-data"-encoded requests for picture uploads.

Data storage is implemented with the abovementioned Core Data framework because it seemed easy to integrate with a list view, for example, and template code was automatically inserted by Xcode after I selected Core Data in the project setup.

As stated at the beginning of this chapter, the approach to scan the file system for folders with pictures is not applicable for iOS because applications cannot access files outside of the application's directory. Instead, the iOS API for displaying a picture selection dialog and reading the selected file was used.

[43] http://allseeing-i.com/ASIHTTPRequest/, accessed: 2012-07-23

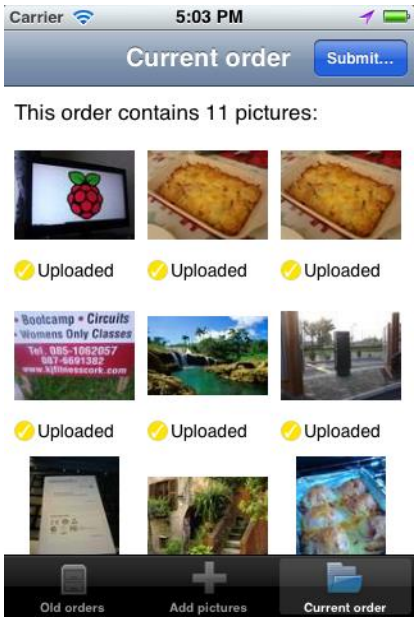


Figure 27: Screenshot of native iPhone implementation of MobiPrint

6.5.3 Category scores

Functionality

iOS-based devices, like the iPhone product range, can include several sensors such as accelerometer or gyroscope, camera and GPS, plus actuators like sound output. All of these hardware features are supported by APIs in different parts of the SDK. These parts are called "frameworks" and are mostly optional so that only required functions have to be linked with the compiled program. With the *Core Motion* framework, for example, an application can be notified about gyroscope value updates, i.e. device rotation changes.

For data storage, the *Core Data* framework is offered which represents a persistence framework for relational data and thus allows modifying entities in an object-oriented manner. As data backend, either a binary format or a SQLite database can be used on iOS. Models, with their attributes and relationships to other models, can be edited in Xcode and later referenced by the model's name in the source code.

During implementation, a few shortcomings of Core Data were discovered, partly coming from the fact that it is not designed as ORM but for object graph management. First of all, no primary key handling exists. Therefore, if the use case is to cache data from a web service locally, one should implement an update mechanism – items that already exist locally must be overwritten and new items must be added. In the sample application, this was the case with orders which have a unique ID. With Core Data, one first has to query existing items,

and then decide whether to change an existing object or create a new one. This could be achieved more easily with the SQL command `INSERT OR REPLACE`. Developers used with SQL databases should consider that data might need to be stored in a different way because of named differences. Using external libraries like ORMs is another option if Core Data does not fit the requirements [44]. If the differences are not an issue, Core Data is a good addition to the SDK and can be integrated quickly, for example it is used to fill the list of orders in the MobiPrint application.

Another helpful API is *iCloud* storage which can save model instances from Core Data, document files and key-value mappings (e.g. application settings) and synchronizes them automatically with Apple's servers and other Apple devices of the same user. Depending on the application type, this feature may be very interesting to share settings or user files across devices.
It can also be used to share data between different applications if they are developed by the same team/developer. By design, iOS applications cannot access files or data of another application except through official APIs (e.g. picture selection in the sample application). Therefore, more complicated ways have to be used, such as iCloud (only applications with the same team identifier), listening to a custom URL scheme (contrived example: `youtube://<VIDEO_ID>`, can only exchange small information) or through an own web service.

Regarding extensibility, any iOS application can be linked with compatible external libraries. As mentioned before, a library for extended HTTP features was added to the sample application without problems.

The iOS SDK offers a clean API with all necessary features. A high-level wrapper for the Core Data framework or for relational databases can easily be added by third party libraries if Core Data alone does not fit the requirements. Also, sharing data between applications is not as easy as it could be and can be an issue for certain types of applications. Overall

[44] For example, the *RestKit* framework includes support for Core Data with an additional ORM layer, amongst other features (http://restkit.org/, accessed: 2012-10-11)

however, almost all expectations were fulfilled and even additional auxiliary APIs are provided. Hence, 4 points are assigned in this category.

Usability features

iOS offers various components, including simple gesture recognition by attaching a certain recognizer to a view – this can even be done in the visual UI editor. As mentioned in the experience section above, layouting is done with absolute coordinates and can be done in a visual editor or in code. In the latter case, it makes layouting more complicated as compared to layouting variants known from Android, for example, which automatically lay out child components accordingly. Most components are customizable, for instance their text or background color, font, and tab background color. Several predefined component instances are available for use, such as a set of common tabs, i.e. a picture and the translated text (e.g. “Downloads”, “Search” or “Favorites”). This is very helpful because a developer does not have to add own pictures but can reuse the picture and text that users are already familiar with. Display of HTML pages is directly supported with the “web view” component.

As of my opinion, the style and design of components is great. User interfaces can be designed in a simplistic way without losing features. For example, a typical pattern is to add a “delete” button on top of a list view that, when clicked, shows individual delete symbols next to each row. This saves space and thus offers more freedom in UI design. Applications often have a quite uniform (sometimes customized) look that integrates well with the rest of the operating system. For example, the sample application uses only built-in components and the picture selection dialog available in the API in order to display captured and downloaded pictures. Since the look of the shown dialog is the same as the application – except for the color – it seems that it is a part of the application itself.

Again, the easiness of applying native looks is a criterion here in order to have a fair comparison with cross-platform frameworks. Of course the effort of native styling is zero. Together with the built-in number of good looking components with simple API and highly customizable attributes, the iOS SDK earns the full score of 5 points in this category.

Developer support

As the first point and major drawback of iOS development, it must be stated that an Apple computer with the Mac OS X operating system is necessary to download Xcode and the iOS SDK in order to develop applications for iOS-based devices. Installing Mac OS X on computers that are not Apple-branded is unlawful since the software license forbids installation on such computers [Apple2012c]. Furthermore, in order to deploy an application to a real device (such as the iPhone), even if only for testing purposes, a developer requires a membership in the *iOS Developer Program* which costs 80€ or $99 per year [45].

Xcode is a well-designed IDE supporting most needs in iOS application development. It is not perfect though: For instance, text strings can be translated with the help of special files but Xcode does not offer extraction of translatable strings from the source code. This and other translation-related features are available through external, sometimes paid tools [Mata2012]. Objective-C development is supported by other IDEs but for a completely integrated development experience, there does not seem to be a good alternative to using Xcode.
Debugging support is as expected with breakpoints, value inspection and also changes to values using special commands of the command line debugger which can even evaluate Objective-C expressions as if they were executed by the program itself.

Documentation is freely available online and offers a reference to available APIs, guides, articles, sample code and more. Reference pages seem to be very complete, describing all public methods and properties of classes and the most important concepts. They also contain links to related guides and example projects, if available. Developers that are members of the paid iOS Developer Program also have access to videos and related slides of many basic and advanced topics, for example from Apple's annual developer conference. As with Android, the size of the developer community is very large and thus solutions to many common or beginner problems can be found on various web sites on the Internet.

[45] Prices as of May 2012 for Germany (80€) and USA ($99), respectively

The learning effort to get started with the iOS SDK is low since only Objective-C is used as programming language and other file formats can be edited visually with Xcode. Also, Apple provides a very concise introduction to Objective-C that is also suitable for beginners [46].

Testing with old iOS versions and simulators is only possible to a certain extent. With the upgrade to Mac OS X 10.8, the latest version of Xcode at the time of writing (Xcode 4.4) does not allow to download the iOS 4.3 simulator anymore which was possible with the previous version of operating system and Xcode (Mac OS X 10.7.4 and Xcode 4.3). However, applications can still be compiled for this iOS version and tested/debugged on devices that have it installed.

Even with the restriction to Apple computers and Xcode as the only good IDE, iOS support for developers surely fulfills more than the basic expectations: its documentation is complete and offers examples and guides for most topics. Therefore, 4 points are assigned for developer support. Note that the need to pay for the developer program in order to test on real devices is considered in the costs category below. It does not influence the score of developer support because the same limitation applies to cross-platform frameworks when targeting iOS-based devices.

Reliability & Performance

The sample application was tested on an iPhone 3GS device and throughout development, no application crashes occurred except, of course, for uncaught exceptions caused by programming mistakes. Regarding performance, the slowest part of the other four implementations of MobiPrint, loading and displaying pictures from a folder, cannot be compared here because the iOS-based implementation cannot scan folders and thus shows a picture selection dialog using the SDK. Absolutely speaking, the sample application runs very fast on the test device and subjectively, no noticeable delays were perceived. The only exception is the first time the picture dialog is shown (delay smaller than 2 seconds) and the delay incurred when a selected picture is loaded for display on the “add pictures” screen – which is not really significant because it is implemented synchronously but could be done without noticeably blocking user interactions. Multithreading is supported for iOS

[46] https://developer.apple.com/library/mac/documentation/cocoa/conceptual/objectivec/objc.pdf, accessed: 2012-07-08

applications and multiple ways exist that simplify running tasks in the background, enabling developers to improve the reaction time of the user interface.

In conclusion, reliability and performance seem to be outstanding when considering only the sample application. The installed size of the application on the test device (616 KB) is acceptable. A score of 5 points is assigned because of the high stability and UI performance with little to no delays.

Deployment, Supportability, Costs

As a native SDK, only iOS-based Apple devices are supported, that is, *iPhone*, *iPod touch* and *iPad* devices. The source code of the iOS core frameworks and operating system are not released as open source, but Apple actively develops both as can be seen from regular SDK and device releases and new features and improvements introduced with each of them. With the high popularity of named products, developers do not have to fear that further development of iOS and its SDK will be abandoned.

Automated builds of projects are possible using the command line tool *xcodebuild*.

Basic support for internationalization is available. For example, text strings can be made translatable in source code using the predefined macro `NSLocalizedString`. User interfaces can be translated as well, but assigning a language to a UI designer file and then changing it will only apply the changes to the file with the selected language, leaving versions of other language of the UI untouched [47]. As already denoted in the developer support category, there are ways and tools to keep the UI files synchronized for all languages [Mata2012], but this functionality is not included in Xcode. Using different folders for each locale, one can specify other language or locale-specific files, such as images or videos.

Apple offers professional support through developer forums, bug tracking and a ticket system for requesting individual support. All of these options are only available for members

[47] Xcode 4.5 and the iOS 6 SDK provide a better way: designer files only need to exist once and can be translated with a separate strings file for each language (https://developer.apple.com/library/ios/releasenotes/DeveloperTools/RN-Xcode/_index.html, accessed: 2012-10-12)

of the iOS Developer Program – the standard program costs 80€ or $99 per year [48] while the enterprise program (enables in-house distribution of applications) is priced at $299 per year. The price is not seat-based, i.e. a company can register for the program with a single account and have multiple developers using it. Two so-called "Technical Support Incidents" are included with a one-year membership. A technical support request counts as such an incident and has a typical response time of 3 business days. Requests are handled via e-mail communication with Apple's support engineers (English only). Additional incidents can be purchased in quantities of 2 or 5 at the cost of 80€/$99 and 202€/$249 [Apple]. No other support options could be found.

The iOS SDK is only developed by Apple and developers need to pay an appropriate fee for a one-year membership to get support or influence development via the bug tracker web site. The price for support tickets is acceptable. Regarding internationalization, no extended features are supported, unfortunately. Therefore, basic expectations are fulfilled in this category and 3 points are thus assigned for the score.

6.5.4 Conclusion

Development with the iOS SDK is restricted to Apple computers with Mac OS X. However, the need to use Xcode is balanced by its quality, stability and ease of use. Apple provides complete documentation with many example projects, and offers a helpful introduction document about the most important concepts of Objective-C. Therefore, starting with iOS development is fairly easy.

All necessary features for accessing device and other functionality are available through well-designed APIs. Extensibility is given by adding third party libraries, for example to add HTTP features missing from the core APIs. Additional features like iCloud storage can be very helpful, depending on the application.

[48] Prices mentioned in this paragraph are from May 2012 in Germany and USA, respectively. No € price known for the enterprise program.

Great performance with little delays was perceived in the sample application except for screens that are not optimized to take advantage of iOS's multithreading capabilities. Apart from the major drawback of being limited to develop using Apple computers and software, only few points of critique were found, resulting in a final score of *4.1* points.

iOS SDK	*Weight*	*Score*
Functionality	4	4
Usability features	3	5
Developer support	2	4
Reliability & Performance	3	5
Deployment, Supportability, Costs	5	3
	Rounded final score:	4.1

Table 6: Evaluation results for iOS SDK

7 Conclusions and recommendations

In this final chapter, I first want to present a summary of the evaluation results (section 7.1) and then explain the main differences between the five tested solutions regarding their scores (section 7.2). Afterwards, I draw conclusions from the scores and my own findings, discuss whether cross-platform solutions can already keep up with native development and give a recommendation concerning the decision of using a cross-platform approach or not. Since I find this approach already usable, I also give recommendations and a set of important points to consider when selecting a framework among several alternatives. As outlook for the future development of such frameworks, potential points of improvement are outlined in a separate section. Next, I elaborate on mistakes and missing considerations in my own methodology as used for the comparison and evaluation. In section 7.3, I state my opinion on web technologies as the future technology, and outline the main benefits and issues of developing applications with HTML5 and related technologies. Finally, I summarize my work (section 7.4) and write about future work expected both in similar research and development of cross-platform frameworks and solutions (section 7.5).

7.1 Summary of evaluation scores

The following table shows all results of the previous chapter. In each row, the framework(s) with the highest score in that category is marked bold. As described before, the final score is calculated by multiplying a framework's scores with the respective weight, and dividing the result by 17 (sum of weights) in order to get a score range between 1 and 5.

Category	*Weight*	*Titanium*	*Rhodes*	*PhoneGap and Sencha Touch*	*Android SDK*	*iOS SDK*
Functionality	4	3	2	3	**4**	**4**
Usability features	3	4	3	3	**5**	**5**
Developer support	2	**4**	3	**4**	**4**	**4**
Reliability & Performance	3	3	2	2	4	**5**
Deployment, Supportability, Costs	5	**4**	**4**	**4**	**4**	3
Rounded final score:		**3.6**	**2.9**	**3.2**	**4.2**	**4.1**

Table 7: Overview of evaluation scores for all frameworks

I have to mention that the given weights only match a single point of view as explained before (see section 3.2.3 on p. 31). Different weights may yield other results.

7.2 Conclusions

As a foreword, developers knowledgeable about Android, iOS or both will have noticed that Android's SDK slightly won over the iOS SDK in my evaluation. The reason is that, in the considered criteria, Apple's SDK had a few shortcomings: deploying to a device requires a paid membership in the *iOS Developer Program*, making development non-free. The vendor lock-in from the developer support category also applies for automated builds using the *xcodebuild* tool. And as a last distinguishing point, there is only simple support for translations available as opposed to the Android SDK, which offers more sophisticated features like pluralization. My work did not focus on the difference between these two SDKs, but rather on advantages or disadvantages of the cross-platform over the native development approach. The fact that both native SDKs have higher scores than the three cross-platform solutions gives an indication that native development could still be superior. In the following, I want to describe important differences and give my personal opinion about which points are important to consider when deciding between the approaches.

7.2.1 Differences in category scores

In this section, I want to clarify the main differences between the five presented application frameworks regarding each category. A short conclusion is drawn for each category, but my overall conclusions and recommendations are expressed in subsequent sections.

Functionality

Regarding functionality, all cross-platform frameworks support basic functionality like application lifecycle events (except Rhodes), access to built-in hardware, intercepting button presses, or network, file and data access. In order to support all functions that the native SDKs offer, a framework would need many high-level abstractions or has to make compromises – Titanium supports some extra features of both Android and iOS without exposing them as platform-independent interface. The number of developers actively involved in the cross-platform frameworks is supposedly small in comparison to the native SDKs that are supported by large companies. Therefore, functionality of the frameworks is first of all restricted to what their developers can implement and support.

In my opinion, PhoneGap takes a good approach here, in that only portable functionality is supported in the core API, whereas other features are available in separate plugins that are written in native code and thus can utilize all functions of the respective platform. These plugins are not an official part of the framework and often developed and published by volunteers.

Aside from that, the native SDKs have features that cannot be employed easily in a cross-platform framework without changing the framework itself. One example is multithreading support: Titanium and PhoneGap incorporate JavaScript as main programming language for business logic and hence also for background operations. Typical blocking operations like network access are implemented asynchronously, but the language itself still cannot run two tasks in parallel and running small operations many times in the only thread can cause performance problems. Facebook confirms this experience in their statement about why they do not use web technology anymore in their iOS application [Facebook2012].

Altogether, native SDKs are slightly superior by providing extra functionality that is not – or not yet – available in the cross-platform frameworks, or can only be added by developing plugins. However, the basic functionality offered by the three cross-platform solutions may be enough, depending on the application.

Usability features

The score difference in this category (both native SDKs gained 5 points) is mostly due to the support for native components or native-like styling capabilities. Also, the native SDKs provide high customizability and well-designed UI components. Regarding quality and availability of features for usability, the cross-platform frameworks are rated good throughout. The modified version of jQuery Mobile included with Rhodes was the main reason for the lower score in this category, caused by issues like incorrect UI styling and the mediocre subjective UI impression. However, this is not a general rating for jQuery Mobile, which has been improved since, and behaves differently in the official, unmodified version. The way how layouting works in the five solutions varies greatly, but with all of them, a customizable frontend can be created. Titanium additionally requires testing on both supported platforms because missing layouting attributes and different default behaviors can lead to problems. The other two cross-platform solutions make porting an application very easy by using (mostly) web technology. One must say that the subjective impression

varies and the look of UI components differs greatly. This can be an important point when deciding between the solutions. Altogether, all five solutions offer the necessary capabilities in this category.

Developer support

For the criteria of developer support, the five solutions are not much of a difference. Except for PhoneGap, which is used with native project files and thus can be used with the respective IDEs, all vendors provide an IDE for development, debugging and deployment of applications. Rhodes is the only framework with a score of only 3 points because the learning effort for two programming languages (Ruby and JavaScript) is considered higher and Ruby code debugging is only supported on RhoSimulator, not on devices or other simulators (e.g. the Android emulator). Still, typical requirements, such as debugging with breakpoints, are fulfilled by both the cross-platform frameworks and the native SDKs, rendering them pretty much equivalent for this category. One could argue about the documentation quality of Titanium and Sencha Touch, in which some small errata exist and some detailed information is missing, but overall, I consider the documentation to be mostly complete and accurate, and hence appropriate for serious development.

Reliability & Performance

During implementation and testing of the sample application, no relevant crashes were found with any of the cross-platform frameworks and native SDKs, with one exception: In the Titanium, Android and iOS implementations, memory management of displayed thumbnails is done manually. The former two implementations showed problems when too many pictures were loaded into memory at once because of the limited amount of memory available for applications. This example is very application-specific but in general, memory management may have to be seriously considered – if done manually, it can add to the complexity of the source code.

Performance-wise, a cross-platform framework can vary greatly from its native competitor. Both in observed reaction times and subjective regard, Titanium seems to be the fastest of the three cross-platform solutions, which, together with the native UI components, brings it very close to a native application framework. The choice for JavaScript as programming language, however, has certain implications: multithreading and asynchronous operations are not supported directly, apart from those APIs which are already implemented to run in

the background (e.g. HTTP requests). PhoneGap as framework for use of web technology has the same problem, while Rhodes and Sencha Touch – both rated with 2 points for bad performance – have other potential causes for their poor performance as mentioned in the evaluation.

As can be seen from the differences I just described, the architecture (e.g. automatic memory management by web view, or manual) and programming language (e.g. JavaScript restrictions) are two major factors of reliability and performance. The reliability issues I experienced were not related to cross-platform vs. the native approach, but rather the fact that with both Titanium and Android, memory management (e.g. thumbnails) had to be done manually in the sample application, to a certain extent. For performance matters, it is clear that the native approach is slightly to strongly superior, depending on the cross-platform framework it is compared to. Nevertheless, Titanium shows that almost native performance can be reached and PhoneGap's performance is dependent on the web framework it is combined with, and thus even an application based on web technology and the PhoneGap application wrapper can be fast enough for many types of applications.

Deployment, Supportability, Costs

Each of the vendors of the tested cross-platform frameworks offers professional support with different options or payment plans. Google and the Open Handset Alliance do not seem to offer such plans for Android, and Apple only seems to offer e-mail support through a ticket system. Free support options such as online forums exist for all solutions, and a point that speaks for Android and iOS is that their developer community is so large that solutions to typical problems can almost always be found on the Internet and not only in the official forums. Therefore, even though support options differ, the five solutions can be considered equivalent if free and paid support are weighed the same. Great free support is offered for the presented native SDKs, but professional support by their vendors is not as good as offered for the cross-platform frameworks. Deciding based on support is probably very subjective. One should consider whether professional support, or a special option (e.g. 24/7 phone support), are necessary for the company, in the respective industry, or for the support of a certain product.

All five solutions can be considered very good in this category. One point was deducted from the iOS SDK score because of its only basic translation support and the need to pay the membership fee to gain access to technical support and the bug tracker. Regarding internationalization capabilities, all frameworks are extensible so that external, more sophisticated solutions can be used for internationalization and other purposes.

7.2.2 Recommendations: Cross-platform vs. native development

In this section, I outline the main points speaking for the cross-platform approach and for purely native development, respectively. Deciding between these approaches is the first major step in choosing the right tools and strategy for mobile application development.

Cross-platform development

A major, potential advantage of the cross-platform frameworks is code reuse, one of the main purposes of a framework (see definition on p. 13). The “Write once, deploy everywhere” promise, which people often associate with such frameworks (but is not actually marketed exactly like this), is not far from reality. Both HTML-based solutions, Rhodes and PhoneGap with Sencha Touch, did not require any changes to the code itself. Titanium, on the other hand, required additional changes when I ported the Android variant to work on iOS. However, these changes were mostly about missing parameters for UI layouting, customizations of the user interface to look acceptable on iOS and only minor changes caused by different behavior on the platforms. These issues could have been mitigated by also testing on the iOS platform from the beginning. Even though porting of the cross-platform applications did not require much effort, one has to consider that customizations to the user interface may be necessary or desired in a real application.

The small effort for porting a finished application that I experienced with the implementation of MobiPrint with all three cross-platform solutions can be mentioned as another great advantage which is especially important if a company plans to extend their platform support at a later time. Developers should be aware of restrictions of each supported platform that may also affect cross-platform frameworks. For instance, I first implemented the sample application to scan for pictures on the file system which is not possible on iOS and thus was not a portable feature (see p. 67).

Regarding functionality like hardware and file system access, application lifecycle events or network usage, the cross-platform solutions are not far behind their native competitors concerning availability of these features. In case a functionality is not supported (e.g. the emerging support for reading NFC tags), many cross-platform frameworks provide support for adding native code to an application. The three presented solutions solve this by a plugin architecture.

Smaller development and maintenance effort due to a high amount of reusable code is surely a potential advantage of cross-platform frameworks, but as well debatable. While maintenance effort could not be evaluated in the limited timeframe of this thesis, I tried to collect indications as to whether development effort between cross-platform and native development is notably different. The measured implementation time of the sample application, however, is greatly influenced by my previous knowledge of the Android SDK. Comparing the iOS SDK and the three cross-platform solutions, which I both had to learn about from scratch, the amount of time spent for the native iOS application (19h 28m) is about 15% lower (or more) than the development time needed with Titanium, Rhodes and PhoneGap with Sencha Touch (between 23h and 24h 15m, respectively). A statistically relevant comparison would consider the development time of multiple, experienced developers and I think this will make a large difference – note that I needed less than 15 hours with the Android SDK about which I was knowledgeable before. If I were more experienced with iOS, the native Android and iOS implementations might have taken around 30 hours together; and with more experience in any one of the cross-platform frameworks, the development time might decrease to 21 hours or even less. Therefore, in my opinion, cross-platform frameworks can decrease development time – and one should definitely consider cross-platform development if an application should be implemented for 3 or more platforms because the more platforms are targeted, the more advantage the frameworks can bring in this context. Of course this personal statement is not statistically proven and may only be valid if experienced developers for the cross-platform solutions are available. Another indicator for high maintainability is the size of the code base [vanVliet2008], e.g. the number of lines of code. This measure was not tested in the evaluation.

Developer support can be considered to be of the same quality, and availability, as compared to the native SDKs. Documentation of the tested cross-platform frameworks was mostly complete and provided not only an API reference but also additional learning sources like guides or screencasts. Only small mistakes were found in the reference documentations which I mainly used during implementation of MobiPrint. As the frameworks are under very active development, the quality can be expected to improve over time.

HTML-based framework types have specific advantages. First of all, experience of web developers/designers can be "reused" for mobile applications. In case of PhoneGap, for example, close to no extra knowledge is necessary for an experienced web developer: project setup is easy and afterwards, an application can be developed with web technology only (unless PhoneGap plugins or other native functionality are needed). Second, an application can – in theory – be deployed as web site on the Internet if it can run without device features that are not available when run in a browser. Depending on the framework architecture, this can be a more or less easy process. Rhodes-based applications, for example, are structured in a client-server fashion and the controllers could be reused on the server side of a normal web site. However, Rhodes is customized to run a local web server on mobile devices and does not directly support deployment as independent web site. With some modifications, it may be possible to reuse the code since Rhodes is similar to, and inspired by the *Ruby on Rails* framework for web applications and thus the application source code might be usable with this web framework. Considering PhoneGap, deploying a mobile application as web site is very simple: only calls to device-specific functions must be surrounded by a conditional statement that decides whether the application is running as mobile application or only in a web browser. Section 7.3 further discusses the usefulness of HTML5 in mobile applications.

Native development

Native SDKs do not foster code reuse because APIs and programming languages are mostly incompatible, and cross-compilation solutions like *XMLVM* (p. 149) are not ready for production use and depend on a compatibility layer. On the other hand, a major benefit of native development is the availability of experienced developers, especially for popular platforms like Android and iOS. Development on these platforms is even taught in university courses, while the vendors of cross-platform frameworks still try to gain more popularity –

for example, *Appcelerator* hosts a yearly conference for developers and enterprises, and also offers developer certifications in order to create a larger community of skilled Titanium developers.

Comparing features, the native solutions are leading in usability feature support and in particular superior in terms of the subjective UI impression ("look and feel") and performance. With the typical platform look and native UI components, native applications integrate well with the operating system and other applications. If a cross-platform framework is chosen, the native look may be hard to achieve or not all customization capabilities can be utilized. Of course it is mostly a design decision whether the native or a custom look should be applied to an application, but there are cases in which non-native applications were rejected from app stores because they are not similar enough to typical native application behavior/design and thus native components or behavior can be necessary [49].

Only Titanium came close to native performance, most probably because it displays native components instead of rendering primarily HTML content. Only one of three tested cross-platform solutions offers a visual designer for user interfaces directly from the vendor (*Sencha Architect* [50], separate product), while tools for few of the other frameworks are available from third parties (e.g. *Codiqa* for jQuery Mobile, *ForgedUI* for Titanium [51]).

The native approach seems much faster regarding application startup time and time needed for screen changes and other user-controlled actions. This can play a major role in user experience and should be considered when deciding for or against native development – graphics or effect intensive applications and applications with heavy data processing on the mobile device should rather be implemented natively. If only modern, high-end devices are targeted, performance issues become less prevalent. Multithreading support is a factor that influences such issues. The trend towards JavaScript in cross-platform frameworks – all presented frameworks except Rhodes primarily use it – leads to problems if background tasks can run very long as JavaScript by itself does not allow concurrent execution.

[49] For example, fullscreen HTML-based applications with a look that strongly differs from the native one, and without using any native components (e.g. tab bar) seem to be problematic regarding acceptance for app stores (especially Apple's store). Detailed scientific evaluation of these cases was not part of my work.
[50] http://www.sencha.com/products/architect, accessed: 2012-04-20
[51] See http://www.codiqa.com/ (accessed: 2012-07-01) and http://www.forgedui.com/ (accessed: 2012-08-22)

Recommendation

In general, I can *recommend* using a cross-platform solution because they are not far behind native development anymore or even on the same level in some of the evaluated categories. Comparing with *Paananen*'s results [52], that described the cross-platform frameworks to be "still in their infancy" and to have "undocumented features" and "constant changes to the framework", my experience has been a better one (roughly a year after his comparison): Projects like PhoneGap have become rather stable through active work of the main contributors together with bug reports and code contributions from volunteers. Documentation is good throughout tested cross-platform frameworks and the quality of developer tools is acceptable but could be more integrated, in particular in the frameworks based on web technology. For instance, debugging the HTML part of a PhoneGap application on a simulator requires additional tools like the debugging server *weinre* that is not integrated with IDEs. Yet some functionality and performance issues, like missing multithreading support, still exist. Since applications created with the presented cross-platform solutions are accepted on app stores, there is essentially no reason to opt against them. Through the often supported plugin mechanism, additional native features can be used if needed. Regarding costs, there is also not much of a difference and professional support options are often available. But still some considerations regarding the type of application and long-term strategy, for example, have to be made before deciding for one framework. They are discussed in the next section.

The main problem of cross-platform frameworks is that they can probably not achieve the same level of user experience as native applications. Less effort is required to get high usability out of a native user interface since existing components can be reused, are highly customizable and users are familiar with them. With frameworks that do not employ native components or at least similar styling, the same quality and user experience may be harder to implement. Therefore, the decision should also focus on the user interface aspect and related requirements of the product that shall be developed. Titanium represents a type of framework that is an exception to this pitfall as it allows inclusion of native components. Other frameworks with this feature exist (see appendix B).

[52] See [Paananen2011], chapters 5 and 6

7.2.3 Recommendations: Selection of a cross-platform solution

In order to decide for a specific cross-platform solution, one should first ensure that a cross-platform approach is really suitable for the application(s) that is planned to be created or ported. Abovementioned advantages of the different approaches should be considered, in particular technical issues that are harder to solve. For example, if the application is graphics intensive, depends on a fast user interface or requires complex calculations in the background, choosing native development for its performance may be appropriate. This section shall give recommendations about which points should mainly be considered when the decision in favor of the cross-platform approach has already been reached. Named considerations can be applied to any cross-platform framework, not only the ones presented.

The first decision should be about the primary technology. Can HTML and other web technology display the desired design without implementation problems? If a native look is preferred over a uniform or customized look for each supported platform, then a framework like Titanium, which uses native components, is more appropriate. Facebook has proved with their iOS application that both HTML5 and native rendering can be very similar with some effort, but may present users with performance issues if not done correctly. In the Facebook example [Facebook2012], the application was rewritten mainly in native code instead of using a hybrid application with HTML for the user interface. HTML is still incorporated in certain cases, but for performance reasons, the application mainly relies on native components, rendering and code. Measurements show significant improvements in startup time and other performance criteria [Applico2012]. End users also seemed to be much more pleased with the natively implemented application, as a significant raise of the rating in Apple's app store showed [Constine2012]. Mark Zuckerberg stated in an interview of September 2012 that it was a mistake to rely too much on HTML5 for mobile applications because the expected quality was not reached with this approach. However, the users of Facebook's mobile web site outnumber the users of installed Android and iOS applications, and therefore HTML5 is an important part in the company's strategy [Zuckerberg2012]. As a conclusion to this example, using HTML as primary technology may not fulfill performance criteria in certain cases but may still be relevant if additionally to mobile applications, a

mobile web site is of strategic importance. I mentioned before that HTML-based frameworks may allow to reuse code for web sites, as well, if device features are not required.
Learning from these mistakes, the decision for or against HTML should be answered by going through the anticipated UI design and weighing the implementation effort with HTML/CSS against the effort with native means. Any potential performance bottlenecks should be revealed, such as screens with lots of presented data or complex design. Long-running tasks are also problematic since JavaScript is single-threaded and the standard "web workers" [53] is far from being supported on all platforms. One should also consider that applications without any native user interface part are more likely to be rejected as mentioned above. Advantages and issues of HTML as technology will be discussed further in section 7.3 (p. 134).

Together with the basic technology decision, stability, active development and acceptance on app stores are major points. Many immature cross-platform projects exist at the time of writing and only few of them can be expected to be developed further in order to become ready for production. It seems that a number of major frameworks are currently emerging as widely accepted solutions, typically the ones with a viable vendor and/or a large open source community as main contributor. In 2011, *Gartner* suggested that the solutions present at the time cannot be assumed to "satisfy all needs or remain viable for a multiyear period" and that the decision should be based on short-term needs [Gartner2011a]. As mentioned above, I think some of the frameworks are stable enough for productive utilization and can already be considered for more than short-term strategies. Availability of knowledgeable programmers – whether about the framework or only the used programming language – is presumably not a problem since the languages and technologies (often JavaScript, HTML) are well-known by many developers and thus the only hurdle is to learn the framework's API. Regarding app store rules, all tested frameworks were accepted with mentioned exceptions in application-specific cases, e.g. if an application does not take advantage of native functionality or the user interface is too different from typical native applications.

The two decision steps above should already have reduced the list of considered frameworks to a small number. Frameworks that do not support targeted platforms can be eliminated.

[53] Extension for JavaScript allowing asynchronous background tasks to run in parallel [W3C2012a].

For example, Titanium only supports Android and iOS at the moment, and probably BlackBerry 10 in the future. If a business may support additional platforms in the future, this must be considered in the selection because framework vendors might be focused on certain platforms and not willing to add support for others.

If at this stage, more than one framework is remaining, additional points of comparison could be available support, likelihood of the vendor to add support for new features (e.g. NFC), extra tools (e.g. a visual UI design tool), etc. In case the previous decision was made against HTML, there may be additional decisions about the technology. For example, is a statically typed/compiled language like C++, Java or C# preferred over dynamic languages like JavaScript? *MoSync*, *mgwt* and the *Mono* ports for Android/iOS [54] are examples for the use of static languages, while all presented cross-platform frameworks employ dynamic languages (specifically, JavaScript and Ruby). Frameworks and tools of types that have not been tested in my work could also prove helpful. Titanium, Rhodes, PhoneGap and Sencha Touch are mobile application frameworks that target fully or mostly cross-platform code, i.e. application logic, user interface and other functionality is written in the same programming language (or fixed set of programming languages) and only needs minor customizations per platform. An application can however also be implemented in a way that the user interface is designed in a completely different manner on each platform. Cross-platform solutions like *J2ObjC* or *MonoTouch* and *Mono for Android* primarily support this model: code for application logic can be reused, while screens or platform-specific features are implemented separately using wrappers around native APIs (e.g. Android's layouting system on the Android platform and iOS-specific UI classes and methods on iOS). Since I did not evaluate the quality of those solutions, no recommendation can be given about this kind of architecture.

A final decision should be made depending on required features, regarding current and potential future requirements (NFC-based payment, for instance). Support for device and usability features, as evaluated in separate categories in my work, are very important, as is the extensibility of a framework (for example using external libraries or plugins).

[54] See appendix B

Specifically, from the evaluation results and experiences, I can recommend the frameworks *Titanium* and *PhoneGap* because they are easy to learn from a developer's point of view, well-documented, extensible and support expected device, UI and application features. PhoneGap can be paired with plain HTML and CSS, or together with one of the various web frameworks built with mobile applications in mind (see appendix B). Since I only evaluated the combination with Sencha Touch, it remains to test the performance and usefulness of other web frameworks (e.g. *jQuery Mobile*). If HTML is chosen as technology, PhoneGap and probably similar frameworks bring the great advantage that a mobile application could be deployed as web site on a normal web server with few changes, i.e. using device functionality only on mobile devices, not if the application is called from a web browser. Many types of applications, including those with special requirements (e.g. background service when application not in foreground, display of native UI components), can be realized with PhoneGap and available plugins.

If a fully or mostly native user interface is desired, or a very good performance is important, Titanium is the better solution. Testing on all targeted platforms should be done at an early stage with this framework in order to find out which platform-specific customizations have to be applied and whether UI behavior is different.

7.2.4 Points of potential improvement of cross-platform solutions

Two of the evaluated cross-platform frameworks can be recommended for production environments, namely Titanium and PhoneGap (in combination with a matching web framework for web content on mobile devices). But all tested frameworks have certain weaknesses compared to the native development. The frameworks are still under active development and their full potential is most probably not yet unlocked. In this section, I want to outline the major points that should be improved in order to make cross-platform frameworks and development easier, more useful and possibly more widely used.

The most important point in my opinion are performance issues: while the sample application implemented with Titanium was quite fast, the other implementations were slow in changing between screens, taking up to several seconds. The exact bottleneck has to be determined and eliminated. As mentioned with Sencha Touch, for example, a complex DOM tree of the HTML page could be the reason – for other frameworks like Rhodes, there may be other influences leading to poor performance.

Application startup time was also slower than with native applications. This can both have to do with the framework's application architecture itself and how much effort is needed at startup, but also with the implemented application – developers using HTML-based frameworks, for instance, should ensure that as little code as possible is loaded from the beginning, and many web site performance tips can be applied, such as minification of JavaScript code [55].

Regarding application performance, multithreading should also be considered for tighter integration with the frameworks' APIs, especially since some modern mobile devices already have dual core processors and also users of older models could benefit from separating the user interface from other workload. Purely JavaScript-based frameworks are problematic because concurrency is conceptually not supported by this programming language. Using future web standards, "web workers" [W3C2012a] are a way to offload the main thread in order to keep the user interface responsive, but it is not widely implemented yet – support for this standard has actually been removed in Android 2.2 and was not reintroduced until the current day [Google2010b].

As second point, I would like to see frameworks being more tighter integrated with native features in order to make cross-platform application look and behave more like native applications. This way, cross-platform frameworks will provide stronger incentives for developers to use them for more types of applications. For example, running a background service can be important for applications that need to regularly update data or notify the user in certain situations, e.g. if a mail or message arrived, or to draw the user's attention for other reasons. PhoneGap does not offer this feature as core functionality, but as plugin that is only available on Android yet. Titanium supports Android and iOS, however does not provide a common interface, so developers have to deal with differences between the platforms. In my opinion, a platform-independent API is appropriate in many such examples because functionality is quite similar. For instance, PhoneGap's core API is already designed to be mostly cross-platform, so native plugins could also be designed to share the same JavaScript interface across platforms, but with separate native implementations.

[55] Minification includes the removal of unnecessary characters (e.g. whitespace, newlines) and shortening of variable/function names as applicable, in order to save space and thus achieve faster parsing.

Regarding UI features, cross-platform frameworks can learn from their native alternatives and also from cross-platform frameworks for desktop applications, for instance from layouting features. *GTK+*, a framework for user interfaces supporting several desktop operating systems, offers hierarchical layouting with clear and intuitive options for sizing components. Apple introduced automatic layouting features with the iOS 6 SDK [Apple2012e] in order to allow adaptation to the different screen aspect ratio of the iPhone 5 (16:9 instead of 4:3). Programming for different screen sizes may be important if several types of devices, and thus many screen sizes and resolutions are targeted. Implementing responsive design [56] or separate screen layouts depending on the device type (e.g. tablet) should be simplified. Sencha Touch already provides this functionality by "device profiles", enabling developers to adapt the UI to the type of device while sharing common code. Other frameworks should simplify this in a similar way.

The above were the main open issues I see in the cross-platform solutions. With these points considered for implementation and improvement, I think the frameworks can become more popular and widely used, and get closer to the quality and feature variety of native SDKs.

7.2.5 Criticism of the methodology

In this section, I shortly want to describe on which points I focused in my thesis, which aspects are missing for timely reasons, and what I could have done better in my evaluation of cross-platform solutions versus native development.

First of all, my comparison and selection of criteria for the evaluation was mainly based on the viewpoint of developers and small to medium businesses that want to enter mobile application development or switch from native development to a cross-platform approach. Therefore, criteria that are interesting from other points of view, e.g. from large enterprises, may have been left out, such as integration with backend systems or cloud services, or more detailed financial considerations. A major factor is the fact that long-term experiences, including maintenance effort and costs, could not be taken into account in the given timeframe, but are very important, both for developers and from a financial perspective. A comparable measurement of development time and size of the resulting code base, for

[56] Adaptation of a single screen/page to work with different screen sizes, for example by automatically hiding or resizing components on small devices.

example by considering implementations from multiple developers, would have been interesting for evaluating maintainability.

Weights for the five categories of criteria could have been chosen differently, but I think the results are distinct enough so that different weights will not have much influence. However, additional criteria, as described above, can of course have an impact on the order. In my opinion, the experiences gained during the sample application implementation and conclusions drawn from the evaluation are much more valuable than the raw scores.

Only a set of cross-platform frameworks could be selected for comparison. I intentionally chose frameworks which seemed to be actively developed and ready for production use. New types and less popular solutions were thus left out, even if they could have been worthy competitors. For example, only cross-platform frameworks based on dynamic languages were evaluated, but with *MoSync* or *Mono* for Android/iOS, statically compiled variants exist and should also be taken into consideration when deciding for a platform-independent development solution.

7.3 Are HTML5 and other web technologies the future?

Web technology is found in many of the presented and considered frameworks (see figure on p. 8 and appendix B). Modern web technologies comprise HTML5 (additions to HTML markup), CSS3, JavaScript and additional extensions and APIs like drawing/rendering, data persistence (e.g. *local storage* or *Web SQL* databases), location or *web workers*, and other technologies such as the *WebSocket* protocol for bi-directional communication between a web application and a server.

After the rise of mobile applications and introduction of cross-platform solutions, the next logical step was to also support web sites and applications on mobile devices in order to gain more visitors and extend use cases (e.g. location-based services). Several working drafts exist that add APIs for specific mobile device features, e.g. for network events (online/offline), media capture (pictures, videos, etc.), hardware sensors or so-called “web intents”, a way to delegate certain actions to external applications [W3C2012b]. At the time of writing, the state of most device API standards is “public working draft”, meaning it will probably take several months or even more to finalize them. Therefore, support in mobile browsers and in web view components – as used by hybrid applications – cannot be

expected very soon, especially on older devices which often do not receive operating system updates anymore.

A possibly interesting future option of cross-platform frameworks is web deployability. Cross-platform solutions based on HTML as primary technology could as well support deploying applications on the Internet and thus allow to reach more users – certain functions specific to mobile applications would have to be disabled, for instance there is no web standard yet for accessing the native NFC interface. From the tested solutions, only PhoneGap allows web deployment because its API mostly consists of (upcoming) web standards and the HTML part is not restricted in any way. Rhodes, on the other hand, follows the design principles of the web framework *Ruby on Rails* but has a client-server architecture that puts device functionality in the server part and thus is different from a normal web server, making it harder to deploy a Rhodes application to the web. The web deployment model is a planned feature [Motorola2011]. Also non-HTML based frameworks can support generation of web applications. Titanium supports this but it seems the web page generation is not yet ready for production use.

Even if deployment of a mobile application as mobile web site is not considered, web technology can offer beneficial features. First and foremost, this is platform-independence, but only to a certain extent, i.e., not all features or standards are fully supported on all browsers and mobile platforms. The separation of HTML markup from CSS styling is very helpful for applying different design themes on each mobile platform. With regard to the sample application, the Android SDK and Titanium based implementations, which both include native UI components and features, caused problems with manual memory management of thumbnails if a large number of them should be displayed (see evaluation in chapter 6). One way to solve this is of course to do more sophisticated management of resources, e.g. releasing thumbnails that are not currently visible. With web technology embedded in a mobile platform's web view component, these particular issues would have been resolved automatically because resources are managed automatically. Nevertheless, automating such tasks is not necessarily an advantage. The previously mentioned Facebook application for iOS had performance issues with scrolling the (possibly long) news feed when the application was still implemented using HTML5. The extra layer of rendering introduced

by the embedded web view and CSS rules may play a role here. A deep hierarchy of HTML elements, as found in the Sencha Touch implementation of the sample application (p. 96), can also cause poor performance.

Another downside that I mentioned before and was possibly also an issue in Facebook's application is multithreading: complex operations and calculations should be done in the background because only one thread of execution is possible in JavaScript. Hence, long-running tasks can block the user interface. *Web workers*, a standard that is not yet implemented on all mobile platforms (mentioned on p. 132), could help here since these workers can run JavaScript code in parallel and the standard avoids typical problems (e.g. race conditions) by only offering certain global objects to a "worker", for instance no DOM manipulation methods, and offers a message-passing mechanism to communicate with the creator of the background task [57]. Mobile application frameworks can already provide platform-independent mechanisms for executing background tasks, for example by calling native code, but a purely web technology based solution seems not possible without named standard implemented across the platforms.

In conclusion, web technology can still be problematic when used in mobile applications because of performance issues and fragmentation in the support of new features. However, applications of small complexity, with low requirements to data processing and other background tasks, or with a simple user interface, can most probably be implemented using HTML without any of said issues.

After I finished evaluating the cross-platform and native solutions for development, I started implementing a business application for the company *Walnuss LLC*, targeted for Android and iOS at the moment. Since data processing is not very complex and other technical and design decisions did not enforce native development, I chose to use *PhoneGap* for the basic project architecture, together with a combination of several web-related frameworks and libraries. *jQuery Mobile* should be named as major component here. As *Sencha Touch* did not prove to be very performant in the evaluation, I chose *jQuery Mobile* instead because its architecture makes it possible to dynamically load content into the DOM instead of loading all screens at application start. The development of this application is not finished, but from the

[57] See [W3C2012a], sections 1.2, 4.6 and 5.3

experience until now, I can say that the chosen technologies seem fast enough even on slow devices. My recommendation is to evaluate whether web technology makes sense over the *long term* and can fulfill performance and other requirements at the *current* time. If a mobile web site is planned for an application, it can be considered to share some or most code between the web site and installable mobile applications. App store issues are also important: applications that are far from a “native experience” could get rejected. Altogether I can say that HTML5, related technologies and upcoming standards are a good step to shift development to a generally cross-platform and particularly cross-device development of applications (mobile and desktop). I assume that in the near future, many more mobile-targeted web frameworks and libraries will be developed, targeting simpler application architectures, improvements of application performance, differences in standards implementations and other potential pitfalls of web technology. Therefore, one should not rush to get into HTML-based applications if the currently available tools and frameworks do not fit requirements yet. A good way to get more involved in this trend may be to start developing a mobile web site first and later decide whether it should be ported to run as installable mobile application as well, potentially using additional features of the device or operating system.

7.4 Summary

I selected, compared and evaluated different cross-platform solutions and native SDKs with the motivation of figuring out whether native development should still be preferred over a cross-platform approach. The evaluation criteria were defined and weighted from the point of view of developers and small to medium businesses that want to begin mobile application development or consider a platform-independent solution. Therefore, I opted for cross-platform frameworks that are already stable and actively developed so that concrete recommendations could be given regarding the choice of a framework. Based on experiences from the implementation of a sample application and other facts about the current version of the compared solutions, I evaluated three cross-platform and two native solutions in the categories functionality, usability features, developer support, reliability, performance, deployment, supportability and costs.

The evaluation results showed that the Android and iOS SDKs still lead in some categories but the cross-platform solutions mostly fulfill typical expectations and cannot be considered immature anymore. My conclusion and recommendation is to take a decision depending on product requirements: Since cross-platform frameworks are often based on interpreted languages and/or web technology, performance is still inferior to native applications and framework-specific implementation details can exacerbate this problem (e.g. a complex user interface hierarchy resulting in slow rendering). Furthermore, the lack of native and familiar UI components may impede the creation of a highly usable frontend and thus might increase the effort needed for a positive user experience of an application. Nevertheless, I found that if these issues were considered and a decision in favor of the cross-platform approach has been reached, the two frameworks *Titanium* and *PhoneGap* are worthwhile solutions that satisfy many expectations.

7.5 Future work

As outlook regarding the development of cross-platform solutions, I expect that even more frameworks will enter the market, but probably not as many as in the last years since there are already several good alternatives available. Existing popular frameworks like Titanium and PhoneGap are backed by companies, and in the latter case also open source developers, and thus will surely be further developed, stabilized and extended with new features. I hope that performance issues and other factors will be taken into consideration so that the remaining gap between cross-platform and native development can be narrowed. Web technology and new extensions, like device APIs, may soon start to play a more important role as standards become stable and widely implemented. *IDC* states that around 80% of mobile developers plan to integrate HTML5 in newly released applications, and “over 80% of all [!] mobile applications will be wholly or in part based upon HTML5 by 2015” [IDC2012].

Research topics similar to my work have already been assigned by several universities and might also be conducted by companies that want to decide on their mobile strategy for the medium or long term. This will give insights from different perspectives and perhaps also long-term experiences that I could not provide in the given timeframe, for example a comparison of maintenance effort. Evaluation of a different set of frameworks, or new approaches that are not yet considered production-ready, are also imaginable.

References

[Adobe2012a] Lunny, Andrew (Adobe Systems Inc.): *[PhoneGap Build] Future Plugin Support*, http://community.phonegap.com/nitobi/topics/_phonegap_build_future_plugin_support, accessed: 2012-08-26, April 2012

[Adobe2012b] Adobe Systems Inc.: *PhoneGap Support*, http://www.phonegap.com/support, accessed: 2012-08-27, 2012

[Allen2010] Allen, Sarah / Graupera, Vidal / Lundrigan, Lee: *Pro Smartphone Cross-Platform Development: iPhone, BlackBerry, Windows Mobile, and Android Development and Distribution*, chapter 6, Apress, September 2010

[Anderson2012] Anderson, Chris: *Pro Business Applications with Silverlight 5*, section *Introducing Business Applications*, Apress, February 2012

[Apache2012] Apache Software Foundation: *Cordova Project Incubation Status*, https://incubator.apache.org/projects/callback.html, accessed: 2012-08-20, July 2012

[Appcelerator2011] Appcelerator Inc.: *Titanium Mobile Overview*, https://wiki.appcelerator.org/display/guides/Titanium+Mobile+Overview, accessed: 2012-03-26, November 2011

[Appcelerator2012a] Appcelerator Inc.: *Appcelerator's 300,000 Worldwide Developers to Gain Application Support on Blackberry 10 Platform*, press release, http://www.appcelerator.com/company/news/press/release-05-2012-rim, accessed: 2012-08-22, May 2012

[Appcelerator2012b] Appcelerator Inc.: *Titanium.UI.SearchBar*, http://docs.appcelerator.com/titanium/2.1/index.html#!/api/Titanium.UI.SearchBar, accessed: 2012-08-10, July 2012

[Appcelerator2012c] Appcelerator Inc.: *Plans & Pricing*, http://www.appcelerator.com/plans-pricing, accessed: 2012-08-17, 2012

[Apple] Apple Inc.: *Technical Support*, https://developer.apple.com/support/technical/, accessed: 2012-09-02, no year specified

[Apple2011] Apple Inc.: *About iOS Development*, https://developer.apple.com/library/ios/documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/IPhoneOSOverview/IPhoneOSOverview.html, accessed: 2012-08-14, October 2011

[Apple2012a] Apple Inc.: *App Store Review Guidelines*, https://developer.apple.com/appstore/resources/approval/guidelines.html, accessed: 2012-07-27, 2012

[Apple2012b] Apple Inc.: *iTunes Store: Frequently Asked Questions for iPhone, iPad, and iPod touch*, https://support.apple.com/kb/HT1689, accessed: 2012-08-31, May 2012

[Apple2012c] Apple Inc.: *Software License Agreement for OS X Mountain Lion*, http://images.apple.com/legal/sla/docs/OSX108.pdf, accessed: 2012-08-31, July 2012

[Apple2012d] Apple Inc.: *About Memory Management*, https://developer.apple.com/library/ios/documentation/cocoa/conceptual/MemoryMgmt/Articles/MemoryMgmt.html, accessed: 2012-10-10, July 2012

[Apple2012e] Apple Inc.: *iOS SDK Release Notes for iOS 6 beta 4*, https://developer.apple.com/library/prerelease/ios/#releasenotes/General/WhatsNewIniPhoneOS/Articles/iOS6.html, accessed: 2012-09-09, August 2012

[Applico2012] Applico LLC: *Native vs. HTML5 Apps: The Facebook Case Study*, http://www.applicoinc.com/native-vs-html5-apps-the-facebook-case-study/, accessed: 2012-09-18, September 2012

[Bishop2006] Bishop, Judith / Horspool, Nigel: *Cross-Platform Development: Software that Lasts*, in: *Software Engineering Workshop, SEW '06, 30th Annual IEEE/NASA*, pp. 119-122, draft available at: http://cs.up.ac.za/cs/jbishop/Homepage/Pubs/Pubs2006/CrossPlatformSoftware.doc, accessed: 2012-07-10, April 2006

[Brügge2004] Brügge, Bernd / Dutoit, Allen H.: *Objektorientierte Softwaretechnik mit UML, Entwurfsmustern und Java*, 2nd edition, pp. 146-154, Pearson Studium, August 2004

[Buschmann1996] Buschmann, Frank / Meunier, Regine / Rohnert, Hans / Sommerlad, Peter: *Pattern-Oriented Software Architecture Volume 1: A System of Patterns*, p. 435, John Wiley & Sons, August 1996

[Buse2002] Buse, Stephan: *Der mobile Erfolg – Ergebnisse einer empirischen Untersuchung in ausgewählten Branchen*, in: Keuper, Frank (ed.): *Electronic Business und Mobile Business: Ansätze, Konzepte und Geschäftsmodelle*, pp. 91-116, Gabler, Wiesbaden, 2002

[Busk2011] Busk, Nina Kirstine: *Smartphone Etiquette: Do we need a smartphone etiquette?*, http://ninakirstineis.me/wp-content/uploads/BMMS-Nina.pdf, accessed: 2012-07-14, 2011

[Chip2011] CHIP Online: *App-Stores: Windows Marketplace wächst um 65%*, http://www.chip.de/news/App-Stores-Windows-Marketplace-waechst-um-65_54848492.html, accessed: 2012-03-03, March 2011

[Constine2012] Constine, Josh: *Facebook For iOS App's Review Average Climbs From 1.5 Stars To 4 Stars In Three Weeks Since Relaunch*, http://techcrunch.com/2012/09/13/facebook-for-ios-review/, accessed: 2012-09-18, September 2012

[Facebook2012] Dann, Jonathan (Facebook Inc.): *Under the hood: Rebuilding Facebook for iOS*, https://www.facebook.com/notes/facebook-engineering/under-the-hood-rebuilding-facebook-for-ios/10151036091753920, accessed: 2012-08-24, August 2012

[Forrester2009] Forrester Research Inc.: *eCommerce Web Site Performance Today, An Updated Look At Consumer Reaction To A Poor Online Shopping Experience*, August 2009

[Forrester2012] Forrester Research Inc.: *Enterprise And SMB Networks And Telecommunications Survey, North America And Europe, Q1 2010*, 2010

[Gartner2011a] Jones, Nick (Gartner Research): *Choosing Cross-Platform Tools for Native Mobile Applications*, June 2011

[Gartner2011b] Gartner Research: *Predicts 2012: Application Development*, December 2011

[Gartner2012a] Gartner Research: *Gartner Says Worldwide Smartphone Sales Soared in Fourth Quarter of 2011 With 47 Percent Growth*, press release, https://www.gartner.com/it/page.jsp?id=1924314, accessed: 2012-03-03, February 2012

[Gartner2012b] Gartner Research: *Gartner Says Worldwide Media Tablets Sales to Reach 119 Million Units in 2012*, press release, http://www.gartner.com/it/page.jsp?id=1980115, accessed: 2012-07-16, April 2012

[Google2010a] Google Inc.: *Android 2.3 Compatibility Definition*, http://source.android.com/compatibility/android-2.3-cdd.pdf, 2010

[Google2010b] Block, Steve (Google Inc.): *Disable workers*, https://android.googlesource.com/platform/external/webkit/+/68698168e7547cc10660828f1fb82be7a8efa845, accessed: 2012-09-20, March 2010

[Google2012] Ball, Tom (Google Inc.): *J2ObjC: A Java to iOS Objective-C translator*, http://google-opensource.blogspot.de/2012/09/j2objc-java-to-ios-objective-c.html, accessed: 2012-09-14, September 2012

[IDC2011] IDC: *Worldwide Smartphone Market Expected to Grow 55% in 2011 and Approach Shipments of One Billion in 2015, According to IDC*, press release, http://www.idc.com/getdoc.jsp?containerId=prUS22871611, accessed: 2012-03-05, June 2011

[IDC2012] IDC: *IDC Highlights Key Milestones in Mobile HTML5 Development Update*, press release, http://www.idc.com/getdoc.jsp?containerId=prUS23480612, accessed: 2012-09-05, May 2012

[IEEE1990] IEEE: *IEEE Standard 610.12-1990, Standard Glossary of Software Engineering Terminology*, 1990

[ISO1998] International Organization for Standardization (ISO): *ISO 9241-11:1998, Ergonomic requirements for office work with visual display terminals (VDTs), Part 11: Guidance on Usability*, section 3, 1998

[Johnson1988] Johnson, Ralph E. / Foote, Brian: *Designing Reusable Classes*, in: *Journal of Object-Oriented Programming*, volume 1, number 2, pp. 22-35, June 1988

[MacFayden2011] MacFayden, Jesse: *PhoneGap Ecosystem (part 1 of 2) – Learn about the framework from two of the engineers behind it*, https://www.youtube.com/watch?v=Bsj4AoavhqQ#t=6m9s, accessed: 2012-03-18, June 2011

[Mata2012] Mata, Albert: *Introduction to internationalization using storyboards on iOS 5*, http://www.albertmata.net/articles/introduction-to-internationalization-using-storyboards-on-ios-5.html, accessed: 2012-07-22, February 2012

[mobl2011] No author specified (mobl project web site): *Features*, http://www.mobl-lang.org/features/, accessed: 2012-03-16, 2011

[MoSync2012] MoSync AB: *MoSync SDK – native mobile app development for multiple platforms using a single code base*, http://www.mosync.com/sdk, accessed: 2012-02-20, no year specified

[Motorola2011] Motorola Solutions: *Motorola Solutions Acquires Rhomobile and RhoElements Version 1 Launched*, http://www.rhomobile.com/blog/motorola-acquires-rhomobile-and-rhoelements-version-1-launched/, accessed: 2012-09-21, October 2011

[Motorola2012] Motorola Solutions: *RhoElements Introduction*, http://docs.rhomobile.com/rhoelements/rhoelements-introduction, accessed: 2012-06-13, no year specified

[OHA2010a] Open Handset Alliance: *Javascript to Java Bridge Throws Exception on Android 2.3*, https://code.google.com/p/android/issues/detail?id=12987, accessed: 2012-06-12, December 2010

[OHA2010b] Open Handset Alliance: *Orientation does not change from landscape to portrait on Emulator on 2.3*, https://code.google.com/p/android/issues/detail?id=13189, accessed: 2012-07-14, December 2010

[Paananen2011] Paananen, Timo: *Smartphone cross-platform frameworks*, bachelor's thesis, https://publications.theseus.fi/bitstream/handle/10024/30221/110510_Thesis_Timo_Paananen.pdf, accessed: 2012-01-24, April 2011

[Pasetti2002] Pasetti, Alessandro: *Software Frameworks and Embedded Control Systems*, p. 8, in: *Lecture Notes in Computer Science*, volume 2231, Springer Berlin/Heidelberg, 2002

[PhoneGap2012] Adobe Systems Inc.: *PhoneGap 1.5.0 Documentation*, http://docs.phonegap.com/en/1.5.0/index.html, accessed: 2012-03-18, March 2012

[Rhomobile2010] Rhomobile Inc.: *Rhodes 2.0 Released! Rhodes apps continuing to be accepted on App Store*, http://www.rhomobile.com/blog/rhodes-2-0-released-rhodes-apps-continuing-to-be-accepted-on-app-store/, accessed: 2012-08-12, June 2010

[Sencha2012] Avins, Jamie (Sencha Inc.): *Sencha Touch 2.0 – Built for Amazing Apps*, http://www.sencha.com/blog/announcing-sencha-touch-2/, accessed: 2012-03-17, March 2012

[vanVliet2008] van Vliet, Hans: *Software Engineering: Principles and Practice*, chapter 14, John Wiley & Sons, June 2008

[vdHelm2010] von der Helm, Daniel: *[Cross-Platform-] Development of Mobile Applications*, http://www.slideshare.net/dvdh/survey-results-cross-platform-development-of-mobile-applications, accessed: 2012-03-04, October 2010

[vdNeyen2011] von der Neyen, Julian: *Mobile Business Applications*, in: Amberg, Michael (ed.) / Lang, Michael (ed.): *Innovation durch Smartphone & Co: Die neuen Geschäftspotenziale mobiler Endgeräte*, 1st edition, pp. 15-32, Symposion Publishing, Düsseldorf, March 2011

[VisionMobile2012]	VisionMobile Ltd.: *Developer Economics 2012, The new mobile app economy*, chapter 3, http://www.visionmobile.com/product/developer-economics-2012/, accessed: 2012-06-20, June 2012
[W3C2012a]	W3C: *Web Workers, W3C Candidate Recommendation 01 May 2012*, http://www.w3.org/TR/2012/CR-workers-20120501/, accessed: 2012-09-20, May 2012
[W3C2012b]	W3C: *Device APIs Working Group*, http://www.w3.org/2009/dap/, accessed: 2012-09-21, September 2012
[Whinnery2012]	Whinnery, Kevin: *Comparing Titanium and PhoneGap*, section *How Titanium Works*, http://kevinwhinnery.com/post/22764624253/comparing-titanium-and-phonegap, accessed: 2012-05-25, May 2012
[XMLVM]	No author specified (XMLVM project web site): *Overview: Android to iPhone*, http://www.xmlvm.org/android/, accessed: 2012-03-26, no year specified
[Zuckerberg2012]	Zuckerberg, Mark: Interview at TechCrunch Disrupt conference, video recording available in: Constine, Josh: *Zuckerberg Shows He's The Right Man For The Job. Now That Job Needs Doing*, http://techcrunch.com/2012/09/11/zuckerberg-the-leader/, accessed: 2012-09-18, September 2012

Appendix

A. Details of the web service used in the sample application

The list of orders, including old orders and the current one (if created), is returned by the web service in the following way:

Request	GET http://hostname/service-uri/orders/

Response example	HTTP 200, JSON data, Content-Type “application/json”
	
	`{`
	`  "orders": [`
	`  {`
	`  "id": 1,`
	`  "pictureIds": [1, 2, 4, 8],`
	`  "storeId": 1,`
	`  "submissionDate": "2012-04-15T17:14:23+01:00"`
	`  },`
	`  {`
	`  "id": 2,`
	`  "pictureIds": [9, 10],`
	`  "storeId": null,`
	`  "submissionDate": null`
	`  }`
	`  ]`
	`}`

Expected errors	none

Thumbnails can be downloaded for a given picture ID. The size defines the number of pixels of the picture’s longer side and may be between 5 and 500. If the original picture is smaller than the requested size, it will not be enlarged and instead, the web service just returns the original picture.

Request	GET http://hostname/service-uri/picture/**id**/thumbnail/?size=**size** • **id** = picture ID • **size** = desired size of the longer side of the thumbnail
Response	JPEG image of the specified size of the longer side
Expected errors	• Invalid ID or size (HTTP 400) • Picture not found (HTTP 404)

For uploaded pictures, the web service accepts only JPEG pictures, up to a maximum file size of 2 MB, with allowed resolutions of 1-4000 pixels per side. The picture is automatically added to the current order if it does not already belong to that order. Should there be no current order, the service will automatically create one. The up-to-date information about the order is returned in the response. Uploading is done via the following request:

Request	PUT http://hostname/service-uri/pictures/ • Upload should be encoded as “multipart/form-data” • JPEG image should be sent as raw data, i.e. without additional encoding or compression

Response example	HTTP 201, JSON data containing the ID assigned to the picture, Content-Type “application/json”
	
	`{`
	`  "picture": {`
	`  "id": 4`
	`  },`
	`  "order": {`
	`  "id": 2,`
	`  "pictureIds": [3, 4],`
	`  "storeId": null,`
	`  "submissionDate": null`
	`  }`
	`}`

Expected errors	• Invalid file format or size (HTTP 400) • Quota exceeded, server running out of space (HTTP 500)
Alternative request	Since the Rhodes framework did not offer a solution for uploading files via PUT requests (see p. 81), the following alternative URL was added as workaround. It works in the same way as defined above. POST http://hostname/service-uri/pictures/put-by-post-workaround/

The web service determines up to five close stores around a certain position in the world.

Request	Two options are available: GET http://hostname/service-uri/stores/by-location/?lat=**lat**&lng=**lng** GET http://hostname/service-uri/stores/by-location/?loc=**loc** • **lat** / **lng** = latitude and longitude of the location in degrees, formatted with a dot as decimal separator, e.g. “lat=37.820904&lng=-122.476702” for the Golden Gate Bridge, San Francisco • **loc** = location such as an address, city or public place, e.g. “Skeppsholmen, Stockholm”

Response example	HTTP 200, JSON data containing up to 5 stores, Content-Type “application/json”
	
	`{`
	`  "stores": [`
	`  {`
	`  "id": 1,`
	`  "name": "MallMart"`
	`  "address": "Marienplatz 1, M\u00fcnchen",`
	`  },`
	`  {`
	`  "id": 2,`
	`  "name": "LITTLE"`
	`  "address": "Leopoldstra\u00dfe 144, M\u00fcnchen",`
	`  }`
	`  ]`
	`}`

Expected errors	• Missing parameters *lat*/*lng* or *loc* (exactly one of the options must be chosen), or invalid format of lat/lng (HTTP 400) • If the location *loc* cannot be found or an error occurred while trying to get the geographical position of an address, any 5 stores are returned (no error code or information given)

An order can be submitted if at least one picture was added to it and the order is not yet marked as submitted. Submission assigns a date and the selected store to the current order.

Request	POST http://hostname/service-uri/order/**id**/submit/ • Content-Type header must be “application/x-www-form-urlencoded” • Request body is a URL-encoded representation of the username, password and store ID `username=user&password=pass&storeId=3`

Response example	HTTP 200, JSON data containing the order in the same structure as used for list items in the query for list of orders, Content-Type “application/json”
	
	`{`
	`  "order": {`
	`  "id": 1,`
	`  "pictureIds": [1, 2, 4, 8],`
	`  "storeId": 3,`
	`  "submissionDate": "2012-04-15T17:14:23+01:00"`
	`  }`
	`}`

Expected errors	• Missing parameter username or password (HTTP 400) • Order already submitted or no picture contained (HTTP 400) • Store not found (HTTP 400) • Order not found (HTTP 404) • Note that no authentication is done as mentioned before, username and password are simply ignored by the web service

B. List of considered frameworks

The following table represents a non-exhaustive list with information based on web sites and other public information found about the frameworks at the time of writing, collected to the best of my knowledge (mistakes possible). The three cross-platform frameworks presented in the thesis are not listed here. A hint "can be used with PhoneGap" only means that the vendor officially claims this, not that other frameworks cannot be combined with PhoneGap. "UI only" means that no device functionality is supported; the framework might still include other non-UI features.

A list with more frameworks and details can be found on Wikipedia at: https://en.wikipedia.org/wiki/Mobile_application_development#Platform_development_environment [58]

MoSync	• C++ and/or HTML5 development • 9 mobile platforms supported • Native user interface components can be used, no UI library for HTML included • GPL 2 license or commercial licenses (free without support) • http://www.mosync.com/, accessed: 2012-02-12
iUI	• JavaScript and CSS framework • 6 mobile platforms supported • UI only, allows theming for different platforms and supports transitions between screens defined as HTML elements • BSD 3-clause license (free for commercial use) • http://www.iui-js.org/, accessed: 2011-12-12
mobl	• Uses new programming language *mobl* that compiles to an HTML-based application • WebKit-based browsers supported (thus at least Android, iOS, WebOS supported) • Application logic and UI written in mobl language • Supports (only) the camera module of PhoneGap and geolocation functionality, but no other device functionality • MIT license (free for commercial use) • http://www.mobl-lang.org/, accessed: 2011-12-12

[58] Accessed: 2012-08-29, not used as source

Application Craft	• Web technology • Development is done with online IDE • 5 mobile platforms supported, 3 more to come • UI and application logic only • Can be used with PhoneGap • Free with advertising, else from $45/month or $450/year • http://www.applicationcraft.com/, accessed: 2011-12-12
AppFurnace	• Web technology • Development is done with online IDE • Android and iPhone supported • Free until publication of an application, then from £500 per application for a single platform • http://www.appfurnace.com/, accessed: 2011-12-12
Wink toolkit	• JavaScript based • 5 mobile platforms supported • UI only • BSD 3-clause license (free for commercial use) • http://www.winktoolkit.org/, accessed: 2011-12-12
Adobe AIR	• Runtime for deployment of Flash/Flex or HTML-based applications • May require Adobe AIR runtime installed on device • Supports some device features and UI • Seems to be free • http://www.adobe.com/products/air.html, accessed: 2011-12-22
Mono for Android / MonoTouch	• C# / .NET • Two products, fostering reuse of .NET code with platform-specific views • Supports Android and iOS, respectively • Device functionality and UI • Commercial, from $399 each • http://xamarin.com/, accessed: 2011-12-22
jQTouch	• JavaScript and CSS framework • WebKit-based browsers supported (thus at least Android, iOS, WebOS supported) • UI only (theming, transitions, components) • MIT license (free for commercial use) • http://jqtouch.com/, accessed: 2012-02-20
jQuery Mobile	• JavaScript and CSS framework • 8 mobile platforms supported • UI only, allows theming and supports single-page architecture with transitions between defined as HTML elements • Choice between MIT or GPL license (free for commercial use) • http://jquerymobile.com/, accessed: 2011-12-14
Corona	• Lua as programming language • 4 mobile platforms supported • Device functionality and UI, many APIs targeted at game development • Free until publication of an application, subscription costs $199/year • http://www.coronalabs.com/, accessed: 2011-12-14

Marmalade	• HTML or C++ • 4 mobile platforms supported • Device functionality, UI components (native and HTML) and theming, game development features • Can be used with PhoneGap • Free for non-commercial use or evaluation, plans from $149 (iOS & Android) or $499 (all platforms) • https://www.madewithmarmalade.com/, accessed: 2011-12-14
LungoJS	• JavaScript and CSS framework • Supported platforms not named • UI only, theming and components • GPL 3 license (free for commercial use) • http://www.lungojs.com/, accessed: 2012-02-03
Trigger	• Mainly JavaScript-based, using web technology • Builds use web service (from command line tools) • Android, iOS and Windows Phone supported • Device functionality and UI (native components supported) • One month free trial, then from $39/month • https://trigger.io/, accessed: 2012-03-18
Dojo Mobile	• JavaScript and CSS framework (part of Dojo framework) • WebKit-based browsers supported (thus at least Android, iOS, WebOS supported) • UI only, theming and components • BSD 3-clause license or Academic Free License 2.1 (free for commercial use) • http://dojotoolkit.org/features/mobile, accessed: 2012-03-16
MWF	• JavaScript and CSS framework • No exact information about platform support found • Seems to require PHP • UI only • Free to use, special license (formulation for commercial use is unclear) • http://mwf.ucla.edu/, accessed: 2012-03-21
Jo	• JavaScript, using web technology • At least 4 mobile platforms supported • UI only, theming and components • Can be used with PhoneGap • BSD 2-clause license (free for commercial use) • http://joapp.com/, accessed: 2012-03-21
XMLVM	• Cross-compilation of programs, e.g. Android to iPhone (still “in an early development phase”) • Android and iPhone projects seem possible • Not a framework, but goal is translation of whole applications • LGPL license (free for commercial use) • http://www.xmlvm.org/, accessed: 2011-12-22

IBM Worklight	• Seems to combine any of the frameworks Dojo, jQuery Mobile and Sencha Touch with PhoneGap and additional APIs for a special application server • Supports Android, iOS, BlackBerry • Commercial license, pricing unclear • http://www.ibm.com/software/mobile-solutions/worklight/, accessed: 2012-04-15
In-the-box	• Port of Dalvik VM and Android 2.3 API to iOS • Apache Software License 2.0 (free for commercial use) • http://www.in-the-box.org/, accessed: 2012-04-16
Resco MobileApp Studio	• C# / .NET • 4 mobile platforms supported • Few device features supported, UI components included • Commercial license, from $999 for a single platform • http://www.resco.net/developer/mobileappstudio/overview.aspx, accessed: 2012-05-01
JUCE	• C++ • Supports iOS and Android • UI only, plus audio/video functionality • GPL license, or commercial license from £399 • http://www.rawmaterialsoftware.com/juce.php, accessed: 2012-05-01
The-M-Project	• JavaScript framework, using web technology • Integrates PhoneGap, jQuery Mobile and other libraries • Device functionality and UI components • MIT or GPL 2 license (free for commercial use) • http://the-m-project.org/, accessed: 2012-05-04
MonoCross	• C# and web technology • Supports Android, iOS, Windows Phone and platforms using WebKit • Device functionality and UI • MIT license (free for commercial use), but MonoTouch or Mono for Android may be necessary • http://www.monocross.net/, accessed: 2012-05-04
Unify	• Mainly JavaScript, using web technology • Android, iOS and WebOS supported • Integrates PhoneGap and the qooxdoo JavaScript framework • Device functionality and UI (with theming) • MIT or Apache Software License 2.0 (free for commercial use) • http://www.unifyjs.com/, accessed: 2012-05-05
Kendo UI Mobile	• JavaScript and CSS framework • Android, iOS and BlackBerry supported • UI only, components and theming • Can be used with PhoneGap • Commercial license from $199 • http://www.kendoui.com/, accessed: 2012-05-18

AppConKit	• No information found about license, architecture, programming language and whether device features are supported • Seems to rely on server-side application part with a client installed on the mobile device • Android and iOS supported • http://www.weptun.de/appconkit/, accessed: 2012-06-21

Other frameworks that were found later and thus not considered for evaluation anymore:

jqMobi / jqUI	• JavaScript framework, using web technology • Android and iOS supported • UI only • MIT license (free for commercial use) • http://www.jqmobi.com/, accessed: 2012-07-14
J2ObjC	• Translates Java classes to Objective-C, supporting "the full Java 6 language and most of its runtime features that are required by client-side application developers" [Google2012] • Android, iOS and web applications (using GWT) supported • No device or UI features, goal is code reuse • Apache Software License 2.0 (free for commercial use) • https://code.google.com/p/j2objc/, accessed: 2012-09-14
mgwt and gwt-phonegap	• Based on GWT, a framework including a Java-to-JavaScript cross compiler for building web applications using Java code. mgwt provides UI components specifically for mobile platforms. gwt-phonegap offers a Java API for PhoneGap features. • WebKit-based browsers supported (thus at least Android, iOS, WebOS supported) • Device functionality (gwt-phonegap) and UI (GWT and mgwt) • Apache Software License 2.0 (free for commercial use) • http://www.m-gwt.com/, accessed: 2012-09-19

C. Time measurements of sample application implementations

Time measurements include the time needed for project setup (but not for installing and setting up the framework), development, researching issues and testing the application against the requirements as specified in chapter 5.

Framework	**Time in hours and minutes**
Titanium	22:59 h
Rhodes	23:06 h
PhoneGap / Sencha Touch	24:15 h
Android (native SDK)	14:47 h Note that the large difference to the other frameworks can be explained by my prior knowledge about Java and Android application development.
iOS SDK	19:28 h