Towards a Mobile Application Performance Benchmark

Florian Rösler
Department of Cooperative Studies
Berlin School of Economics and Law
Berlin, Germany
florian.roesler@gmail.com

André Nitze
Department of Cooperative Studies
Berlin School of Economics and Law
Berlin, Germany
andre.nitze@hwr-berlin.de

Andreas Schmietendorf
Department of Cooperative Studies
Berlin School of Economics and Law
Berlin, Germany
andreas.schmietendorf@hwr-berlin.de

***Abstract*—In this work-in-progress paper, we present our current findings concerning performance efficiency in cross-platform mobile applications (apps) and how they can contribute to a general benchmarking approach. At first, several test cases for evaluating performance of mobile applications are described. Then, the performance efficiency of native and hybrid apps is compared on a mobile device using IBM Worklight. The results show that hybrid applications still suffer performance issues in comparison to native apps. The performance deviations and reasons for them are discussed and evaluated. It is concluded that the performance of mobile applications is crucial to user experience and satisfaction. Software quality should thus not be sacrificed, despite the economic attractiveness of hybrid development approaches. The results provide a starting point for a general approach to benchmark mobile application performance, which is discussed in the end.**

Keywords- Mobile applications; benchmark; software quality; performance efficiency.

I. INTRODUCTION

The market for mobile devices is currently contested by several Operating system (OS) providers. The two most popular OSs, Android and iOS, currently, make up about 90% of the market [5], but both bear big differences in their development processes. The remaining 10% are made of less popular OSs including BlackBerry, Windows Phone and Symbian. Therefore, when developing smartphone applications, a wide range of skills is required to cover all available platforms in their native environments. To supply this diverse market, software companies need a competent workforce that is capable of handling the development for the required platforms within multiple codebases. This leads to expensive development processes and costly maintenance.

To overcome the differences among the various OSs, several cross-platform development frameworks have been published to streamline the creation of apps for multiple platforms. As most of these frameworks are based on web technologies, web developers are able to build apps without first learning specific programming skills required by the individual platforms, eliminating the need for specialists for each targeted platform. This enables companies to employ a smaller and less specialized workforce, creating a more cost efficient way to create apps for multiple OSs. On the downside, web technologies bear limitations that confront development companies with a number of tradeoffs. Some of these limitations have been already made public by scientific research, whereas others still remain unclear.

There are several aspects of performance measurement in mobile app development. Delivering products in an efficient manner demands short development cycles with high quality (i.e., few errors), which can be seen as a form of process performance. The product performance (the app itself) is primarily reflected by user ratings in the app distribution platforms. Consumer apps with poor performance can lead to disgruntled users, who delete the app and subsequently cause negative publicity. In the future, apps meant for the business sector will continue to significantly affect business processes and revenues. In this case, the impact of performance will be much more of an issue since it can significantly constrain the operation of a company. For example, when there are contracts to be approved, sales representatives must be able to quickly receive customer data or conduct other time-critical processes dependent on mobile interaction with business data.

This paper shows performance related problems that come with cross-platform approaches comprising web technology. It aims to emphasize that mobile app development should not be conducted as economically as possible, but rather in a manner that is the most appropriate for the customer.

After considering related work in the field, we will describe the technical concept of hybrid applications. Then, we will describe the method used to gather data and present our results. Eventually, we will interpret our findings and outline an approach for further research.

II. RELATED WORK

Charland and LeRoux explain the key problems of cross-platform development, which include code execution time and User Interface (UI) issues [4]. They also point out that end users care about the quality of the app more so than they do about the efforts put into its development.

According to Ohrt and Turau, the use of cross-platform frameworks results in slower launch times and bigger application package sizes in comparison to their native counterparts [10]. The results for each individual framework vary widely, from being unremarkably less efficient to being slower and bigger by several orders of magnitude.

Corral, Sillitti and Succi test the performance of cross-platform apps in terms of accessing hardware features of an Android phone [6]. They conclude that most routines, except one (launching a sound notification, 35% faster), are slower than native code. Whereas some routines are only slower by a factor of around 2, some are considerably slower, by a factor of 30 or even 500.

Toca compares several cross-platform development frameworks by measuring various functions, including start-up time and scroll performance [12]. He states that the usage of some frameworks may lead to a bad user experience; frame rates during scrolling drop to insufficient values and starting the apps sometimes takes longer than 10 seconds.

Heitkötter, Hanschke and Majchrzak identify criteria to rate cross-platform and native development [7]. Their work is based on interviews with domain experts and developing prototypes. As one of the reviewed frameworks, called PhoneGap appears as fast as its native counterparts, they conclude that cross-platform frameworks could also be an alternative when developing for a single platform.

III. HYBRID APPLICATIONS

Hybrid mobile apps are wrapped local web applications, which allow the execution of native code. This requires the native code pieces to be called out of the browser. Such a technique is known as the “PhoneGap Hack”, which led to a library for calling several device APIs [2]. These are currently included and maintained in the PhoneGap framework, also known as Apache Cordova. Apache Cordova enables the creation of cross-platform apps using only Hypertext Markup Language (HTML), Cascading Style Sheets (CSS) and JavaScript. Moreover, developers are enabled to access a device’s camera, Global Positioning System (GPS) sensor and many other device functionalities using JavaScript [1]. Cordova currently supports all the major Oss [2] and offers the possibility to implement plug-ins by the developer, which are own pieces of native code [3]. These plug-ins can then also be used with JavaScript calls.

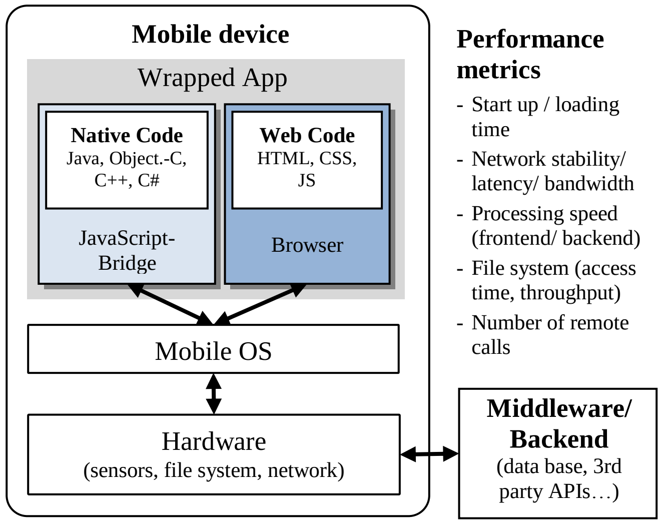
Mobile device
Wrapped App
Native Code
Java, Object.-C, C++, C#
JavaScript-Bridge
Web Code
HTML, CSS, JS
Browser
Mobile OS
Hardware
(sensors, file system, network)
Performance metrics
- Start up / loading time
- Network stability/ latency/ bandwidth
- Processing speed (frontend/ backend)
- File system (access time, throughput)
- Number of remote calls
Middleware/ Backend
(data base, 3rd party APIs…)

Figure 1. Architectural overview of hybrid mobile applications with performance metrics.

Among today’s most prominent hybrid frameworks stand the already mentioned Apache Cordova and IBM Worklight. Unlike Cordova, Worklight is distributed under a proprietary license. While Worklight comprises Apache Cordova, it adds several features aimed at business applications. These include the operation of a backend server, which supports access to different data sources [8]. It also provides multiple authentication mechanisms and security concepts for accessing business data (ibid.).

Figure 1 shows the architecture of hybrid mobile applications and exemplary metrics to measure performance in mobile software systems, including device configuration, network characteristics, and backend and third party services.

As hybrid apps at their core are web applications, they utilize UI toolkits to display user interfaces. Because UI toolkits can only imitate certain behaviour of native controls, they sometimes lack the native look and feel that most users expect [11].

IV. METHOD

For a meaningful comparison two nearly identical Android apps, containing all in the following presented test cases are developed. Besides a native app, a hybrid IBM Worklight app is created, each utilizing jQuery Mobile as its UI toolkit. The defined test cases are meant to compare these apps by the subcharacteristics of performance efficiency as described in ISO/IEC 25010 [9], namely time behavior and resource utilization. Both versions only comprise basic UI elements and no rich media. In order to minimize the interference of background threads, the used smartphone is put into flight mode during testing. Every test case is executed ten times to obtain an arithmetic mean value.

A. *System under Test*

In order to retrieve comparable results, the test cases are each executed on the same device. The chosen device is a Samsung Galaxy Tab 2 10.1, which can be classified as a mid-class tablet and should therefore provide satisfactory performance.

When measuring time behavior, two timestamps are taken; one before a test case is executed and one right after execution has finished. In the case of resource utilization, instead of timestamps, a representational key figure for the memory consumption is recorded. As apps share resources among each other on Android, we use the Private Dirty Random Access Memory (RAM) as a representative key figure. The Private Dirty RAM indicates which amount of memory is only consumed by the specific app and is therefore freed upon closing the app.

B. *Test cases*

Although Ohrt and Turau already compared the start-up time of hybrid apps as well as their memory consumption after start-up [10], we recreate their experiments. This is because their tested apps were virtually empty and the hybrid app did not contain a UI toolkit. We expect a remarkable increase in time and memory consumption when the app’s web resources are loaded. Those parts of an app cannot rely on an intelligent library sharing mechanism like Zygote, which shares Java libraries across apps.

It must be noted that Worklight apps are in general much more extensive than a basic Cordova app due to the included backend functionality. It is currently not possible to exclude these libraries from Worklight projects even when they are not used by an app.

In order to retrieve comparable values for the basic UI performance of an app, a certain amount of items are added to a list view. List views represent a common way of navigation and display of data, thus its performance is crucial to the overall impression of an app. In the case of the hybrid apps, the list items are added by utilizing standard DOM (Document Object Model)-methods. As Android utilizes data binding to connect an array to the list view, the creation of the array and its items are excluded from the time measurement. The described test case is additionally tracked in terms of memory consumption, thus indicating how efficient list items are handled by the specific system. Such a test case cannot act as a precise performance benchmark, but shall rather point out a general performance comparison as list view operations are a basic feature that should be executed close to real time. If an app struggles adding 100 items in a benchmark environment, where the number of background processes is minimized, it may have stronger execution issues when adding these items in a real life situation, where other processes take large amounts of processing power.

V. RESULTS

The outcome of the first test case reveals that, while the native application is nearly immediately loaded, the hybrid counterpart is significantly slower by a factor of around 20 (see Figure 2). With a startup time of more than two seconds,

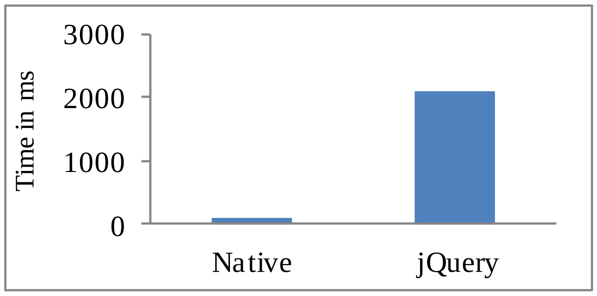

Figure 2. Start-up time comparison of native (Android) and hybrid (jQuery) apps.

the hybrid app shows a remarkable delay, which is

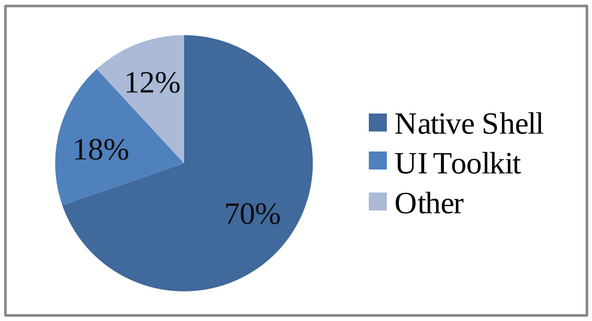

Figure 3. Start-up time details for hybrid (jQuery) apps.

noticeable by the user. In an app, which contains real content, this additional loading time may negatively influence a user’s satisfaction.

When analyzing the start-up process further, it becomes clear that the native shell, which wraps hybrid apps, takes up a majority of the time span followed by the UI toolkit's loading time (see Figure 3). During this time, the internal Cordova server is started, which refers JavaScript calls to their native counterparts. Additionally, required JavaScript libraries as well as web resources are loaded into the browser view, which hosts the app. Thus, loading a native app is a more trivial process to the operating system and can be executed much faster.

The measurement of the memory consumption during start-up shows similar results. The differences in memory

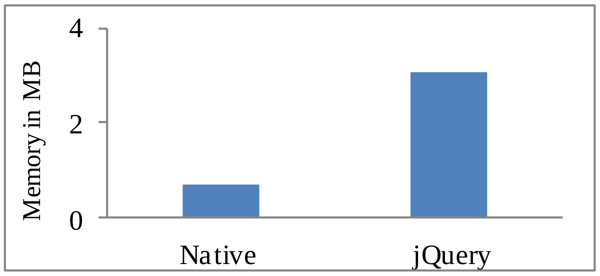

Figure 4. Memory consumption after start-up.

utilization after start-up are significant, with the hybrid application consuming more than four times the memory of the native implementation, which takes up less than 700KB (see Figure 4). This difference is explainable by the Worklight shell and the UI toolkit, which cannot be shared among hybrid apps. When running multiple hybrid apps at the same time, each utilizes its own copy of the aforementioned resources. Native Android apps on the other hand can share libraries that bring in basic functionalities like UI operations among each other, which decreases the overall memory footprint. Additionally, the DOM, which is required to display web pages within Cordova is also stored in the phone’s RAM.

The test case for adding 100 list items to a list view shows that the native implementation performs close to real time (see Figure 5). In the case of the hybrid app, the process takes nearly half a second, therefore being slower by an

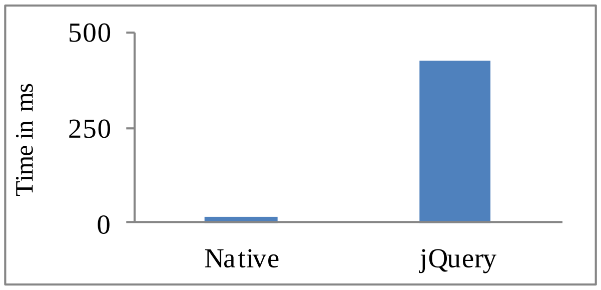

Figure 5. Time to add 100 list items.

order of magnitude. The reason for the difference when adding the items in the hybrid implementation could be the utilization of the DOM, which cannot compete with the efficiency of native UI mechanisms. Although the performance of the hybrid app is still acceptable, users might feel a delay when loading the screen with the list items, which again could affect the user's satisfaction. It also should be mentioned that low-end phones may show worse results. On low-end phones, adding list items can lead to long waiting times, which may be unacceptable for such a common operation.

The results for measuring the memory consumption when adding list items indicate that the native implementation is remarkably more efficient than the hybrid

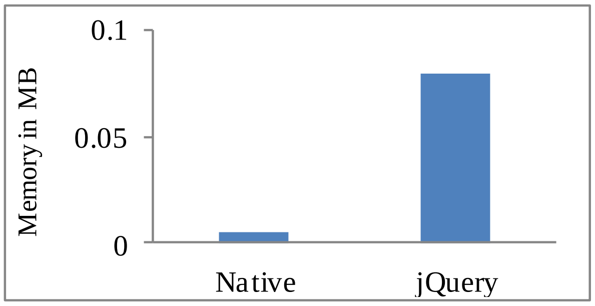

Figure 6. Increased memory consumption when adding 100 list items.

version (see Figure 6) as the jQuery Mobile app utilizes 16 times more memory than the native implementation. The reason for the higher memory increase of the hybrid implementation may again be the DOM, which stores the document in an expensive tree structure, which usually includes redundancies like recurring element names. On the contrary, the OS can handle native apps in a more efficient way and store the values in inexpensive data structures.

VI. CONCLUSION AND FUTURE WORK

Hybrid apps were analyzed in terms of performance efficiency, which is an important factor for the software quality of apps. In all conducted tests, native apps were superior to hybrid apps. Since performance is considered crucial for user experience, low performance is likely to influence a user's satisfaction and rating of the app. Users of low-end phones seem to be particularly disadvantaged by a market shift towards hybrid apps. A large share of the market for hybrid apps is currently advertised by many consulting companies due to the economically efficient development process. Despite this, companies should focus on their clients who expect a satisfying performance, which is more likely to be achieved with the native approach. Some cases cannot yet be covered sufficiently in terms of responsiveness using hybrid approaches. Although web technologies and hybrid frameworks are progressing steadily, native development prevails, at least for consumer-facing apps.

While many papers have already covered performance efficiency of hybrid mobile apps, there is still no clear statement of which approach to choose for a certain project. We therefore suggest the creation of a general benchmark method that can be implemented at the beginning of a software development project for evaluation purposes. It should cover the most important aspects of an app's performance, including the utilization of hardware features or UI performance. These tests should support lead developers and managers in deciding whether the disadvantages in performance are negligible for the certain use case.

As the environment of a hybrid app can differ in many factors like OS, hybrid shell, UI toolkit and smartphone hardware, it should be possible to implement the benchmark for a specific system in a cost efficient manner with low time expenses. However, a more general mobile application performance benchmark would need to include a set of configurations to cover the most widely used technological pathways. To achieve this, more factors apart from performance have to be incorporated as comparison criteria. Furthermore, a more typical set of UI elements should be derived from practical use cases. Also, more economical factors have to be included as their impact on the platform choice can be significant.

Model-driven development approaches like those discussed in [13][14] and [15] have not yet found wide adoption outside of academic projects and hence shall for now be excluded of performance evaluations.

Regarding future developments, it can be assumed, that the typical increase of computing speed and memory capacity of mobile devices will lead to improved performance. Nevertheless, decisions on the trade-off between performance and other factors will always have to be made.

REFERENCES

[1] Adobe PhoneGap 2013a. PhoneGap Documentation Overview. [online] Available at: <http://docs.phonegap.com/en/2.9.0/guide_overview_index.md.html#Overview> [Accessed 10 May 2014].

[2] Adobe PhoneGap 2013b. Adobe PhoneGap Build. [online] Available at: <https://build.phonegap.com/> [Accessed 10 May 2014].

[3] Adobe PhoneGap 2013c. Plugin Development Guide. [online] Available at: <http://docs.phonegap.com/en/2.8.0/guide_plugin-development_index.md.html> [Accessed 10 May 2014].

[4] Charland, A. and LeRoux, B. 2011. Mobile Application Development: Web vs. Native. In *Communications of the ACM,* vol. 54, 5 (May 2011), pp. 49-53. DOI= http://dx.doi.org/10.1145/1941487.1941504.

[5] comScore, 2013. US Smartphone Subscriber Market Share April 2013. [online] Available at: <http://www.comscore.com/Insights/Press_Releases/2013/6/comScore_Reports_April_2013_U.S._Smartphone_Subscriber_Market_Share> [Accessed 10 May 2014].

[6] Corral, L., Sillitti, A., and Succi, G. 2012. Mobile multiplatform development: An experiment for performance analysis. In *Procedia Computer Science*, vol. 10, pp. 736-743.

[7] Heitkötter, H. Hanschke, S., and Majchrzak, T. 2012. Comparing Cross-Platform Development Approaches For Mobile Applications. *Lecture Notes in Business Information Processing*, vol. 140, pp. 120-138.

[8] IBM 2012. IBM Worklight V5 Technology Overview. [pdf] Available at: <ftp://ftp.software.ibm.com/software/pdf/mobile-solutions/worklight/WSW14181USEN.pdf> [Accessed 10 May 2014].

[9] ISO 2011. ISO/IEC 25010:2011. Geneva: ISO.

[10] Ohrt, J. and Turau, V. 2012. Cross Platform Development Tools for Smartphone Applications. *IEEE Computer*, vol. 45, pp. 72-79.

[11] Quilligan, A. 2013. HTML5 Vs. Native Mobile Apps: Myths and Misconceptions. [online] Available at: <http://www.forbes.com/sites/ciocentral/2013/01/23/html5-vs-native-mobile-apps-myths-and-misconceptions/> [Accessed 10 May 2014].

[12] Toca, F. 2011. Cross-Platform-Entwickung unter iOS und Android: Technologieüberblick und Prototyp-basierte Bewertung. (Cross-Platform Development on iOS and Android) [Diploma Thesis] University of Magdeburg. Available at: <http://wwwiti.cs.uni-magdeburg.de/iti_db/publikationen/ps/12/thesisAlcalatoca.pdf > [Accessed 10 May 2014].

[13] Balagtas-Fernandez, F.T. 2008. Model-Driven Development of Mobile Applications. In: *23rd IEEE/ACM International Conference on Automated Software Engineering*, pp. 509-512.

[14] Dunkel, J. and Bruns, R. 2007. Model-Driven Architecture for Mobile Applications, In: *Proceedings of the 10th International Conference on Business Information Systems*, Springer, *v*ol. 4439, pp. 464-477.

[15] Kramer, D., Clark, T., and Oussena, S.: MobDSL: A Domain Specific Language for multiple mobile platform deployment. In: 2010 IEEE International Conference on Networked Embedded Systems for Enterprise Applications (NESEA 2010). Suzhou, S., pp. 1–7.