LUNATECH
RESEARCH

Native Cross-platform Mobile Application Development

by
W. de Kraker (0815283)

CMI-Program *Informatics* – Rotterdam University

August 13, 2012

First supervisor *Dhr. Y. S. Tjang*
Second supervisor *Dhr. A. Chamani*

Abstract

Nowadays mobile devices are vastly integrated into modern society. They bring us one step closer to satisfy our ever growing need to have information available anytime, anywhere. To help gain access to information on mobile devices we use software applications, so called *apps*.

However, the fragmented nature of today's mobile ecosystem poses a challenge for developers to develop apps which are suitable to run on all mobile devices, since there is no de facto standard in *cross-platform* app development.

Currently there are several solutions available to solve the cross-platform challenge.

Lunatech, having expressed its interest in mobile app development, would like to know which solution, *if any*, suits Lunatechs needs. A study has been set up in order to resolve this question, the results of which are laid out in this thesis.

Versions

Version	Date	Author	Details
0.1	12/07/2012	W. de Kraker	Intial draft
0.2	20/07/2012	W. de Kraker	Improved main research structure
0.3	08/08/2012	W. de Kraker	Changes based on feedback from Mr. Y.S. Tjang
0.4	12/08/2012	W. de Kraker	Changes based on feedback from Mr. S. de Kaper
1.0	14/08/2012	W. de Kraker	Final version

Table 1: Version history

Credits

Contents

Background

Lunatech Research B.V

Lunatech provides application development services, completely based on open-source web and Java technologies and open standards. They are early adopters of new technology, and use cutting-edge frameworks and tools. To stay up-to-date, their developers have the opportunity to research, try new technologies and contribute to open-source projects. The company consists mainly of software developers. Everyone (except the director) writes code, on top of which some staff have a secondary management role, and the staff who will deliver a project interact with the customer directly.

Rotterdam University of Applied Sciences (Hogeschool Rotterdam)

Rotterdam University is one of the major Universities of Applied Sciences in the Netherlands. Currently almost 30,000 students are working on their professional future at the university. The university is divided into eleven schools, offering more than 80 graduate and undergraduate programmes in seven fields: art, technology, media and information technology, health, behaviour and society, engineering, education, and of course, business.[**?**]

Wolfert de Kraker

Wolfert de Kraker is a 4th year computer science student at the Rotterdam University of Applied Sciences, with 2 years (professional) experience in mobile application development.

Introduction

Problem statement

To meet their customer's demands Lunatech needs to develop cross-platform mobile applications. Currently these applications are developed using web technologies such as HTML and Javascript. A mobile application developed in this manner is referred to as web app because it runs in a browser based environment and is often hosted at a web server rather than downloaded to the device itself.

The problem with web apps is that they lack in user experience. This is mainly due to the browsers' sandbox environment in which the app runs. Every platform has its own set of recognizable user interface elements, but these cannot be accessed from within the browser environment. To overcome this problem web app developers build the user interface in HTML. As a result of this the app does not feel native to the user because its style does not match the style of the platform. Web app developers often try to mimic the native look-and-feel, but a web app does not live up to the level of user experience which a native app offers

The direct alternative to web apps are native apps, native apps are written using technologies proprietary to each platform, hence the term *native*. What these applications lack in terms of cross-platform support they make up in terms of user experience. A native app has access to all the platforms libraries and can rely on user interface elements provided through these libraries.

Lunatech would like to know how to make use of the look-and-feel from native apps with the cross-platform support of web apps.

Research questions

Main research question:

- *How to develop a cross-platform mobile application while retaining the native look-and-feel?*

Sub research questions:

- *Which solutions to cross-platform mobile application development currently exist?*
- *Which of these solutions offer the defined native look-and-feel?*
- *Are these solutions viable for commerical useage?*
- *Is it viable to implement a custom solution to cross-platform mobile application development?*

Research method

This paragraph will describe the method used during the research.

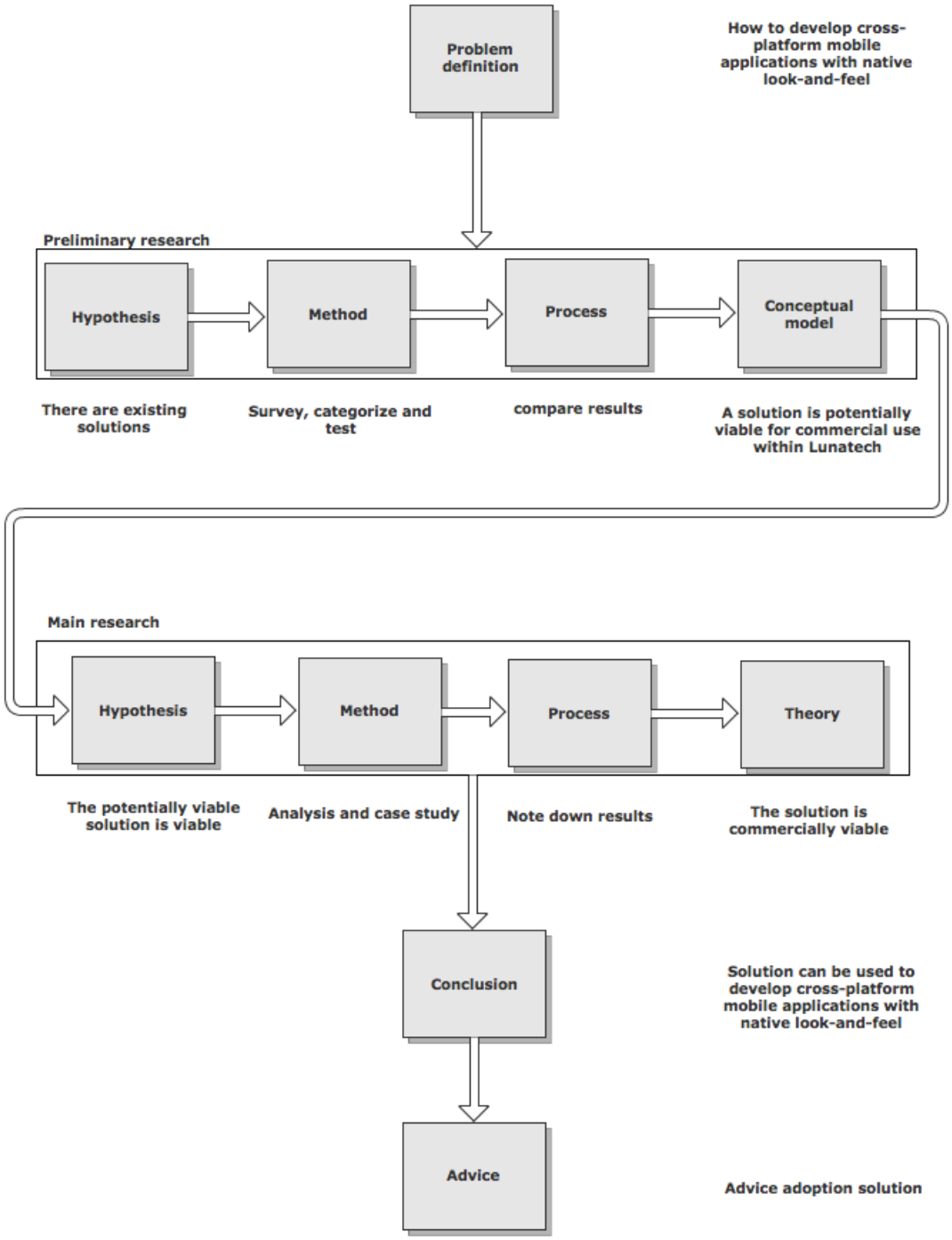
First the context of the main research question needs to be determined. This is done by defining the concepts *native look-and-feel* and *cross-platform*. The *native look-and-feel* is defined by the a set of criteria related to the user experience while *cross-platform* is defined as the scope of which platforms should be targeted.

Once the context of the main research question has been defined it is needed to research which solutions fit within its scope. During a preliminary research a market survey will have to be performed in order to determine which solutions to cross-platform app development are available on today's market. Each found solution will be given a closer look and categorized based on requirements provided by Lunatech. Once the categorization is complete the solutions will be filtered based on how far the requirements have been met.

A comparison test will be performed in order to determine which solution is potentially viable for use by Lunatech. This test will consist of a short analysis per solution based on Lunatech's criteria, and a practical part where a benchmark application will be built. The results will be compared and the solution which offers the most complete set of features will be deemed potentially viable for use by Lunatech. At this point a decision has to be made whether to continue with the potentially viable existing solution or research a custom solution. If no existing solution meets the criteria set by Lunatech the main research will focus on implementing a custom solution.

To determine if the chosen solution is viable for commercial use by Lunatech a case study will be performed based on a realistic scenario Lunatech might encounter. During this case study the solution will be analyzed and evaluated.

The results of the case study and the analysis will contribute to answering the main research question. In addition to answering the main research question a recommendation will be given to Lunatech regarding the possible adaptation of the solution for commercial usage.



Research method diagram

Defining the context

The goal of this chapter is to define the context of the main research question.

"How to develop a cross-platform mobile application while retaining the native look-and-feel?"

Accordingly this chapter will define:

- the scope of *cross-platform*
- the concept of the *native look-and-feel*

Cross-platform

This section will define the scope of *cross-platform*. This will be done in two parts, first term platform will be defined after which will determined which platforms fall within the scope.

Platforms

In order to determine which platforms should be supported for mobile development the following criteria have been provided by Lunatech:

1. *Platform type*
   The platform has to be mobile, equipped with a touchscreen and support 3rd party applications.
2. *Platform marketshare*
   The platform should have 10% market share in the european continent.
3. *Platform marketshare trend*
   The market share trend from the past 6 months should not depict a trend directed below the set threshold of 10% within the next 6 months.

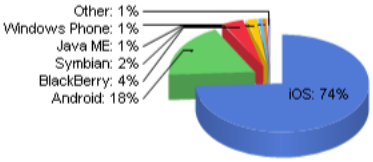
The platform type criteria is based on Lunatech's requirement to support smartphones and tablet computers. A smartphone can be defined as a smartphone is a next-generation, multifunctional cell phone that provides voice communication and text-messaging capabilities and facilitates data processing as well as enhanced wireless connectivity.[**?**] Tablet computers, commonly referred to as tablet PCs, are wireless portable personal computers which provide the user with a touchscreen to access or process information.[**?**] The platform market share is the percentage per platform representing its share of the platforms market. Platform market share trend defined as the general course or prevailing tendency of the platforms market share.

Platform marketshares & trend

As mentioned in the previous paragraph, platforms included in the cross-platform scope need to have a 10% in share in the european smartphone market.

There are several sources which provide statistical data over mobile platform market share. One of these is *Net Market Share*. Net Market Share defines itself as the standard for tracking key internet technology usage market share. This means their statistical data is usage based, because it is aggregated from tracking users on the internet. Net market share aggregates its data from over 40,000 websites with approximately 160 million visitors per month.[**?**]

An alternate source for market share data is *Gartner*. Gartner describes itself as being the world's leading information technology research and advisory company.[**?**] In an annual publication Gartner publishes a smartphone market share report. In contrast to Net Market Share this report has been based on manufacture unit sales data. Although very accurate, it does not reflect real usage nor is it an up to date source, i.e. the latest publication dates from august 2011.[**?**] Therefore Net Market Share is to be preferred over Gartner.

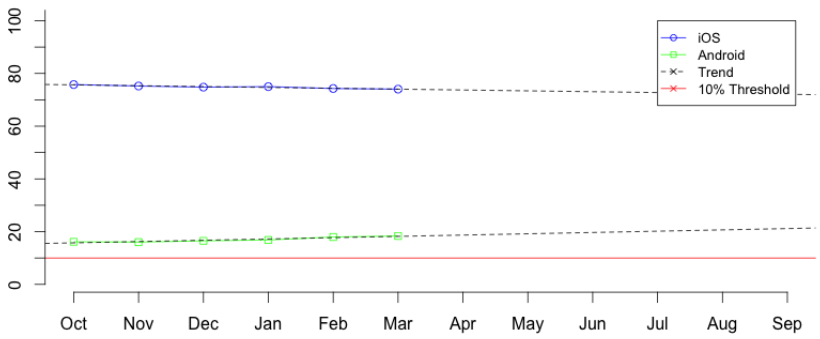


Operating System	Market Share in October 2011	Market Share in March 2012
iOS	75.78	74.04
Android	16.14	18.36
BlackBerry	3.56	3.84
Symbian	2.69	1.75
Java ME	0.87	0.83
Windows Phone	0.27	0.68
Bada	0.35	0.29
Windows Mobile	0.27	0.14
Kindle	0.00	0.05
Samsung	0.06	0.03
LG	0.01	0.01

Table 1.2: Market share in the European continent as of October 2011 and March 2012[**?**]

iOS and Android both cover over 10% of the market, together they are good for 92.40% of the total mobile market as of march 2012 in the european continent.

To assess the likelihood that these market shares will not hit below the 10% market share a R plot has been generated to visualize the trends:



Market share trends between October 2011 and September 2012, generated using R

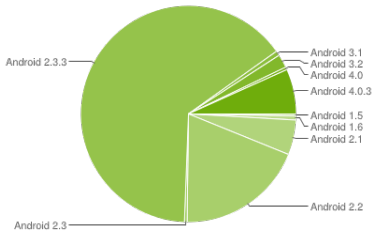
Platform versions

In order to determine which versions per platform will be supported Lunatech has stated that a version must have at least a 10% user share of the platform.

For Android this means the versions:

- 2.2 - *Froyo* (19.1%)
- 2.3.3 to 2.3.7 - *Gingerbread* (64.6%)

This statement is based on data gathered by the number of Android devices that have accessed Google Play[1] within a 14-day period ending on the data collection date noted below. This source was chosen because it originates from an offical publication by Google.[**?**]



Android platform version distribution, as of June 1st, 2012.[**?**]

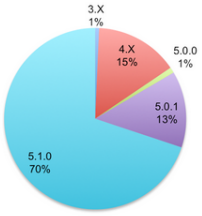
For iOS this includes the versions:

- 4.3 (15.0%)
- 5.1 (84.0%)

Apple does not have an official channel trough which version statistics are published, however there is an alternate way to generate statistics. When an iOS app is updated a separate version of the update is released per iOS version, this allows the developer to count the number of updates downloaded per version. The chosen statistics originate from iOS developer David Smith. There

[1] Google Play is the offical app distribution platform for Android

are two main arguments to perfer his statistics over those of other iOS developers, his audiobook app is downloaded over 100.000 times weekly and his statistics are up to date.[**?**]



iOS platform version distribution, as of May 26th, 2012.[**?**]

Conclusion

Apple iOS and Google Android both adhere to the criteria set by Lunatech and will be included in the scope of cross-platform. In conclusion *cross-platform* support is defined as compatible to run on iOS 4.3 or greater and Android 2.2 or greater.

Native mobile applications

This paragraph will define the paradigm of *native look-and-feel*.

A native application

A native application is an application inherent to the platform for which it was built using techniques proprietary to the platform. For example, an iOS application is native when written in Objective-C and an Android application is written in Java. Native applications are typically fast and can access the device's native API's.

The native look-and-feel

When written in the native framework for a platform a mobile application receives access to the available public libraries of the platform. These libraries include those which provide the developer with a pre-fabricated set of user interface components. These can be seen as the building blocks for the graphical user interface on that platform. When used, the general style of the mobile application gains constancy to the overall user interface design of the platforms operating system. This gives an application its native look, which in turn participates to the *native feel*.

The *native feel* of a mobile application can be defined as the speed in which the user interface elements, the responsiveness of user interface elements to touch events, and smoothness of the animation in which the user interface elements are moved. A native mobile application has the advantage of hardware acceleration. This means its code has been precompiled and directly executed by the device CPU, rather than having to be interpreted by the device's browser. As a result of this the user interface elements are rendered faster and it *feels* smooth.

Alternative mobile application types

Web applications

A mobile web application is an application developed with web technologies such as JavaScript and HTML with CSS. It is in fact nothing more than a website designed to fit on a mobile device's display, often they resemble the style of a native application rather than a traditional website. These applications are built with a JavaScript library to add support for scrolling and handling events. Touch events are handled via user interface elements provided by the library. Examples of these libraries include jQtouch, SenchaTouch

Hybrid applications

A hybrid application in mobile development refers to an application which use a native shell wrapped around web app. There are generally two forms of native shells, the first is a *webview* and the second a native framework which exposes an JavaScript API to provide the web application access to otherwise native API's. The following paragraphs will describe these types in more detail.

Webview-based hybrid applications
A web-view based hybrid application is a web based mobile application wrapped in a web-view. A web-view is a view or element which acts like a browser would, e.g. it is able to render HTML and run javascript. It is readily available in the native libraries. The advantage of a web-view based hybrid application over an normal web application is that it can be published via the devices native application publishing platforms, i.e. a web-view based hybrid application targeted for the iPhone can be placed in the Apple Appstore.

Worklight is an example of a framework which can be used to develop web-view based hybrid applications.

Framework hybrid applications
A Framework based hybrid application is a web-view based application built upon a framework which provides an API to allow the application access to otherwise native API's. The framework is written in the platforms native programming language making it possible to access the native API, such as reading contact list, composing of text messages, full access to the location API, etc.

PhoneGap is an example of a framework which can be used to develop mixed hybrid applications.

Comparison

Web applications are written entirely in HTML, CSS and JavaScript. This means that web applications are executed by the mobile browser and therefore cross-platform by default. As a consequence web applications are relatively quick and easy to develop, however their performance is only as fast as that of the platform's browser.

Webview-based hybrid applications, the app's source code consists of web code executed within a native wrapper that is provided by a framework. The advantage over web applications is that the app can be distributed and installed in the same manner that native applications can.

Framework hybrid applications, the developer augments the web code with a Javascript API to create unique features and access native APIs that are not yet available via the browser, such as bluetooth, GPS and others.

Native Application are platform-specific. Developing a native app requires unique expertise and knowledge for each platform and are therefore pricey and time consuming to develop, however a native app delivers the highest user experience of all approaches.

Different types of mobile applications[**?**]

Conclusion

A natively written mobile application provides the user with an experience immersed to that of the level of the device's operating system, this is due to two reasons:

- *Performance*
  a native application is faster because it has direct access to memory and CPU.
- *Looks*
  a native application looks and feels more coherent to the device's operating system because it is able to make use the of the provided user interface elements.

Preliminary research

The goal of the preliminary research is to determine whether or not a solution for cross-platform mobile application development that meets Lunatech's criteria already exists.

Introduction

In today's industry there exist several cross-platform mobile application development frameworks which offer a solution to cross-platform problem. All of these frameworks provide a solution for crossing the bridge between platforms. In order to determine which framework could be adopted by Lunatech for mobile development the following criteria have been determined for comparison:

1. *Platform support*
   Which platforms and their versions are supported by the framework.
2. *Native UI support*
   Whether or not native user interface elements are supported for each supported platform.

secondary criteria:

1. *Programming language*
   Which programming language is used to develop using the framework.
2. *IDE* (Integrated Development Environment)
   Which IDE can be used to develop using with the framework.
3. *License type*
   Which license types are available.
4. *Application type*
   Which type of mobile application is produced using this framework.

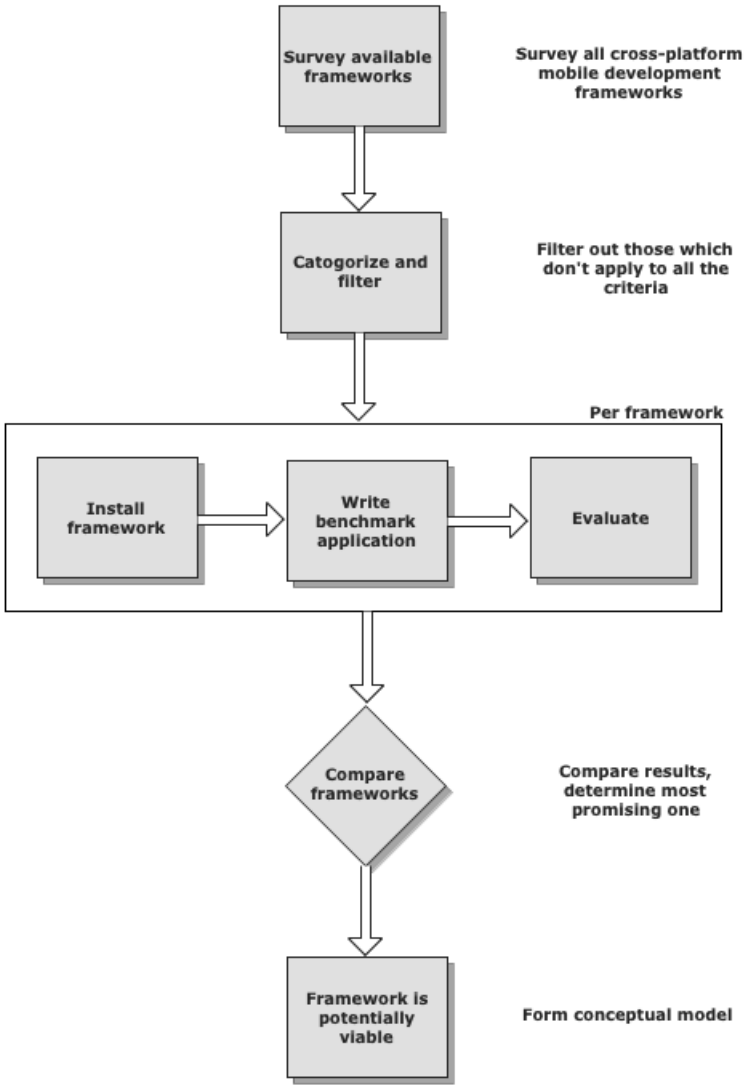
The cross-platform criterium is based on Lunatechs requirement to build mobile applications for the operating systems have at least a 10 percent market share in the European continent. Second comes the support for native user interface elements. Together these criteria form the essence of the main research question: *"How to develop a cross-platform mobile application while retaining the native look-and-feel?"* The remaining criteria are of secondary importance, they will provide more detailed means to compare the frameworks which offer native user interface support.

Based on the previous statement and criteria a hypothesis can be formed:

> There is a solution for cross-platform mobile application development that meets Lunatech's criteria.

Method

In order to test the hypothesis it is necessary to conduct a market survey to determine which solutions to cross-platform app development are available on todays market and which of those meet the criteria. Each solution will be evaluated during a test in which a benchmark application will be developed using the solution. The resulting evaluations will be compared. The solution which offers the most complete set of features will be deemed potentially viable for commercial use by Lunatech.



Preliminary research structure

Selection

There are over 30 frameworks which offer cross-platform development of mobile applications[**?**]. These frameworks have been added to an initial list.[2] This initial list contains only frameworks which adhere to the requirement of supporting cross-platform mobile application development. The frameworks in the list are categorized by the first set of criteria.

To determine which frameworks should be included in the comparison we'll take a closer look at each and filter out those who do not adhere to criteria of supporting native user elements. This should leave us with an acceptable number of frameworks to compare. Acceptable in this scenario will be five frameworks, which is the goal because the timeframe of the internship doesn't allow for more.

Benchmark app

In order to evaluate the frameworks it is necessary to build a application with all of them. The application will have the same requirements on each platform. Before the evaluation process begins a native application has to be written, the results of building it natively will serve as benchmark setting during the evaluation, hence the term: *benchmark app*

The application should be related to what Lunatech might anticipate for mobile contracts. In this case the use case was taken from *Stager*. Stager is a modern web-based resource planning and ticketing application to help manage live music events. Lunatech took the opportunity to use the relatively new Play framework to build a web application with an HTML and Java architecture. The application should be a publicity app, allowing for users to view published events and share them on social media.

This concludes that the app should be able to:

- display events from the xml based atom feed,
- show event details,
- allow for sharing of events to social media

The design will be based on the website of an live music venue which already uses Stager to publish their events. WORM.ORG

For the native version of the benchmark application was chosen to use the iOS platform simply for practical reasons, because there was an iOS device directly available for testing. And because I have extensive experience programming native iOS applications.

It took three days to complete the benchmark app, of which the majority of the time was spent on parsing data using x-path, which I had not used before.

Evaluation process

Estimated is that it takes four days to do experiments with a framework, this is based on N+1 in which N is the number of days it took to develop the benchmark app + 1 to familiarize with the framework. If in those four days I am unable to complete the benchmark app, it is still a result. The experiment consumes a timespan of 32 hours per framework and consist of the following activities:

[2] see appendix: *Existing solutions*

Figure 1.1: iOS version of the benchmark application

- Install framework
- Familiarize with how it works
- Rewrite the benchmark app in it
- Noting down results and documenting the experience

The remaining framework will be Evaluated on available of documentation, licensing, community, and flexibility.

Results

This paragraph will summarize the results of the research, more detailed results are included as an appendix.

From 30 existing solutions a total of four frameworks were selected to be evaluated:

- Titanium
- RhoMobile
- MoSync
- Worklight

Titanium. In the 3 days available for rewriting the benchmark app only the first two features were implemented successfully.

However there was a complication: Titanium did not support the DOM3 specifications. Not having been developed beyond DOM2 specifications the required methods to evaluate an XML document were missing, to be more precise, x-path could not be used. X-path is an industry standard used to locate data within XML structured documents. This is particularly troublesome because the resulting solution required hardcoded namespace resolving. Hardcoded namespace resolving means that one has to know the exact path to a node to access its specific piece of data. This is against best practice in handling XML documents.

Overall developing with Titanium proceeded really well, with exception for the DOM issue. The built results were as expected: cross-platform with the native look and feel.

Rhomobile. Up on installing it and reading the quick start guide it turns out that elements are not by definition *native* but HTML elements mimicking the native user interface style. The native feature advertised on their website referred to the fact that Rhomobile can encapsulate your web app in a web-view. This essentially devaluates the application type to Web-view based hybrid application as defined in chapter *Defining the context*, section *Alternative mobile application types*.

After some further research it turns out Rhomobile does have *some* support for native user interface elements, however only the very basic. When a the developer wants to use an element not in that list, he or she has to implement it in native code. [**?**] This is not how the cross-platform definition is envisioned by Lunatech.

MoSync. During the installation of the framework the installer prompted for the system location of *PhoneGap*, it would not install without. As it turns out, the developers of MoSync sync have the same definition for native as Rhomobile. The native feature advertised on their website referred to the fact that MoSync can encapsulate your web app in a web-view. The user interface is built using a PhoneGap implementation. Further research did not show a way for building actual native user interface elements using MoSync.

Worklight. Worklight also required PhoneGap. Even though Worklight classifies itself as producing actual native applications, it merely encapsulates a PhoneGap built user interface. The value Worklight brings is in their API which augments the PhoneGap application by exposing a JavaScript API to access some of the devices' native features.

Conclusion

The goal of this preliminary research was to "*determine whether or not an existing solution offers cross-platform mobile application development as defined by Lunatechs' criteria.*". Surveying and categorizing the available frameworks on the market resulted in a list of four possible solutions. By evaluating each solution it turned out that Titanium meets the criteria set by Lunatech. The remainder three frameworks did not adhere to the specific native user interface requirement. Although Rhomobile does offer *some* usage of native elements, it is not the main focus op the framework, as opposed to Titanium. Therefor the following conceptual model can be defined:

> “ Titanium offers cross-platform mobile application development as defined by Lunatech's criteria. ”

Since there is an existing solution which meets Lunatech's criteria it is not nessecary to research the implementation of a custom solution.

Main research

Introduction

The preliminary research confirmed that there is an existing solution as defined by Lunatech's criteria. In order to determine if this solution is suitable for adoption by Lunatech it is necessary to determine the its viability. Henceforth primary goal of the main research is to confirm the following hypothesis:

> “Titanium provides a viable solution to the cross-platform development of mobile applications”

The secondary goal of the main research is to analyze how Titanium works and why it provides the desired native *look-and-feel*. The results of this will add to the answer of the main research question.

Defining viable

To determine if a cross-platform mobile application development solution is considered viable for usage a set of criteria has been provided by Lunatech.

- *Extensibility*
  The possibility to extend new features to the framework.
- *Maturity*
  Maturity of framework.
- *Documentation*
  Coverage, accessibility and up-to-dateness of the documentation.

Extensibility is the possibility to extend Titanium with a new (custom) functionality. When the developer wants to implement a custom feature in the solution, the interfacing API should be easy to understand and use

The **maturity** is determined by checking for predefined *trustworthy elements* which are derived from the OpenSource Maturity Model[**?**]. The OSMM provides a formal methodology which utilizes a standardized analytical scheme for evaluating the maturity of an open source product. For this research the OSMM scheme has been augmented with elements fit to the specific mobile nature of the framework.

The **documentation** should be publicly accessible, cover all public methods in the most current version of the framework's API.

Method

In order to determine the viability of the solution it is necessary to perform a case study which originates from a realistic scenario. During the case study the solution will be analyzed on its inner workings.

The extensibility will be tested by adding a custom or missing feature to the case study application. The following points will be evaluated:

- Requirements for extending a custom feature
- Availability of third party extensions

The maturity of the framework will be assessed using an augmented version OpenSource Maturity Model assessment scheme. This scheme consists of a set predefined trustworthy elements:

- Product Documentation,
- Use of Established Mobile Standards,
- Licenses,
- Community activity,
- Number of Commits and Bug Reports
- Use of code management and versioning tools,
- Requirements Management,
- Availability and Use of a product roadmap.

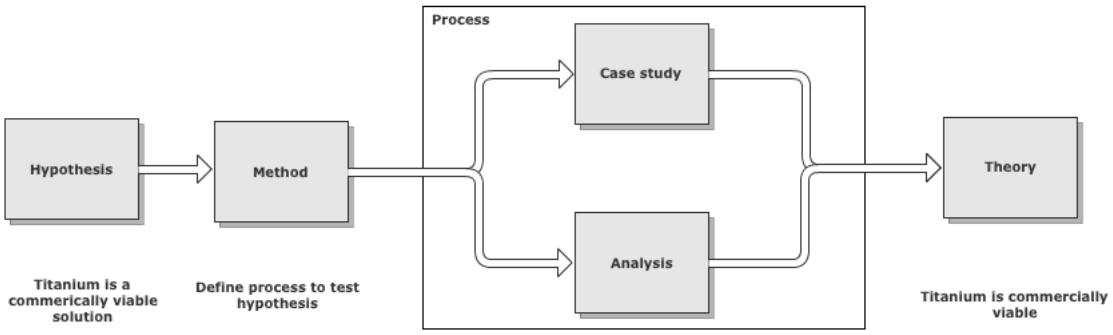
Quality of the documentation will be judged on:

- Completeness
  Whether or not any missing parts are noticed during the development of the case study.
- Up-to-dateness
  Determined by the support of the latest version.
- Availability
  Determined by the location of the documentation and or IDE integration.

criteria	**method**	**process**
Extensibility	Case study	Integrating a custom or missing feature in the case study application
Maturity	Analysis	OpenSource Maturity Model assessment
Documentation	Case study & analysis	By usage during the case study development and analyzing it

Table 1.3: Main research method

The main research method is divided in two processes, the case study and the analysis. Even though performed parallel, both serve different goals. The global structure is as following:



Main research process schematic

Case study

This goal of this paragraph is to introduce iStager and summarizes its development process. iStager is a cross-platform mobile application developed using Titanium. The summary will cover the application requirements and design, the implementation process and the results. The development of iStager provides a realistic case study. This is necessary in order to determine the commercial viability of the Titanium. Parallel to the case study an analysis of Titanium is performed which will be discussed in detail in the next chapter.

Stager

In 2011, live music venue WORM hired Lunatech to build *Stager*, a modern web-based resource planning and ticketing application to help manage live music events. Lunatech took the opportunity to use the relatively new Play framework to build a web application with an HTML5 and Java architecture. Stager has broad requirements ranging from high performance and security for the public ticket sales component to high usability for the internal resource planning component that will be used for hours a day by employees and being open to enhancements in the future for new customers. [**?**] In April 2012 Stager became Stager B.V. and is able to function independently from WORM. As main marketing concept Stager B.V. offers the use of Stager to live music venues. Although WORM is the main customer of Stager B.V. this might change in the future as Stager's market share expands.

WORM

WORM is an institute for avantgardistic recreation Rotterdam, consisting of an artistscollective, a podium with a bar and Parallel University (DIY workshops for film, music and media). Born under the stars of punk, Dada, Fluxus, Situationism and futurism WORM is grown into a headstrong organization that the 'Do-It-Yourself' mentality of their ancestors, combined with ultra-pragmatism, love of technique (s) and proper accounting. Worm outputs film, radio, concerts, courses, parties, publications, performances, web projects, installations, workshops and an accumulation of tactile media and internet.WORM focuses (cheerful yet serious) in avantgarde, resource scarcity and open source. [**?**] Due to a drain on subsidized funding resource from the government WORM is forced

halt its' current operations. It is possible that WORM closes its doors in august 2012, alternatively WORM might reorganize and continue in a s smaller form.

Introducing iStager

iStager is a mobile event publication platform. As described above Stager is a planning and ticketing application to help manage live music events. In addition to planning and ticketing Stager features an *atomfeed* to publish events. iStager makes use of this atom-feed to publish planned events on a mobile device. Because Stager Every customer of Stager would be granted their own version of iStager. This indicates that iStager needs to be have a generic user interface.

Stager application requirements

Functional requirements:

1. *Cross-platform*
   Application must be compatible to run on iOS 4.3 or greater *and* Android 2.2 or greater.
2. *Native look-and-feel*
   Application must make use the of native user interface components.
3. *i18n*
   Application must have language support for both Dutch and English.
4. *Generic style*
   Application must have a configurable stylesheet to generate different styles of user interface looks.

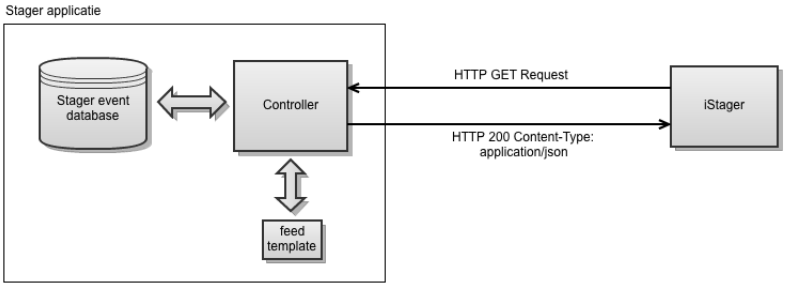
Below is a simplified list of the feature requirements of the iStager application.[3]

1. List of current and upcoming events
2. Detail-view of a event, shows detailed event information of a selected event
3. Add event to agenda
4. Start GPS-based navigation towards physical location of event
5. View media attached to a event
6. Display in a grid or list, categorize media types.
7. Share event details to social media

[3] The complete set of requirements is attached as a appendix II - Preliminary research

Architecture

The global architecture consists of a client-server model where the mobile application acts as the client, and the Stager application acts as the server.

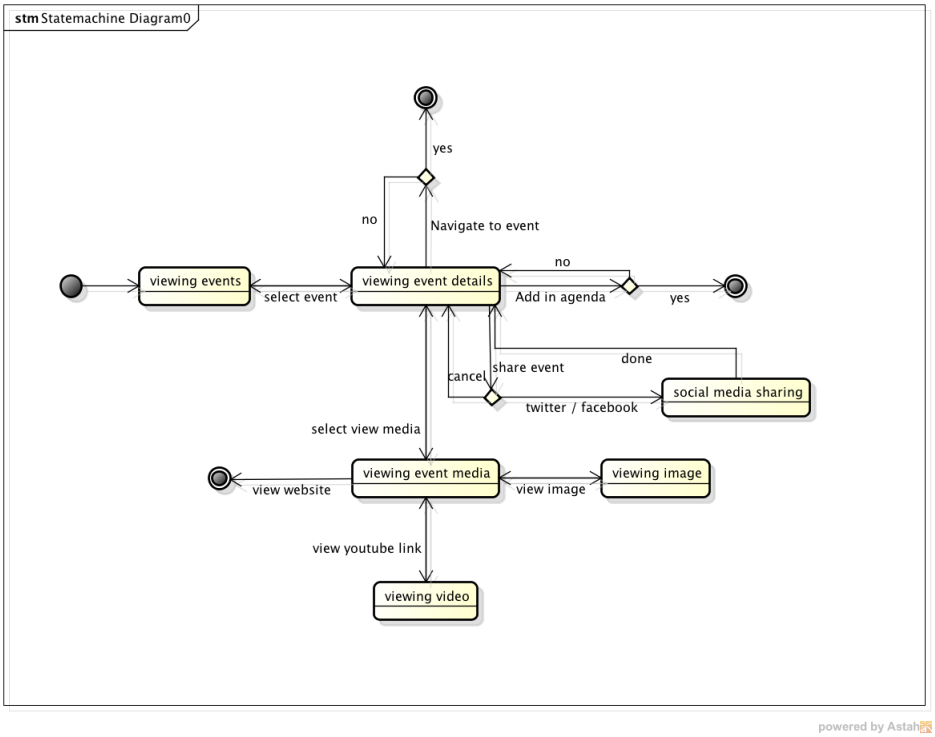


The client-server architecture of Stager and the iStager app

Design

The application designed in consultation with WORM. It had to be modern enough to stand out from other apps, yet it had to simple enough to support a generic stylesheet.

User interaction



Flowchart diagram of the iStager app

Figure 1.2: iStager application on running iOS 5.0

Feed

Stager is able to publish any planned live music events trough a feed which is publicly accessible. A feed is an simplified standardized view of content which may be frequently updated. In Stager the feed has an atom-based XML format.

Events

An event contains contains detailed information about an occurrence at a certain time, date and location. The detailed information consists of a title, subtitle, content, and the location of any attached media items.

Media

When an event is published it is possible to attach media items to the publication. These range from audio files and images to links to websites such as to Youtube content. Any attached media will be included in the feed as a hyperlink to its location, which is publicly accessible.

i18n

iStager has to support both English and Dutch. The internationalization of the app is to be automatically determined by the device locale settings.

Observations

DOM issue

An issue was the lack of DOM3 specification support. This was already determined during the development of the benchmark application in the preliminary research, however the chosen solution was a quick fix to solve the symptom of the issue and would not be acceptable for a real scenario. During the development of iStager was decided for a JSON implementation of the atom-feed. Since there are no atom standards for JSON it meant a whole new feed had to be written. It was not possible to convert the format. This is due to the manner in which the atom feed was designed: using a default xml template with groovy integrated code. The JSON feed implementation was done the same way to preserve consistency.

Internationalization

Internationalization in Titanium works by creating a folder called i18n the root of your project in which folders are to be created for each of the supported language in your application. These folders should be named according to the ISO 639-1 standard. For example, *nl* for Dutch. English *en* is the default language.[**?**] In each of these folders a `string.xml` file is to be created. This file provides the string resources. The string resource file closely mirrors the format of Android localization files, which have an XML-based format. The name attribute represents the 'key' for the string, and the text inside the XML node represents the value.[**?**]

Cross-platform NavigationController

Titanium does not provide a navigation controller. A navigation controller is used to navigate trough windows[4] of a mobile application it is common practice maintain a stack of windows in a controller. The controller is stack owner and has public methods for the pushing or popping of a window. Titanium does not offer a replacement. Developers who would like to stick with this best practice are advised to write their own navigation controller.

Titanium analysis

The primary goal of this paragraph is to determine the maturity of Titanium. As mentioned in *Main research*, section *Defining viable*, the maturity of a framework is determined based on the presence the following of trustworthy elements:

- Product Documentation
- Use of Established Mobile Standards
- Licenses
- Community activity
- Number of Commits and Bug Reports
- Use of code management and versioning tools
- Requirements Management
- Availability and Use of a product roadmap

Secondarily this paragraph is to analyze how Titanium works and why it provides the desired native *look-and-feel*. The results of this will add to the answer of the main research question

Inner workings

At runtime a mobile application developed with Titanium consists of three major components:

- The JavaScript source code
- A platform-specific implementation of the Titanium API

[4] Note that Titanium refers to what is normally considered a UIViewController on iOS as a Window

- A JavaScript interpreter

During runtime the JavaScript source code will be integrated in a native class where it is encoded as a string and compiled. The implementation of the Titanium API done in a platform specific native programming language, Java for Android and Objective-C for iOS. The JavaScript interpreter evaluates the JavaScript code at runtime.

Runtime

At runtime a JavaScript execution environment set up in the native environment, this is where the application source code is evaluated. Injected into JavaScript execution environment are so called *proxy* objects.

Proxy objects

A proxy object is a JavaScript object paired to an object in native code.[**?**] This means the object exists in both JavaScript and native code. Proxy objects gap the bridge between the native and the JavaScript environment. A global Titanium object in JavaScript exposes access to the proxy objects.

For example: In the JavaScript code, when a function is called on the global Titanium object to create a native UILabel a proxy object is created.

JavaScript-object

```
var label = Titanium.UI.createTabel({
   text: "Lorem impsum",
   top: 10,
   left: 10,
   width: 100,
   height: 20
});
```

In iOS the proxy button object:

Native-object

```
-(UILabel*)label
{
    if (label==nil)
    {
        label = [[UILabel alloc] initWithFrame:CGRectZero];
        label.backgroundColor = [UIColor clearColor];
        label.numberOfLines = 0;
        [self addSubview:label];
    }
    return label;
}
```

JavaScript

As mentioned above, Titanium uses JavaScript for cross-platform code compatibility. JavaScript is a logical choice because there are JavaScript Interpreters available for most platforms. This includes the targeted mobile platforms:

V8 is the default JavaScript Interpreter for Android but Rhino is also supported. V8 is has a better performance than Rhino because it is directly integrated to the NDK[5]. This means the code does not have to run trough the JVM[6]. The performance gain can exceed over 200% processing time when parsing a JSON object.[**?**] For iOS JavaScriptCore is the chosen interpreter.

These interpreters support the CommonJS specification.

CommonJS

JavaScript is a powerful object oriented language with some of the fastest dynamic language interpreters around. The official JavaScript specification defines APIs for some objects that are useful for building browser-based applications. However, the spec does not define a standard library that is useful for building a broader range of applications.

The CommonJS API will fill that gap by defining APIs that handle many common application needs, ultimately providing a standard library as rich as those of Python, Ruby and Java. The intention is that an application developer will be able to write an application using the CommonJS APIs and then run that application across different JavaScript interpreters and host environments. With CommonJS-compliant systems, you can use JavaScript to write:

Server-side JavaScript applications Command line tools Desktop GUI-based applications Hybrid applications (Titanium, Adobe AIR)

Modules

By default, JavaScript runs programs in a global scope and doesn't have any native name spacing language features. This means that, unless you are careful, your programs can descend into a mess of code spaghetti, full of conflicting variables and namespace pollution.

CommonJS modules are one of the best solutions to JavaScript dependency management.

CommonJS modules solve JavaScript scope issues by making sure each module is executed in its own namespace. Modules have to explicitly export variables they want to expose to other modules, and explicitly import other modules; in other words, there is no global namespace.

Modules prevent variables from clogging up the global namespace.

A sample module:

eventCell.js

```
1 function eventCell(Event, delegate) {
2   // cell code removed
3   return this.cell;
```

[5] Native Development Kit
[6] Java Virtual Machine

```
};
module.exports = eventCell;
```

Is used like:

eventCell.js

```
var messageCell = require('stager/tableview/messageCell');
tableview.appendRow(new messageCell(message, that));
```

Maturity

As mentioned the maturity of a framework is determined based on the presence of the predefined of trustworthy elements.

Product Documentation is found at the official website as well as a simplified version which included on the code completion. The documentation is updated either on the day of a new release or the day after. A simplified version of the documentation is integrated into Titanium-Studio, a feature named *'dynamic help'* automatically looks up the documentation belonging to the a selected method or object. Many objects in the documentation are supplemented with an code example.

Use of Established Mobile Standards

Licenses Appcelerator offers four plans from which the first one is free, the other three provide different levels of support and include the use of cloud services. The commercial plans' pricing is per application. Additionally Appcelerator offers services for app analytics or stand alone cloud services. A drawback is that prices are not publicly listed.

Community activity Titaniums community exists of mobile developers whom use Titanium to develop their mobile applications. Some of these have knowledge of native programming on the supported platforms. It happens frequently that they publish features they implement because it was missing from Titanium.

Number of Commits and Bug Reports Since the beta there have been 15,991 commits. On Titaniums' Jira page there have been 8784 issues reported of which 2601 are open. Issues: 398 created and 366 resolved

Use of code management and versioning tools Titaniums' code base is maintained in a public Github account.

Requirements Management Requirements and features are managed using Jira tickets.

Availability and Use of a product roadmap Not public: `http://developer.appcelerator.com/doc/mobile` "We apologize but at this time we're not publishing our roadmap."

Results

This paragraph summarizes the results of the main research.

Extensibility

It is possible to extend new functionality in Titanium. This is done at two levels:

- Project based
- SDK based

The Project based extending of features is done by placing the module in the Titaniums installation directory. SDK based means that the module is built inline with the Titanium's source code, from there the module augments Titanium's SDK. Extending Titanium's features at SDK level requires the entire framework to be rebuilt and compiled which can take up to 50 minutes. When a module is added on project base it only requires a restart from the framework, this process might take up to 2 minutes. Another downside is that a custom version of Titaniums SDK will have to be built and compiled for every new release. Therefore it is preferable to extend new functionality on project base.

Maturity

The following diagram is based on observations done during the analysis.

Thrustworthy element	Score
Product Documentation	+++
Use of Established Mobile Standards	++
Licenses	++
Community activity	++
Number of Commits and Bug Reports	+
Use of code management and versioning tool	+++
Requirements Management	+
Availability and Use of a product roadmap[7]	-+

Table 1.4: Maturity level

Documentation

During the case study the documentation of Titanium was used daily for a period of 8 weeks. The following observations were made: The documentation is located at a central web resource and in addition has been integrated in Titanium's IDE. All methods and functions which were needed during the development of the case study were found in the documentation. During the development of the case study a new version of Titanium was release, the documentation was update consequently. Each described function has a simple overview of which platforms it supports and in which SDK version it is included. As a result of these observations the following score's are met:

Criteria	Score
Completeness	+++
Up-to-dateness	+++
Availability	+++

Table 1.5: Maturity level

Conclusion and Recommendation

The preliminary research confirmed that Titanium offers cross-platform mobile application development as defined by Lunatech's criteria. The resulting main research tested the viability of Titanium by using it to develop iStager as a case study, and simultaneously by conducting an analysis of the framework.

Although not completely finished due to time restrains, the development of iStager proved that it is possible to build a cross-platform application which is truly native. In addition the realistic nature of the case studies' context proved Titanium can be used for development in a scenario Lunatech might encounter.

The analysis which was performed parallel to the case study resulted in a understanding of the inner working of Titanium, which explained why it provides truly native applications. Next to this the analysis also determined the maturity of the framework using an analytical scheme derived from the OpenSource Maturity Model.

Based on these statements the following conclusions can be drawn:

- *Titanium provides the desired extensibility*
  During the case study it was necessary to extend a feature in Titanium, this was done using a third-party module which augmented Titaniums' SDK with the required feature.
- *Titanium is a mature open source product*
  Titanium is a mature framework because the trustworthy elements derived from the OpenSource Maturity Model are present.
- *Titanium is a well documented open source product*
  The documentation is up-to-date with the latest version, augmented with code examples, and features an integration with the IDE.
- *Titanium provides truly native applications*
  Titanium provides cross-platform support while retaining the *native look-and-feel* trough the use of proxy objects. A high level API provides JavaScript access to the devices' native features. Each supported platform has its own implementation of the API.

All criteria of the main research are met, consequently its hypothesis may be accepted as a theory:

> “Titanium is a viable solution to the cross-platform development of mobile applications while retaining the native look-and-feel”

Which is a direct answer to the main research question: *"How to develop a cross-platform mobile application while retaining the native look-and-feel?"*

Recommendation

Titanium provides Lunatech with a viable solution which allows for the cross-platform development of mobile application while retaining the native look-and-feel. Given that both the scope of cross-platform and the definition of native look-and-feel remain unchanged, it is the recommendable to use Titanium as solution to develop mobile applications.

However in retrospective, it is the opinion of the author that the definition of native look-and-feel was defined to strict. It could have been defined in a broader, more abstract manner, as opposed to coupling it too performance and looks. For example, if the native look-and-feel were to be defined using:

- *Distribution*
  Ability to distribute the application trough its official distribution platform. (Appstore for iOS and Google Play for Android)
- *Performance*
  Have direct access to the memory and CPU of the device to allow lightning fast executing of code.

These settings could have changed the outcome of the preliminary research and therefore have resulted in an alternate conclusion.

appendix I - Preliminary research

Appcelerator Titanium

Appceleator Titanium is a commercially supported open source platform for developing cross-platform mobile applications. It was introduced by Appcelerator Inc in December 2008. Built upon the Eclipse IDE Titanium offers a Javascript API to native proxy classes which allow the developer to generate truly native cross-platform mobile applications.

Platform support

As of may 2012 Titanium supports iOS and Android. Next to building a native application for these platforms Titanium offers the option to generate a web application. Support for Research in Motion (BlackBerry) is in active (however closed from public) development. May first 2012 Appcelerator announced that it is extending its core value of cross-platform native application development beyond iOS and Android, on to RIM's BlackBerry devices.[**?**]

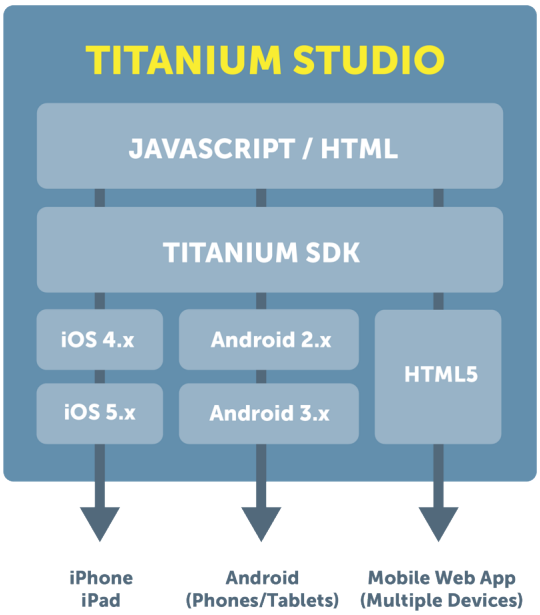
Techniques and tools

TitaniumStudio is an Eclipse based IDE with integration the proprietary mobile SDKs and simulators. For iOS this means Titanium requires Xcode with the iOS SDK to be installed, for Android the Android SDK and the Android AVD[8] are required.

Titanium applications are written in JavaScript but can be augmented with HTML & CSS. During runtime the JavaScript is evaluated and executed via so-called *proxy objects*.Proxy Objects are objects which are paired to a native object and can resemble native user interface elements.[9].

[8] AVD: Android Virtual Device (device simulator)

[9] Proxy objects are discussed in more detail in chapter x:Proxy objects



Titaniums bridge architecture[**?**]

Application type

As mentioned above, Titanium applications are written in JavaScript but can be augmented with HTML & CSS. The latter is in case a web application is required rather than a native application. This devices applications built with Titanium in two types:

- Web applications
- Native applications

Philosophy

The goal of Titanium is to provide a high level, cross-platform JavaScript runtime and API for mobile development.[**?**] Titanium aims to help developers leverage their JavaScript knowledge to build native mobile apps that run across multiple platforms.

Results

One of the issues which came up was that Titanium does not support DOM3 level specifications[**?**]. This effectively rules out the x-path evaluate function which has support for an xml namespace resolver.[**?**]

Finding this out took a while. I posted in Titaniums public Q&A to no effect. I updated and closed the question myself after I found the cause of problem in Titaniums' documentation. [**?**]

JavaScript-object

```
1 function nsResolver(prefix) {
2     var ns = {
```

Criteria	Score
Platform support	++
Native capabilities	+++
Programming language	JavaScript
IDE	Eclipse based studio
License type	Free but commercial available
Application type	Native

Table 1.6: Preliminary evaluation of Titanium

```
        'atom' : 'http://www.w3.org/2005/Atom',
        'gd'   : 'http://schemas.google.com/g/2005'
    };
    return ns[prefix] || null;
}

function getFeed() {
    var url = 'url of service returning a valid xml document';

    var request = Titanium.Network.createHTTPClient();
    request.open('GET', url);
    request.onload = function() {
        var doc;

        Ti.API.info('>>> got server reply!');
        if(request.readyState == request.DONE) {
            if(request.status == 200 && request.responseXML != null) {
                doc = request.responseXML;
                Ti.API.info('>>> success!');
            } else {
                Ti.API.info('>>> error!:(');
                return;
            }
        } else return;

        var entries = doc.evaluate('//atom:entry', doc, nsResolver, XPathResult.ANY_TYPE,
        Ti.API.info('>>> number of entries: ' + entries.length);
    };

    request.send();
}
```

The solution is to use the predating DOM2 level xpath evaluate function:

DOM2

```
var results = xml.evaluate("//*[local-name()='entry'
                and namespace-uri()='"+ xml.namespaceURI+"']");
```

Note that the function dates back to 2003 and prior. The namespace selection is hardcoded in the query, which is not very elegant.

Rhodes

Rhodes is an open source Ruby-based framework to build native applications for all major smartphone operating systems (iPhone, Android, RIM, Windows Mobile and Windows Phone 7). These are true native device applications (not mobile web applications) which work with synchronized local data and take advantage of device capabilities such as GPS, PIM contacts and calendar and the camera.

Platform support

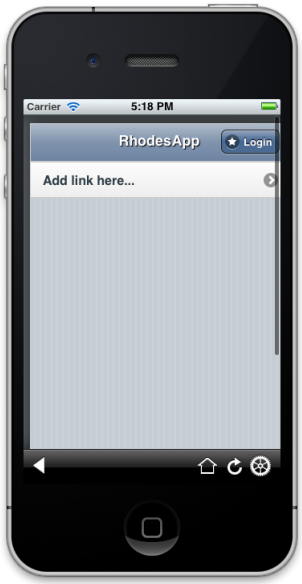
Rhodes supports iOS, Android, BlackBerry, Symbian, Windows Mobile

Techniques and tools

Rhodes is an comes with an Eclipse based studio. Rhodes applications are primarily written using Ruby for business code and user interface control. For the actual user interface Rhodes relies on web techniques such as HTML, JavaScript and CSS. Rhodes focusses on the *Model-View-Controller* pattern which is implemented using Ruby for the controllers and HTML for the views. The philosophy of the user interface is geared towards web technology using JavaScript libraries such as Sencha touch.

Application type

Even though Rhodes advertises producing fully native apps[**?**], they do not. In fact all they do is provide the user with a framework, access to some navigational user interface controls, but the rest of the user interface is html presented in a web-view. Although it is possible to augment the user interface with specific native components, it is not the focus of Rhodes.



Rhodes sample iOS application: not fully native

Philosophy

Rhodes focusses on a user interface built with JavaScript libraries such as jQuery Mobile and Sencha, it is their believe that this creates a better user experience as native.

Results

Criteria	Score
Platform support	+++
Native capabilities	+-
Programming language	Ruby & HTML, JavaScript and CSS
IDE	Eclipse based studio
License type	Free but commercial available
Application type	Framework hybrid applications

Table 1.7: Preliminary evaluation of Rhodes

Worklight

Worklight Studio is an eclipse based IDE for the cross-platform development of mobile applications. Worklight Studio was introduced in 200x by Worklight Inc. In early 2012 Worklight Inc. became an IBM company. Worklight Studio offers mobile development trough the use of web technologies such as HTML, and Javascript.

Worklight advertises the capability for developers to develop crossplatform HTML5, hybrid and native mobile applications.

Platform support

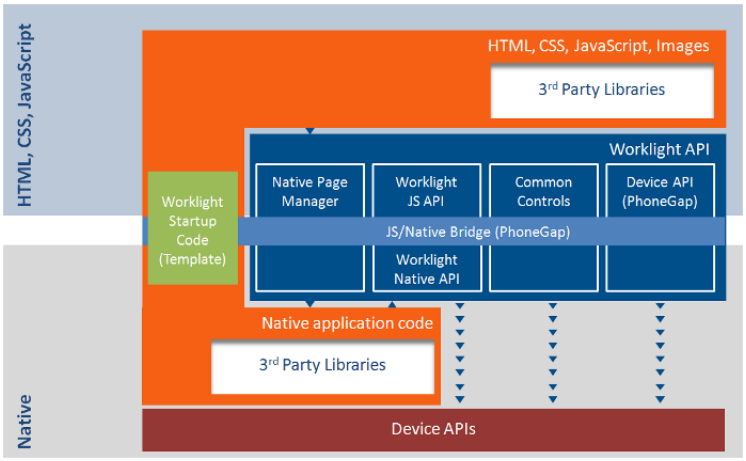
Worklight supports the following platforms: iOS, Android, BlackBerry and Windows Mobile 7.

Techniques and tools

As mentioned above, Worklight Studio is an eclipse based IDE. Allows the developer access to device APIs using native code or standard HTML5, CSS3 and JavaScript over a uniform PhoneGap bridge.

Application type

Applications developed with Worklight can be classied as *Webview-based hybrid applications*. As defined in the previous chapter this means the applications are based fitted inside web-view and build on a framework which provides an API to allow the application access to otherwise native API's. The latter is achieved trough Worklights SDK while the former is made possible by an implementation of PhoneGap. [**?**]



Worklights bridge architecture[**?**]

Philosophy

Worklight aims to grant developers access trough HTML5 to the capabilities that mobile devices provide.

Results

MoSync

The MoSync mobile SDK offers cross-platform development trough the use of web technologies or C/C++.

Criteria	Score
Platform support	++
Native capabilities	-
Programming language	HTML, JavaScript and CSS
IDE	Eclipse based studio with PhoneGap integration
License type	Commercial
Application type	Webview-based hybrid applications

Table 1.8: Preliminary evaluation of Worklight

Platform support

Android, Java ME, iOS, Symbian, Windows Mobile,

Techniques and tools

HTML, CSS and Javascript, Eclipse plugin, Visual Studio plugin

Application type

Webview-based hybrid applications

Philosophy

Results

Criteria	Score
Platform support	++
Native capabilities	-
Programming language	C++ & HTML, CSS and Javascript
IDE	Eclipse plugin, Visual Studio plugin
License type	Free
Application type	Webview-based hybrid applications

Table 1.9: Preliminary evaluation of MoSync

Comparisson

Conclusion

appendix II - iStager requirements

Functional requirements:

1. *Cross-platform*
   Application must be compatible to run on iOS 4.3 or greater *and* Android 2.2 or greater.
2. *Native look-and-feel*
   Application must make use the of native user interface components.
3. *i18n*
   Application must have language support for both Dutch and English.
4. *Generic style*
   Application must have a configurable stylesheet to generate different styles of user interface looks.

Feature requirements:

1. List of current and upcoming events
   (a) events are downloaded in JSON format from the Stager event atomfeed at `http://<feedurl>/web/feeds/events`
   (b) events are displayed in a row-based layout
   (c) events are linked to their corresponding event detailview
   (d) events in the list are sorted by date (ascending)
   (e) events in the list contain labels with event title, subtitle, date
2. Detailview of an event, shows detailed event information of an selected event:
   (a) event title, subtitle, date, times (doors open, start, end), location details (venue name, street, number, city), event content (html rendered text)
3. Add event to agenda
   (a) Prompts the user: "*Add event 'x' on date 'y' in agenda?*"
   (b) Adds a selected event to the mobile devices agenda.
   (c) Prompts the user of succes of add action.
4. Start gps-based navigation towards physical location of event

   (a) Prompts the user "*Navigate to y?*" (y in the format of: streetname, housenumber, cityname, postalcode)
   (b) Opens map application with address as argument.

5. View media attached to an event
   (a) Media defined as: URL's to event images, videos, websites.
   (b) When selected a media item will be displayed in new window

6. Display in a grid or list, categorize media types.
   (a) Each displayed media item is resembled by a tumbnail or icon.
   (b) When selected a media item opens to its content in:
   (c) images an included webview,
   (d) websites the device browser,
   (e) for videos the youtube app or the browser(depending on video type & location).

7. Share event details to social media
   (a) An event detail view will contain a 'share/deel' button.
   (b) Prompts the user for platform to share. (twitter/facebook/email)
   (c) Default value (editable): "*I am attending event x on date y in location x !*"

8. (Un)Register device to receive push notifications on new events of interest

9. Register the device to receive pushed notifications about upcoming events which might be of interest to the user. Based on Relation.interest model in Stager.

appendix III - Glossarium

iOS is a proprietary mobile operating system, developed by Apple Inc. It was originally released in 2007 for the iPhone and iPod Touch. iOS also became the main operating system of the iPad and Apple TV.

[**?**]

4th generation Apple iPhone running iOS 4.3

Android is a open source mobile operating system, developed by the Open Handset Alliance, led by Google and other companies.[**?**]

Samsung Galaxy S2 running Android 2.3

i18n is an a abbreviated numeronym of *internationalization*, where 18 stands for the number of letters between the first i and last n in internationalization.