Native versus Cross-platform frameworks for mobile application development

Rosario Madaudo
MOVIA SpA, Milano, Italy
rmadaudo@movia.biz

Patrizia Scandurra
Università degli Studi di Bergamo, Dipartimento di Ingegneria, Dalmine (BG), Italy
patrizia.scandurra@unibg.it

ABSTRACT
Mobile application development is vibrant and reach of opportunities. However, new development questions arise, including what devices to target and which development frameworks to use for creating simple and reliable applications.

Based on our recent experience on developing for mobile devices, in this paper we compare the two main approaches for developing mobile applications (or simply apps), namely i) by using native APIs of the target platform and ii) by adopting cross-platform environments (mostly WEB-based). In particular, we examine the role of Eclipse in both approaches and present our position on combining preciseness and efficiency of native APIs with the flexibility and automation of the Web-based frameworks to achieve significant boosts in both productivity and quality in mobile application development.

1. INTRODUCTION
In recent years, our increasing dependence on mobile applications running on a wide spectrum of new devices, from smartphones to tablets, are rapidly posing new development challenges [1]. With mobile and the proliferation of operating systems – the so called “platforms” (including Android, Apple iOS, Microsoft Windows Mobile and Microsoft Phone 7, RIM BlackBerry, etc.) – even experienced developers are left feeling like beginners [2]. All of the tools, processes, and techniques they have acquired to build, debug, test, and deploy software are suddenly powerless against mobile.

In particular, recently there has been a proliferation of development environments specific to the mobile world. Developers can choose from either native development tools for each of the major mobile devices and platforms and cross-platform environments (like PhoneGap [5] and Appcelerator’s Titanium [6]) to create an application that run, at least in principle, across multiple mobile devices and platforms. So new development questions arise for creating simple and reliable applications, including what devices to target and which development frameworks to use.

Based on our recent experience on developing for Blackberry and Android mobile devices, in this position paper we try to provide an answer to two fundamental questions. What kinds of mobile development environments are available, and what are their advantages and disadvantages? Specifically, we compare the two main approaches for developing mobile applications, namely i) by using native APIs and toolkits for the target platform and ii) by adopting cross-platform environments, mostly WEB-based. We present our position on combining preciseness and efficiency of native tools with the flexibility and automation of the cross-platform frameworks to achieve significant boosts in both productivity and quality in mobile application development. Finally, we examine the role of Eclipse in both approaches.

2. DEVELOPMENT TOOLS FOR MOBILE APPLICATIONS
Tools vendors have created multiple development environments, but they fall in two main categories: native tools and cross-platform tools.

Native tools. They are designed to create applications that run on specific platforms. For example, in the case of Android, this normally means Java and the Android SDK (software development kit). In the case of Apple iOS, development is based on the Objective-C programming language and a toolset for designing and distributing applications include the Xcode IDE for UIs, performance analysis tools, and iOS Simulator. Though based in the past on Java, Blackberry provides for the recent BlackBerry 10 operating system an C/C++ app framework and a plug-in Eclipse. Similarly for other platforms.

Cross-platform tools. They provide developers the flexibility to create an application that run across multiple mobile devices according to the ideal principle “write-once-run-everywhere”. Examples of cross-platform frameworks that we experienced with are Appcelerator’s Titanium and PhoneGap. They are designed to limit the development work and costs to create applications for iOS, Android, BlackBerry, Windows Phone and beyond. An entire sub-industry of development tools and languages exist to develop and deploy a mobile application to multiple platforms.

Basically, most of the existing mobile cross-platform environments are toward open-source world and Web-oriented by incorporating three key technologies: HTML5, Cascading Style Sheets (CSS), and JavaScript. However, they adopt different development approaches. There are web apps that run inside of a browser (either standalone or embedded into a container to more closely mimic a native application). This approach is adopted, for example, by PhoneGap. Phone-
Gap uses HTML5 inside of a WebView\(^1\) on the device. It essentially creates a mobile web app that sits inside a native application wrapper. The web code is packaged with a library that bridges web code to native functionality.

There are also approaches that include their own runtime, like Adobe Air\([7]\). And there are tools, such as Titanium and Corona SDK\([8]\), that allow you to write the code in an abstracted scripting language, and then generate native code at compile time. Titanium, for example, compiles the JavaScript code into a native binary-converting the JavaScript into native classes and object files (whereas PhoneGap simply renders a WebView with the code being interpreted inside). Though, it is close to pure native mapping, there is still an interpreter running in interpreted mode to allow, for example, dynamic code.

As cross-platform tools, there also exist Mobile Enterprise Application Platforms (MEAP) (including Antenna Software Inc. and Kony Solutions Inc.)\([3]\) that have more full-fledged development environments, with a wider variety of traditional tools such as graphical user interfaces, version control, and workflow. They tend to have more integration tools and gateways to third-party services (such as Facebook and Twitter), as well as better technical support capabilities. In addition, they focus on the enterprise segment and strive to incorporate stronger security capabilities, taking into account that the applications will be used to access back-end corporate information.

### 2.1 Cross-platform vs. native app development

Both development approaches has advantages and drawbacks. Table 1 summarizes our comparison of the two approaches according to the following criteria (a subset has been taken and revised from\([4]\)).

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Cross-platform</th>
<th>Native</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>UI User Experience.</strong></td>
<td>Native apps provide a more fluid and responsive interface than cross-platform solutions, especially for animations and gestures. This is because when coding with the indented programming language of the platform you have access to the full device APIs. Though cross-platform solutions offer native APIs to use, it always refers to a limited subset of the device-specific features and often you have to wait until they are released in order to use them.</td>
<td></td>
</tr>
<tr>
<td><strong>Performance.</strong></td>
<td>One key advantage of using native development tools is that applications run more smoothly on whichever mobile devices use that operating system. Instead, the cross-compilation process can sometimes be slower than using native tools for an app. This difference can be easily noted during graphical rendering and animations.</td>
<td></td>
</tr>
<tr>
<td><strong>Device-specific features.</strong></td>
<td>In addition, native tools let developers take full advantages of platform functionality. On the contrary, a cross-platform application serves everyone, but has more limited functionality.</td>
<td></td>
</tr>
<tr>
<td><strong>Security.</strong></td>
<td>Cross-platform apps present more security risks than native apps. The reason is that they inherit the same risks of HTML5. For example, to name a few, application source code is freely available on the mobile device, data cached on the device (within the browser) not properly secured and encrypted, URL security vulnerabilities. On the contrary, operating systems like iOS and Android offer built-in security services like data encryption. Attack techniques such as cookie manipulation and SQL injection to gather sensitive data from back end servers and from the mobile device itself are not possible in a (well-built) native app.</td>
<td></td>
</tr>
<tr>
<td><strong>Timely access to new OS innovations.</strong></td>
<td>A cross-platform framework may not support every feature of an operating system or device. Any new added feature of the operating system is not immediately available on the cross-platform framework you are using. You need to wait it is updated to support those new features.</td>
<td></td>
</tr>
</tbody>
</table>

\(^1\)A browser screen within the native application that then renders the HTML5/CSS/JavaScript page.
### Table 1: Cross-platform vs. native development

<table>
<thead>
<tr>
<th></th>
<th>Native</th>
<th>Cross-platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>UI User Experience</td>
<td>high</td>
<td>low</td>
</tr>
<tr>
<td>Performance</td>
<td>high</td>
<td>low</td>
</tr>
<tr>
<td>Device-specific features</td>
<td>high</td>
<td>low</td>
</tr>
<tr>
<td>Distribution via app-store</td>
<td>high</td>
<td>low</td>
</tr>
<tr>
<td>Multiple platforms deployment costs</td>
<td>high</td>
<td>low</td>
</tr>
<tr>
<td>Developers support</td>
<td>high</td>
<td>low</td>
</tr>
<tr>
<td>Security</td>
<td>high</td>
<td>low</td>
</tr>
<tr>
<td>Timely access to new OS innovations</td>
<td>high</td>
<td>low</td>
</tr>
<tr>
<td>Code reusability</td>
<td>low</td>
<td>high</td>
</tr>
<tr>
<td>Design challenges</td>
<td>low</td>
<td>high</td>
</tr>
<tr>
<td>Availability of programming expertise</td>
<td>low</td>
<td>high</td>
</tr>
</tbody>
</table>

**Code reusability.** Cross app code is considered reusable. Rather than having to write a specific action or a sequence of actions for each target platform, a cross developer can just write the code once and then reuse it on other platforms or in other projects. This is not always true. Often some cross-platform frameworks often use their own subsets of JavaScript, which means that if you want to switch to another platform, that code you wrote before is likely not going to be reusable without refactoring or substantial changes.

**Design challenges.** In native development, design is simplified by the support and services provided by the operating system. The operating system can, for example, notify applications about events such as message arrival and power levels. In a cross-platform environment, developers will need to add such features explicitly.

Moreover, with cross-platform frameworks, developers must design how each feature they need has to be implemented on each target platform. For instance, designing an app for the iPhone is different than designing one for Android; the UI and user experience conventions are different, and touch points and menus work in different ways. Personally, we think a good cross-platform application looks at home on whatever platform it is used on. A bad cross-platform tries to look identical everywhere.

**Availability of programming expertise.** It is widely acknowledged that there are more Web developers than native developers. Since most cross-platform frameworks are based on HTML5 CSS3, they are easy for web developers to jump in and use alongside the calls to more native functions. On the other hand, due to the low availability, native developer skills usually cost more.

### 3. Eclipse for Mobile Application Development

Some tools, native or cross-platform, offer easy access to plugins and modules that can easily plug into other services or tools including Eclipse. For example, both Titanium and PhoneGap are released as Eclipse-based environments. Providing compatibility with common integrated development environments (IDEs) like Eclipse is to be considered a fundamental feature that enable developers to leverage existing tools and expertise.

The real challenge is to find the next generation of development tools and development processes that make mobile application development as productive and manageable as desktop and web development have been for so long. To achieve these productivity goals, there are five aspects requiring for better mobile development tools and for which the Eclipse-based ecosystem may help:

**Building.** Many platforms means many different “build” requirements for writing and compiling an app for iOS, Android, Windows Phone, BlackBerry etc. The building phase requires at the moment the use of different IDEs, SDKs, and operating systems. Also cross-platform apps, which leverage existing Web skills to reach multiple platforms, require lots of complex and messy configurations for each target OS.

New and improved tools to help mobile developers abstract platform differences and manage the building phase in less time are necessary.

**Debugging.** Mobile app debugging is based on mobile operating system emulators that implies software being written on a PC, run on a device, and then debugged from the PC. This is quite sufficient. Tools that makes mobile apps painless to debug on mobile devices are necessary.

**Testing.** Once an app is built, it needs automated tests to ensure it works properly before updates are shipped to app stores and users. For conventional applications this is a relatively straightforward task and a great variety of testing automation tools and techniques exist. This not the same for mobile apps due to the wide variability of today’s mobile devices. Mobile app testing needs to happen not only on many different operating systems, but on many different physical devices.

Recently, tools and “cloud device labs” for testing are emerging, but much more is still needed to make it productive to record, playback, and manage tests across devices. For example, when using BlackBerry WebWorks and PhoneGap APIs, a sophisticated emulator called Ripple is avail-
able as multi-platform mobile environment emulator that is custom-tailored to mobile HTML5 application development and testing. The Ripple emulator is an extension of the Google Chrome browser that allows you to quickly see how your application looks and functions on multiple mobile devices and platforms in a browser-like environment. You can use the Ripple emulator to perform JavaScript debugging, HTML DOM inspection, automated testing, and multiple device and screen resolution emulation in real-time without redeploying the application or restarting the emulator.

**Deploying.** Deploying mobile apps on app stores requires many manual steps. The problem is even more challenging if the destination is not a public app store but a private group of users. These last must find their own path to employee devices. Tools that help and automate the delivering of multi-platform deployment mobile apps are required.

**Monitoring and Optimizing.** Unlike websites that live on servers or desktop applications that live on relatively stationary PCs, mobile apps get around and once deployed, they are out of your control and destined to be abandoned.

To understand what your app is doing and why, tools that help developers to monitor it in-place to final users are necessary. In particular, developers need to monitor usage and performance, watching for common user problems and those characteristics to fine-tune accurately and productively after initial use. A nice tool toward this direction is, for example, the plug-in for Google Analytics provided by PhoneGap that allow to get reports about the usage and navigation of an app.

4. **CONCLUDING REMARKS**

The application development approach developers have to choose really depends on the application requirements itself. Will it be browser-based, with little or no data saved on the mobile device? Does it require capabilities native to the operating system? Does it require security features or support capabilities?

However, by the comparison we provided in this paper, we can also conclude that both approaches have advantages and disadvantages shortly summarized in Table 1. According to the chosen criteria, we can conclude that native and cross-platform development approaches are complementary. We believe these two approaches can be combined showing how the advantages of one can be exploited to cover or weaken the disadvantages of the other. In order to combine in a tight way preciseness and efficiency of native apps with flexibility and automation of cross-platform apps, conventional software architectural design patterns may be adopted and revised to adopt an hybrid development approach. For example, by using the layer pattern, one can provide a reach UI natively and a functionality core by a cross-platform. The decision to use in a layer one or the other relies on how deeply developers want to link the application with the underlying operating system, as capabilities in one operating system may not be available in another.

Moreover, developers have to guarantee all those desired software qualities (reliability, efficiency, maintainability, etc.) also for mobile apps. To this purpose, the integration of mobile development tools with existing or new Eclipse-based tools for building, debugging, testing, deployment and optimizing is fundamental.

5. **REFERENCES**