Native versus web applications for smart phones

Is HTML5 the solution for cross-platform development?

Marko Helin
Royal Institute of Technology, Stockholm, Sweden
helinm@kth.se

Fredrik Wallenius
Royal Institute of Technology, Stockholm, Sweden
walleniu@kth.se

Abstract: When developing a mobile application the developer faces a problem with today's many different mobile platforms. If a native application is to be released, one version has to be developed for each platform. An alternate solution is to release a web application adapted for viewing on smartphones. In this document we describe the difference between the techniques and show that it is the purpose and functionality of the application that determine whether it can be created using web technologies or if a native application has to be developed.

Keywords: HTML5, Web application, multi-platform deployment

1. Introduction

Developers that wish to deploy their applications to multiple platforms have to make a decision on whether to go native for each platform or to build a cross-compatible web application.

Before a decision can be made, the following aspects and the problems they imply must be considered:

i) The services the application needs access to
ii) The budget and target platforms
iii) Application deployment
iv) User interface requirements
v) The sales process
vi) Marketing

An alternate solution for cross-platform deployment is the use of middle-layer interfaces, such as *PhoneGap* [1] or *Mosync* [2]. These middle-layer vendors provide developers with an integrated development environment (IDE) including compilers that transform source code to a platform of choice.

In this paper we lay down the similarities and differences between developing a native application and a web application. We point out a few strengths and weaknesses with both techniques.

2. Evaluation

2.1. The services the application needs access to

Mobile device services are consumed via an application programming interface (API).

These differ in mainly two ways:

i) The services they provide (e.g. GPS, accelerometers, data storage)
ii) The programming language

2.1.1. Services

Developers going for the native approach download a software development kit (SDK) and make calls to objects in the API to consume the services of the device, whilst HTML is a tag-based protocol. The browser maps the HTML tags to services that in turn supply the web application with the appropriate data.

Location based services are already defined in HTML. In 2012 we might have support for camera and near field communication tags.

2.1.2. Programming language

When working with HTML5, the browsers are the middle-layer responsible of rendering your work uniformly across different platforms. This is very similar to what Phone Gap and Mosync among others do: the hard work of mapping device services to a uniform API. In native development, the programmer is in full control of the look and feel, but also needs more specialized knowledge of the native programming interface. This is an obvious advantage for web developers: they just need to know one API and get support for all devices.

The core problem is that mobile devices evolve first - their native development environments included, while HTML and 3rd party middle-ware developers are always one step behind.

2.2. The budget and target platforms

One significant factor behind the decision is whether the application is to be released to all platforms or just one. Costs will be significantly higher if you choose to develop several native releases compared to making one web application. Not only is specialist competence required for each platform, but separate developing, testing and deployment projects will take place. Targeting several platforms will also unquestionably have a significant impact on the amount of time spent on code maintenance.

It is within this area a web application has it's strongest advantage. Correctly developed it will look and function similar on all smartphones. That the development and maintenance cost will be lower goes without saying.

2.3. Application deployment

A native application is usually distributed through a special application on the smart phone, for example through the iPhones App Store or the Android Market. To the user this is a convenient way to both find and install applications.

A web application, however, is “started” by entering a URL in the smartphone’s web browser. Although easily accessed the application is easily forgotten once the web browser is used for something else. The application itself does not create a shortcut icon in the application launcher, as is the case with native applications.

The deployment process differs from the developer’s point of view. On some platforms the application must go through an approval process that might span over several weeks. The users need to accept the update. In a web application new features are available to the users at the same moment the developer publishes them to the web server.

Some developers choose a middle way. A native application is released that holds an embedded browser that points to the publisher’s web application.

2.4. User interface requirements

The quality of the user interface is determined by three factors:

i) GUI elements
ii) Responsiveness
iii) Interaction design

2.4.1. GUI elements

Interfaces with rounded corners, zoomable pictures, drag and drop functionality and dialog windows among other have been hard to accomplish in HTML and JavaScript. The development of the CSS3 and HTML5 recommendations has been made with these cases in mind:

The main area that has not been adequately addressed by HTML is a vague subject referred to as Web Applications. This specification attempts to rectify this, while at the same time updating the HTML specifications to address issues raised in the past few years.[3]

An example is the support for scalable vector graphics (SVG) in HTML5.

It is also possible, using various frameworks designed for it, to mimic the look of the native graphical components (buttons, lists and such) in a web application. One such framework is jQuery Mobile [4].

When it comes to releasing graphic-intense games for smartphones, the developer only has one choice - to develop native applications for each platform. There are, however, coming standards for displaying advanced graphics in a web browser, such as WebGL 3D [5]. Though these are not yet fully supported on the smartphone platforms.

2.4.2. Responsiveness

A requirement for many mobile applications is that they should be able to be used without an Internet connection. HTML5 solves this with its cache manifest [6] that forces the user to completely download and cache all the HTML documents and scripts that are needed for the application to run.

The table below summarizes mobile browsers overall HTML5 compliance compared to a couple of standard desktop browsers. It would be interesting to make a similar benchmark illustrating native performance vs HTML5 rendering performance in terms of speed and other quality metrics - not only HTML5 compliance. This is, however, not within the scope of this paper.

User Agent	Total score (HTML compliance)
Chrome 11.0.659	293
Safari 5.1	273
Firefox Beta 4.0b7	264
Opera 11.50	254
Blackberry 6.0.0	240
Opera Mobile 11.00	234
Android 3.0	218
iPad 4.3	206
iPhone 4.3	206
Android 2.3	182
Opera 10.51	174
IE 9.0	130
Android 1.6	44
Nokia 7	29

Table 1. Comparison of mobile/desktop browser’s HTML5-compliance [7]

If the demand for responsiveness is high, developing a native application is the better choice, because parsing and interpreting is slower than running a compiled program. Web applications are getting more responsive, much thanks to the browsers increasing JavaScript performance. Table 1 clearly shows that the gap between desktop and mobile browsers in terms of HTML5 compliance is getting ever smaller, which means that the quality of the mobile HTML5 experience is close to using a regular desktop browser.

2.4.3. Interaction design

Some smartphone vendors, such as Apple, have guidelines for how the user interface is to be designed for best user experience [8]. It is the fact that the guidelines exist, and not the quality of the guidelines themself, that increase the quality of the user interface. A new application looks and behaves similar to other applications providing the user with a familiar interface, which increases learnability.

In HTML5 and CSS3, there are few limitations on how to design your user interface. A web developer can easily follow Apples guidelines, even if the code behind is HTML/CSS/JS.

2.5. The sales process

We identify four revenue models:

i) Onetime download charge
ii) Subscription
iii) In application advertising
iv) Microtransactions

2.5.1. Onetime charge

The ‘onetime charge’-payment occurs when a user downloads an application from an authorized market service and pays for the download. Many platform vendors offer excellent services to manage this process (e.g. Apple, Google, Nokia). It is straightforward to publish the application and to get paid. For the user it usually takes a few clicks to buy applications (money is automatically withdrawn from a previously provided credit card).

A web application can be implemented in two ways:

i) HTML-document with associated CSS/JS that are hosted on a server
ii) Native application that holds a browser-wrapper

If the application is published using the first method it is more difficult to charge a onetime fee. A registration procedure is needed and the payment will have to go through another channel. In the latter case the same procedure as for native applications can be used.

2.5.2. Subscription

For some types of applications a monthly or annual subscription fee is charged (for example Spotify). The procedure will be essentially the same for native and web applications, since a user registration is required in both cases.

2.5.3. Advertising

There is no relevant difference between native and web applications when it comes to advertising. Both have support for in-application commercials.

2.5.4. Microtransactions

The application is free to download, but the user can pay for extra functionality and features. An example is a music player application through which the user can buy music. Both Android and IOS support these in-app micro transactions.

This would be doable in a web application as well, however not as simple and transparent.

2.6. Marketing

With application downloader software, marketing becomes easy as telling the users to search for a certain title in the application store. It's harder to find web applications, since the market channels are diverse.

3. Conclusion

Our conclusion is that a few key factors can be used when deciding whether to develop a native application or if to use web technologies. When developing an application with less advanced functionalities a web application is a great and cost-effective alternative. However if access to hardware near APIs, advanced graphics or the effective charging methods offered by for example Apple and Google are needed the developer will have to make native applications. The development within web technologies is fast paced though and this might change over the coming years.

References

[1]
PhoneGap, 2011-03-10
http://www.phonegap.com

[2]
Mosync, 2011-03-10
http://www.mosync.com

[3]
W3C, 2011-03-10
http://www.w3.org/TR/2009/WD-html5-20090212/introduction.html

[4]
jQuery Mobile, 2011-03-10
http://jquerymobile.com

[5]
WebGL, 2011-03-10
http://www.khronos.org/webgl/

[6]
W3C, 2011-03-10
http://www.w3.org/TR/html5/offline.html

[7]
Apple, 2011-03-10
http://developer.apple.com/library/ios/#documentation/UserExperience/Conceptual/MobileHIG/Introduction/Introduction.html

[8]
Browser scopes HTML5 test, 2011-03-10
http://www.browserscope.org/browsers