A Comparative Analysis of Cross-platform Development Approaches for Mobile Applications

Spyros Xanthopoulos
Directorate of Technical Services and Computerization
Aristotle University of Thessaloniki
54124 Thessaloniki, Greece
0030 2310998828
xant@auth.gr

Stelios Xinogalos
Department of Applied Informatics
University of Macedonia
54006 Thessaloniki, Greece
0030 2310891820
stelios@uom.gr

ABSTRACT

Nowadays, native *mobile applications* (*mobile apps*) are targeted at specific mobile platforms. This phenomenon imposes severe constraints, such as the use of different development environments, technologies, and APIs (Application Programming Interfaces) for each mobile platform, leading inevitably to a waste of development time and effort, and an increased maintenance cost.

The paper focuses on the current trends in developing cross-platform mobile apps. Our analysis focuses primarily on three areas. In the first place, we clarify the cross-platform development landscape by exploring the most important cross-platform app types, which are *web, hybrid, interpreted and generated apps*. Secondly, key issues for each app type are presented and a comparative analysis is performed to highlight the advantages and disadvantages of each type. Thirdly, taking into account the current status in cross-platform mobile app development we identify a promising cross-platform app type and we investigate its effectiveness in practice. Finally, we draw some conclusions regarding cross-platform mobile app development approaches and make proposals for further research on the field.

Categories and Subject Descriptors

D.3.3 [**Language Constructs and Features**]: *Frameworks*. H.4.3 [**Information Systems Applications**]: Communications Applications – *Information browsers*. H.5.0 [**Information Interfaces and Presentation**]: General.

General Terms

Languages, Performance, Standardization.

Keywords

Smart mobile device, mobile application, cross-platform development, native app, web app, hybrid app, interpreted app, generated app, HTML5.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
BCI'13, September 19-21, 2013, Thessaloniki, Greece.
Copyright 2013 ACM 978-1-4503-1851-8/13/09 ...$15.00.

1. INTRODUCTION

Nowadays, application development for smart devices is an evolving field with great economic and scientific interest. According to Gartner [9] the total number of mobile app store downloads worldwide will increase to 81 billion in 2013, paid downloads will surpass 8 billion and free downloads 73 billion.

With the currently increasing number of mobile platforms, developing mobile applications became very difficult for companies as they need to develop the same applications for each target platform. The typical process of developing *native applications* is the appropriate way of deploying mobile apps but has one major disadvantage: it is not possible to reuse the source code for another platform; the very same app must be redeveloped from scratch.

Native apps are developed using an Integrated Development Environment (IDE) that provides the necessary development tools for building and debugging the applications. Native apps are more difficult to develop and require a high level of experience and technological know-how than other types of applications.

From the end user perspective, native apps provide the richest *user experience.* Source code is efficient, with fast performance, consistent look and feel and full access to the underlying platform hardware and data. The term *user experience* refers to the experience of the user on how to use the device. This is an important factor because the user must be able to operate the application immediately after installation and also *expects* from the app to function in a standard way. For example, special user settings (such as default language) or transition from portrait to landscape view should be taken into account during the application execution.

The ultimate goal of cross-platform mobile app development is to achieve native app performance and run on as many platforms as possible. This paper aims at investigating this issue. Specifically, the paper presents a review and comparative analysis of current cross-platform mobile app development approaches and a hands-on experience on utilizing one such approach for developing a typical mobile app.

Related work and papers are cited in the appropriate sections. In the paper by Heitkötter et al. (2012), the authors proposed a set of criteria to assess cross-platform development approaches [16]. A subset of those criteria is taken into consideration for evaluating the different cross-platform app types presented in our work. The authors in [16] also utilized those criteria for evaluating apps developed with frameworks that bridge the gap between web and

mobile systems like PhoneGap and Titanium Mobile and found PhoneGap viable if native look and feel can be neglected.

In general, papers in the field tend to focus on separate aspects and topics like operating systems of mobile devices [15], openness of architecture [2][7][25], comparisons between native versus web apps [8][24] or comparisons between platforms [13][23][33]. Our work is organized to foster better understanding of the overall cross-platform development process exploring the most important cross-platform app types, starting with an insight into the major mobile platforms and finally developing a cross-platform app (typical RSS reader) for the operating systems iOS and Android.

The paper is organized as follows. Section 2 presents some concepts related to mobile app development and the underlying challenges, and also a brief overview of mobile platform architecture. A review and a comparative analysis of cross-platform mobile app development approaches are presented in Sections 3 and 4 respectively. Section 5 reports on a case study of developing a typical RSS mobile app utilizing the approach that - based on the aforementioned comparative analysis – emerged as the most effective one for the time being. Finally, some conclusions and proposals for further research on the field are presented in Section 6.

2. MOBILE APPS AND PLATFORMS

Smart mobile devices are characterized by great computing prowess and can run complete operating systems with a number of pre-installed applications. Moreover, third-party applications can be downloaded and installed in smart mobile devices by users. These third-party applications are developed using sophisticated APIs and extend the device functionality through the direct access to the operating system components. A common term used for such applications is *mobile apps*. The term *app* is so common that it was included in 2010 in the list "Word Of The Year" by the American Dialect Society [1].

The mobile app deployment is made through special distribution *app stores*, with or without fees. In the general case each app store, such as the Apple App Store, the Google Play, the Windows Phone Store and BlackBerry App World, is owned by the company that manufactures the operating system. The most common operating systems used in smart mobile devices include the operating system Android (Google), iOS (Apple), Symbian (Nokia), BlackBerry OS (RIM), Samsung Bada, Microsoft Windows Phone, webOS (Hewlett-Packard) and Linux Maemo and MeeGo. According to Gartner [10] the operating systems that monopolize the mobile market are Android (by Google) with a share of 69.6% of installations and iOS (by Apple) with 20.9%. The shares for the rest operating systems are much smaller: 3.5% for RIM; 2.9% for Microsoft; 1.2% for Symbian; and 1.9% for the rest.

Each one of the aforementioned platforms requires a particular programming language, different development environments and programming models based on platform-specific APIs. For example, developing applications for Android requires Java, while developing applications for iOS requires Objective-C. It is apparent that, if a company decides to support both Android and iOS platform, there is a constraint to maintain two versions of a single product: one version implemented with Java for Android and a second version implemented using Objective-C for iOS. Platform-specific development and variations in the underlying operating system cause a phenomenon known as *fragmentation* that increases development time and maintenance costs [11].

As the concept of *write once, run anywhere* can't be applied when building native applications, the best alternative option for companies is the *cross-platform mobile development*. Cross-platform development simplifies the maintenance and deployment processes, and saves development time and effort.

The concept of *platform* in its general form includes a set of hardware or software components that make it possible to develop additional services and extensions [7]. In the case of smart mobile devices the mobile platform consists of the operating system and the necessary hardware components. The operating system is responsible both for managing the device's hardware and providing a framework for implementing native apps. The operating system is distributed with various built-in applications, such as a web browser. The software development kits (SDK) provide the necessary tools and resources for the development, installation, and test of the applications.

In the general case the mobile platform technologies can be viewed as a set of hierarchical layers. Higher levels contain sophisticated services; lower layers contain the necessary technologies on which the apps rely.

Android and iOS platform contain four well defined layers [6] (Table 1).

Table 1. Android and iOS layer hierarchy

Layer	Android	iOS
1	Applications	Cocoa Touch
2	Application framework	Media layer
3	Libraries	Core services
4	Linux Kernel	OS layer

At the highest level the mobile platform acts as an intermediate layer between the user applications and the underlying hardware components. This layer includes all pre-installed and third-party applications that extend the functionality of the device.

The second layer contains the audio and video technologies that create multimedia experience.

The third layer contains all the fundamental system services that all applications use directly or indirectly, such as security and data storage.

Finally, the last layer contains the necessary technologies and operating system interfaces that other layers need, such as files, drivers and memory management.

3. REVIEW OF CROSS-PLATFORM DEVELOPMENT APPROACHES

Nowadays, there is a strong growth in new software development tools and applications for smart mobile devices. The unexpected growth in the mobile market motivated the implementation of cross-platform software development environments that could make the development easier and more efficient. The main categories of applications produced by these software environments are *web*, *hybrid*, *interpreted* and *generated* apps.

None of the approaches is neither prevalent nor the best solution to the problem of developing cross-platform mobile applications. There are many other development environments which can be classified into intermediate categories. For example, the commercial software IBM Worklight subdivides the category of *hybrid* apps into two subcategories: the *hybrid web* applications and *mixed hybrid* applications [20]. According to IBM, the source code of hybrid web applications consists exclusively of HTML5, and is executed through a web browser, while the mixed hybrid applications can execute native code calls through a native API.

3.1 Web apps

Web apps are browser-based applications in which the software is downloaded from the web. Web apps are based on widespread Internet technologies such as HTML and JavaScript. The main disadvantage of web apps is the limited access to the underlying device's hardware and data. Another problem is the extra time needed to render the web pages and the extra cost needed to download the web page from Internet.

Web apps do not require installation and subsequent upgrades. On the other hand, since they can't be installed physically on a device there are situations such as when the device is in Airplane mode where web apps are inaccessible for the end user.

Today many software libraries promise to develop web applications that simulate the functionality of native applications, such as JQuery Mobile, Sencha Touch, JQTouch, WebApp.net, Xui and many others. It is worth mentioning that the development and adoption of the HTML5 [29] standard promises access to the device's hardware and software components through a number of APIs. HTML5 is evolving to a mobile friendly version, focuses on standardization and a range of new mechanisms is available on any HTML5-compliant web browser ([28], [32]).

3.1.1 HTML5

HTML5 is the fifth major version of HTML, started in 2004. The contents of HTML5 are still being discussed on the HTML Working Group and differences between HTML4 and HTML5 are documented by W3C in the technical report "HTML5 differences from HTML4" ([18], [29], [30]).

HTML5 focuses on standardization and includes more concise statements and sections (such as <section>, <header>, <nav>, among others), while a number of older elements and attributes have been changed or removed. Also, advanced functionality has been implemented including form field validations and many new data types covering a wide range of requirements (e.g. "tel", "search", "url", "email" etc) [32].

HTML5 is not a simple set of formatting tags. A number of APIs have been implemented in a way to avoid energy consumption and new mechanisms have been implemented, such as data storing and exchanging, video presentation and complex content formatting. The classic web applications lacked persistent storage functionality for large data and storage standardization, so partial solutions have been introduced by companies such as Microsoft, Adobe and Google. Microsoft added the *userData* hierarchical structure in Internet Explorer, having a capacity of 64KB per page for persisting information across sessions as an alternative to cookies. Adobe Flash Player 6 introduced the *Local Shared Object* structure, while Google initially introduced Google Gears software [23] and finally migrated to Web Storage providing support for HTML5 client-side storage through libraries such as GWT SDK.

In the HTML5 standard, a number of APIs have been created to provide standardization such as Web Storage, Indexed Database API, File API, Web SQL Database and Offline Web and Geolocation API [31].

3.2 Hybrid apps

Hybrid apps try to combine the advantages of web and native apps. Hybrid apps are primarily built using HTML5 and JavaScript, and a detailed knowledge of the target platform is not required.

Hybrid apps embed HTML5 apps inside a thin native container (UIWebView in iOS and WebView in Android). Like web apps, the source code is still executed by a browser that is part of the final application and can be packaged with the application unlike web apps where the source code is downloaded from the web. Hybrid apps are installed on the device and access to the underline device hardware and data is feasible through specialized APIs. An example of the most popular container for creating hybrid mobile apps is PhoneGap [26] [27].

The code implementation of the hybrid apps can be done using various technologies and development platforms but in order to achieve a native look and feel it is necessary to use specific development libraries, such as JQuery [22][33].

3.3 Interpreted apps

In interpreted apps native code is automatically generated to implement the user interface. The end users interact with platform-specific native user interface components, while the application logic is implemented independently using several technologies and languages, such as Java, Ruby, XML etc.

The main advantage of this approach is efficiency because of the native user interfaces, but the downside is the complete dependence on the software development environment. More specifically, new platform-specific features (e.g. new user interface features of a new android version) can be available to apps only when and if supported by the development environment.

An example of the most popular software development environments for creating interpreted apps (as well as hybrid apps) is Appcelerator Titanium Mobile [3].

3.4 Generated apps

Generated apps are compiled just like a native app and a platform-specific version of the application is created for each target platform. A popular example of software development environment for creating generated apps is Applause [12].

Generated apps achieve high overall performance because of the generated native code. In theory it is also possible to exploit the produced native code, in order to meet specific needs (e.g. in the case that a suggestion has been made from the market store to correct a deprecated call), or even to adopt and extend the produced native source code to build native apps. However, in practice utilization of the generated native code is difficult because of its automated structure.

Also, we must mention that we were not able to track down a free productive development environment while existing ones like Applause [12] or iPhonical [21] are targeted to model-driven solutions [16].

Model-driven software development (MDSD) focuses on describing the problem domain in a *model* [17]. A modeling language is used for representing the functionality and behavior of the application. The subsequent source code generation process is based on that model to produce the final target platform source-code. When it comes to mobile cross-platform application development, developers model their app functionality and specifications to a higher abstraction level and finally truly native apps are *generated* for different mobile platforms like iOS and Android.

The open-source development environment Applause uses a domain-specific language (based on XText framework) as input, explicitly designed for data-driven mobile applications to produce human-readable source code in Objective-C, Java, C# or Python. In data-driven apps the main functionality involves create, read, update and delete operations (CRUD) and other data-centric operations like tabular views and data entries [17].

Applause is licensed under the EPL license, tightly integrates with Eclipse with IDE support and produces mobile applications for iOS, Android and Windows Mobile.

4. COMPARATIVE ANALYSIS OF CROSS-PLATFORM DEVELOPMENT APPROACHES

In this section a comparative analysis of the aforementioned cross-platform mobile app development approaches is presented based on a set of characteristics converging with the set of criteria proposed in [16]. In this paper the authors proposed the specific set of criteria for assessing cross-platform development approaches and utilized them for comparing frameworks that bridge the gap between web and mobile systems, like PhoneGap and Titanium Mobile. The criteria we use for the comparative analysis of cross-platform development approaches are the following:

- *Market place deployment (distribution).* Evaluates whether and how easy it is to deploy apps to the app stores of mobile platforms, like Google Play or Apple's iTunes.
- *Widespread technologies*: Evaluates whether apps can be created using widespread technologies, such as JavaScript.
- *Hardware and data access*: Evaluates whether apps have no access, limited or full access to the underlying device hardware and data.
- *User interface and look and feel*: Evaluates whether apps inherently support native user interface components or native user interface and look and feel is simulated through libraries, such as JQuery Mobile.
- *User-perceived performance:* Evaluates whether apps have low, medium or high performance as perceived by the end users (like loading time and execution speed) compared to a native app. This criterion is an empirical rough estimate based on our practical experience and information published on the web. The overall performance could be affected by plenty of factors; a detailed research is required to identify and evaluate factors that impact user-perceived performance in practice [19].

There are many other characteristics that are as important as these stated here that could be used to enrich this comparative analysis [16]. The ideas presented in this paper can be the basis for further research as proposed in the final conclusions.

The results of the comparative analysis are presented in the next paragraphs and summarized in Table 2.

Table 2. Comparative analysis of cross-platform development approaches

	Web	**Hybrid**	**Interpreted**	**Generated**
Marketplace deployment	No	Yes, but not guaranteed*	Yes**	Yes**
Widespread technologies	Yes	Yes	Yes	No
Hardware and data access	Limited	Limited	Limited	Full access
User interface and look & feel	Simulated	Simulated	Native	Native
User-perceived performance	Low	Medium	Medium	High

** E.g. simple web clippings content aggregators or a collection of links, may be rejected by Apple.*

*** Apps can be distributed without difficulty but the experience that the app provides must comply with the app store development guidelines.*

Web apps are browser-based applications that can't be installed physically. Extra time needed to download the source code and resources form the web leads inevitably to low performance compared to native apps while web apps may become unavailable after a network failure. Libraries like jQuery Mobile promise to develop applications with a native look and feel and the adoption of HTML5 will provide standardization through a number of APIs.

Hybrid apps simulate the look and feel of a native application. The main advantage of hybrid apps is the ability to run the source code in a wide range of platforms and the fact that the development of the source code is made using widely used web development technologies (especially using JavaScript). A detailed knowledge of the target platform is not required. Hybrid apps have a medium performance as perceived by the end users compared to native apps, given that the user interface is still web based with no use of optimized native components, and on the other hand specialized APIs are intersecting to interpret business logic and hardware access. Hardware and data access to the underlying platform is limited and in general apps can be distributed via app stores although apps that are primarily web apps could be declined [16]. For example, Apple has severe development guidelines [5], and apps that are simply web clippings, content aggregators or a collection of links, may be rejected. While testing hybrid apps implemented using PhoneGap and JQuery Mobile, Apple rejected apps explaining that *"We found that the experience your app provides is not sufficiently different from browsing a content aggregator web site, as required by the App Store Review Guidelines"*.

Interpreted apps have native user interfaces and the application logic is implemented independently. The overall performance as perceived by the end users is medium compared to native apps, considering the rapid native user interface on the one hand but the extra time needed to interpret the application logic and the

presence of specialized APIs to access the underlying hardware components on the other hand. Hardware and data access is limited. In terms of reliability, specialized APIs can be used to store apps state and data (e.g. SQLite API) and in general apps can be distributed without difficulty (given that the experience the app provides must comply with the app store development guidelines).

Generated apps achieve high overall performance as native source code is generated. However, the generated code has automated structure. Software development environments in this category support major platforms such as iOS, Android and Windows Phone and can generate source code (Objective-C, Java, C# or Python). However, non-commercial versions do not offer a productive development environment, and are mainly targeted to simple data-driven apps (focused on data storage and manipulation).

In conclusion, according to our analysis, there is no best solution regarding the cross-platform development approach that should be chosen for a mobile application in general. Selecting a cross-platform development approach is a function of several factors, which are actually related to the aforementioned criteria used for the comparative analysis. For example, when designing a mobile app for which app store distribution is not a requirement, hardware access is not needed and also there are short project deadlines with no intension to invest extra time and effort in a specific development environment, the web app approach might be considered as the best option. Furthermore, web apps are a good starting point for cross-platform development and they can be ported on other approaches. Hydrid and Interpreted apps are the best option if distribution through app store channels is required and also apps must be more complicated and generic in nature, apart from data-driven applications. In Google Play app store there are more than 20 category types used to organize apps such as comics, communications, finance, health & fitness etc [14]. If native, and not simulated, user interface and look and feel is considered important then an interpreted app should be chosen.

The implementation of generated apps seems to be a promising cross-platform mobile app development approach but existing free development environments (like Applause [12] or iPhonical [21]) support only model-driven apps and are not production ready [16]. Evaluating commercial development environments that fall into this category and provide a productive development environment could be the subject for further research.

The approach of interpreted apps will be used for creating a typical application as a case study, as we don't want to put extra effort in simulating native user interface and also our primary platform targets are Android and iOS. An example of the most popular software development environments for creating interpreted apps is Titanium [2] from Appcelerator, which will be used for creating a typical RSS reader application as a case study. Titanium provides a highly productive development environment that supports iOS, Android and BlackBerry and apps are built using JavaScript [4].

5. CASE STUDY: DEVELOPMENT OF A TYPICAL MOBILE APP

The case study application that will be created is a simple RSS feeds client, targeted at Android and iOS platforms. The RSS client will query for the latest Apple's press info news (http://www.apple.com/pr/feeds/pr.rss) and display the data on the device screen. In the code fragment of Figure 1 we create the main window of the application. Setting the property exitOnClose:true ensures that the app will exit when the "Back" button is pressed. The property url:"ui/rssFeedList.js" loads the rssFeedList.js JavaScript file into the window body.

```
var windowMain = Titanium.UI.createWindow({
    title : 'Rss feeds reader',
    backgroundColor : '#00A9EC',
    font : {fontWeight: 'bold',fontSize:'15dp'},
    barColor : '#00A9EC',
    url : "/ui/rssFeedList.js",
    exitOnClose : true
});
```

Figure 1. The main window of the application.

In Figure 2 we create the necessary HTTPClient object that opens a connection to the RSS server and get the RSS data.

```
var tbdata=[];
var hTTPClient = Ti.Network.createHTTPClient();
hTTPClient.open('GET',
  "http://www.apple.com/pr/feeds/pr.rss", false);
hTTPClient.setRequestHeader("Content-Type",
  "text/xml;   charset=\"utf-8\"") ;
```

Figure 2. Connecting to the RSS feeds server.

Next, the code loops through all the items (Figure 3).

```
hTTPClient.onload = function(e) {
  var xml = this.responseXML;
  var doc = Titanium.XML.parseString(
      this.responseText);
  var items = doc.documentElement.
      getElementsByTagName("item");
  var data=[];
  for (var i=0;i<items.length;i++){
    // Loop through the RSS data
  }
}
```

Figure 3. Fetching and loop through the RSS data.

Following, the code extracts the title, description, link and publication date of each item (RSS feed) and creates a table row that will be displayed on the device screen (Figure 4).

```
for (var i=0;i<items.length;i++){
  var item = items.item(i);
  var title = item.getElementsByTagName('title').
      item(0).text;
  var link = item.getElementsByTagName('link').
      item(0).text;
  var pubDate = item.getElementsByTagName('pubDate').
      item(0).text;
  var description = item.getElementsByTagName(
      'description').item(0).text;
  var tablerow = Ti.UI.createTableViewRow({
      height : '75dp', title: title,
      description: description, link: link,
      backgroundColor : '#FFFFFF'
  // display data
});
```

Figure 4. Extracting the title, description, link and publication date from the RSS data.

After that, we create the necessary labels (Figure 5).

Finally, we add the table view to the main window of the application (Figure 6) and request the RSS data from the server through a hTTPClient.send() call.

```
var titleview = Ti.UI.createLabel({
    text : title, color : '#000',
    height : '55dp',
    font : {fontSize : '15dp'},
    bottom : '2dp',  left : '3dp',
    right : '1dp',
    backgroundColor : '#FFFFFF'
});
var dateview = Ti.UI.createLabel({
    text : pubDate,
    textAlign : 'center',
    color : '#00A9EC',
    font : {fontSize:'12dp',fontWeight:'bold'},
    height : 'auto',
    left : '3dp',
    top : '3dp',
    backgroundColor : '#FFFFFF'
});
tablerow.add(dateview);
tablerow.add(titleview);
tbdata.push(tablerow);
```

Figure 5. Inserting the RSS data into visual elements.

```
var tableview = Titanium.UI.createTableView({
    top : '35dp',
    width : '100%',
    separatorColor : '#5C9FB4',
    backgroundColor : 'transparent'
});
win.tableview = tableview;
win.add(tableview);
win.tableview.setData(tbdata,{});
//…
hTTPClient.send();
```

Figure 6. Adding the table view to the main window of the application.

In Figures 7 and 8 we present the RSS feeds reader application on Android and iOS Simulator.

The total source code is less than 150 lines and the client has been tested on Android and iOS device, so we can confirm that the same source code actually works on Android and iOS as well.

Every code line is pure JavaScript without the need of coding anything using a native language.

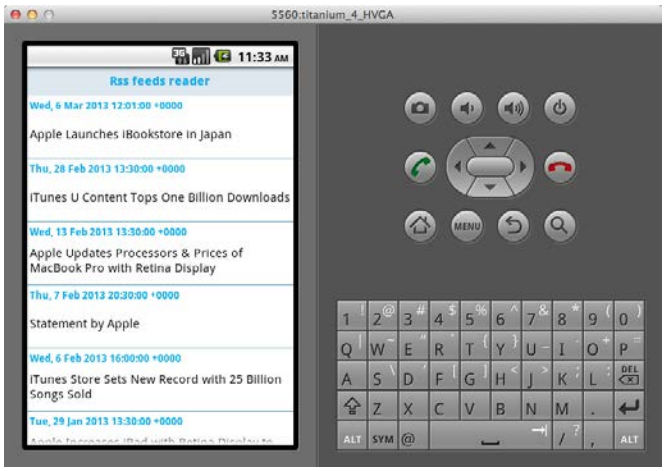

Figure 7. RSS feeds reader run on Android Simulator.

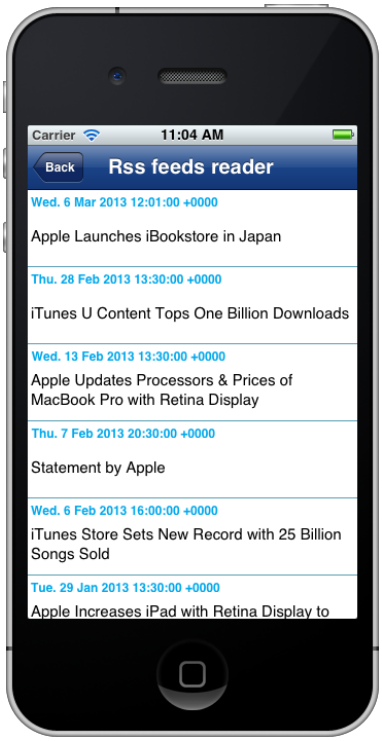

Figure 8. RSS feeds reader run on iOS Simulator.

6. CONCLUSIONS

Mobile app development is an evolving field that has attracted great interest. The most common type of mobile apps was, until recently, native mobile apps. Native mobile apps are developed for a specific platform and offer without doubt the best user experience. In native mobile apps source code is efficient, with fast performance, consistent look and feel and full access to the underlying platform hardware and data. However, the increasing number of mobile platforms has constituted the development of native mobile apps inefficient both in terms of development time and development/maintenance cost. Developing a native mobile app for various platforms requires a distinct implementation for each platform, a phenomenon known as fragmentation.

The ultimate goal is devising cross-platform mobile app development approaches and corresponding development environments/frameworks. Writing a mobile app and running it on various platforms, providing a native look and feel is a challenging task. The aim of this paper was to investigate the cross-platform approaches that have been proposed and their contribution towards achieving the aforementioned goal. Our review showed that the current trends in developing cross-platform mobile applications focus on four primary app types, which are *web*, *hybrid*, *interpreted* and *generated apps*.

The comparative analysis of these app types shows that the most promising approach is that of generated apps. However, although generated apps seem to be the best solution after native apps, we were not able to track down a non-commercial development environment supporting this approach for building generic apps. Open source initiatives like Applause or iPhonical are targeted to model-driven applications. In general, the development process is based on a domain-specific language (like XText) while the final human-readable source code in Objective-C, Java, C# or Python is produced through a set of code-generators. When it comes to

commercial solutions, although there are various alternatives that seem to fall into this category, further research is needed to evaluate them in detail.

On the other hand, the implementation of *hybrid* and *interpreted apps* seems to be a promising cross-platform development solution for building generic apps, supported by production ready non-commercial tools.

If default native look and feel is necessary then the implementation of interpreted apps is the best option. A popular development environment for building interpreted apps is Titanium. Hybrid apps simulate native user interface components through specific libraries (such as JQuery), so extra effort to invest in such libraries is needed. PhoneGap is one of the dominant development environments for building hybrid apps. Frameworks like Titanium and PhoneGap are using widely used web development technologies (especially JavaScript), do not require a detailed knowledge of the target platform and are certainly worth considering for building cross-platform applications. Titanium supports the operating systems iOS, Android and Blackberry (is in beta), while supported operating systems by PhoneGap include iOS, Android, Blackberry, Windows Mobile and more.

The overall performance as perceived by the end users for both types is medium; hybrid and interpreted apps can be generic in nature although limitations like restricted hardware access still exist. In general, hybrid and interpreted apps can be distributed without difficulty given that the experience the app provides must comply with the app store development guidelines.

Our case study showed that Titanium indeed supports what interpreted apps are promising. Specifically, we managed to create a simple rss reader application using JavaScript without any knowledge of the target Android and iOS platform. We found that Titanium classes are easy to understand and grasp with little effort, and we verified in practice the characteristics of the created interpreted app, such as the native user interface and the satisfactory performance. However, the downside is the complete dependence of the application to the software development environment. For example, a new platform-specific feature must be supported by the development environment to become available to the application.

Regarding mobile app development environments in general, it is important to note that today cross-platform software development environments seem to focus on the coverage of the characteristics of each platform and not on a general infrastructure that provides complete independence from the existing platform containing a superset of all available features of operating systems. For example, the operating system iOS provides additional infrastructure build menus that are not supported in the operating system Android. Thus, the use of these features requires additional programming for the operating system Android.

In conclusion, according to our analysis, the implementation of interpreted apps is a promising cross-platform development solution, while the evolution of other app types and standards like HTML5 and generated apps will probably play a major role in future.

The work presented in this paper can be the basis for further research in the following axes:

- The proposed set of criteria of Table 2 for assessing cross-platform development approaches could be enhanced by including factors that impact performance, reliability etc.
- Formal assessment of the various features of the four app types summarized in Table 2 by conducting tests in several real life conditions. For example, evaluation of start-up times, user interface responsiveness, execution times, session and data protection against network failures or other interrupts like incoming calls could be made.
- Categorization of popular free and commercial cross-platform software development environments according to the four mobile app types (web, hybrid, interpreted, generated apps) and evaluation.
- Cross-platform development environments hide target-platform implementation details behind APIs. The question is: do they really make proper use of resource management? Indicatively, specific tests could be designed in order to expose memory leaks and improper use of memory, common reasons of app crashes.

7. REFERENCES

[1] American Dialect Society. All of the Words of the Year, 1990 to Present. Available online at: http://www.americandialect.org/woty/all-of-the-words-of-the-year-1990-to-present#2011, (last access on February 26, 2013).

[2] Anvaari, M. and Slinger, J. 2010. Evaluating architectural openness in mobile software platforms, In *Proceedings of the Fourth European Conference on Software Architecture: Companion Volume*, ACM. 85-92.

[3] Appcelerator Inc. Build native apps with Appcelerator Titanium. Available online at: http://www.appcelerator.com/, (last access on March 25, 2013).

[4] Appcelerator Inc. Quick Start. Available online at: http://docs.appcelerator.com/titanium/latest/#!/guide/Quick_Start, (last access on March 25, 2013).

[5] Apple Inc. App Review. Available online at: https://developer.apple.com/appstore/guidelines.html, (last access on March 25, 2013).

[6] Apple Inc. iOS Technology Overview, (2013). Available online at: http://developer.apple.com/library/ios/#documentation/miscellaneous/conceptual/iphoneostechoverview/Introduction/Introduction.html, (last access on March 25, 2013).

[7] Boudreau K. 2010. Open platform strategies and innovation: Granting access vs. devolving control. *Management Science*, Vol. 56, no. 10, 1849-1872. DOI:10.1287/mnsc.1100.1215.

[8] Charland, A. and Leroux, B. 2011. Mobile application development: web vs. native. *Communications of the ACM*, *54*(5), 49-53.

[9] Gartner. Gartner Says Free Apps Will Account for Nearly 90 Percent of Total Mobile App Store Downloads in 2012, (2012). Available online at: http://www.gartner.com/newsroom/id/2153215, (last access on February 26, 2013).

[10] Gartner. Gartner says Worldwide Mobile Phone Sales Declined 1.7 Percent in 2012, (2013). Available online at: http://www.gartner.com/newsroom/id/2335616, (last access on March 27, 2013).

[11] Gavalas, D. and Economou, D. 2011. Development Platforms for Mobile Applications: Status and Trends. *IEEE Software*, Vol. 28, no. 1, 77-86.

[12] GitHub. Applause. Available online at: https://github.com/applause/applause, (last access on March 25, 2013).

[13] Grønli, T. M., Hansen, J. and Ghinea, G. 2010. Android vs Windows Mobile vs Java ME: a comparative study of mobile development environments. In *Proceedings of the 3rd International Conference on PErvasive Technologies Related to Assistive Environments*, p. 45. ACM.

[14] Google Play. Category Types. Available online at: http://phonegap.com/, (last access on June 10, 2013).

[15] Hall, S. P. and Anderson, E. 2009. Operating systems for mobile computing. *Journal of Computing Sciences in Colleges*, *25*(2), 64-71.

[16] Heitkötter, H., Hanschke, S. and Majchrzak, T. A. 2012. Comparing cross-platform development approaches for mobile applications. In *Proc. 8th International Conference on Web Information Systems and Technologies (WEBIST)*, 299-311.

[17] Heitkötter, H., Majchrzak, T. A. and Kuchen, H. 2013. Cross-platform model-driven development of mobile applications with md 2. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, 526-533.

[18] Hickson I. HTML Living Standard. Available online at: http://www.whatwg.org/specs/web-apps/current-work/multipage/, (last access on March 29, 2013).

[19] Huang, J., Xu, Q., Tiwana, B., Mao, Z. M., Zhang, M. and Bahl, P. 2010. Anatomizing application performance differences on smartphones. In *Proceedings of the 8th InternationalConference on Mobile systems, applications, and services*, ACM, 165-178.

[20] IBM. IBM Worklight Application Types. Available online at: http://www-01.ibm.com/software/mobile-solutions/worklight/features/, (last access on March 23, 2013).

[21] iPhonical. Model Driven Software Development for the iPhone (and Android). Available online at: http://code.google.com/p/iphonical/, (last access on June 2, 2013).

[22] jQuery Foundation. jQuery write less, do more. Available online at: http://jquery.com/, (last access on March 23, 2013).

[23] Mansfield-Devine, S. 2010. Divide and conquer: the threats posed by hybrid apps and HTML 5. *Network Security*, 2010(3), 4-6.

[24] Melamed, T. and Clayton, B. 2010. A comparative evaluation of HTML5 as a pervasive media platform. In *Mobile Computing, Applications, and Service*, Springer Berlin Heidelberg, 307-325.

[25] Müller, R. M., Kijl, B. and Martens, J. K. 2011. A comparison of inter-organizational business models of mobile App Stores: there is more than open vs. closed. *Journal of theoretical and applied electronic commerce research*, *6*(2), 63-76.

[26] Palmieri, M., Singh, I. and Cicchetti, A. 2012. Comparison of cross-platform mobile development tools. In *Proc. 16th International Conference on Intelligence in Next Generation Networks (ICIN*,, 179-186. DOI: 10.1109/ICIN.2012.6376023

[27] PhoneGap. Easily create apps using the web technologies you know and love: HTML, CSS, and JavaScript. Available online at: http://phonegap.com/, (last access on March 25, 2013).

[28] W3C. Device APIs Working Group, (2013). Available online at: http://www.w3.org/2009/dap, (last access on March 25, 2013).

[29] W3C. HTML 5.1 Nightly A vocabulary and associated APIs for HTML and XHTML, (2013). Available online at: http://www.w3.org/html/wg/drafts/html/master/, (last access on March 23, 2013).

[30] W3C. HTML5 differences from HTML4. Available online at: http://www.w3.org/TR/html5-diff/, (last access on March 29, 2013).

[31] W3C. Web Storage, (2013). Available online at: http://dev.w3.org/html5/webstorage/#storage-0, (last access on March 23, 2013).

[32] Xinogalos S., Psannis E. K. and Sifaleras A. 2012. Recent advances delivered by HTML 5 in mobile cloud computing applications: a survey, In *Proceedings of the Fifth Balkan Conference in Informatics* (BCI '12). ACM, New York, NY, USA, 199-204. DOI=10.1145/2371316.2371355.

[33] Zibula, A. and Majchrzak, TimA. 2012. Cross-Platform Development Using HTML5, jQuery Mobile, and PhoneGap: Realizing a Smart Meter Application, In *WEBIST (Selected Papers): SPRINGER, 2012 (Lecture Notes in Business Information Processing)*, ISBN 978-3-642-36607-9, Vol.140, 16-33.