Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 10 (2012) 736 – 743

Procedia Computer Science

The 9th International Conference on Mobile Web Information Systems (MobiWIS)

Mobile multiplatform development: An experiment for performance analysis

Luis Corral*, Alberto Sillitti, Giancarlo Succi

Free University of Bozen-Bolzano, Piazza Domenicani 3, 39100 Bozen-Bolzano, Italy

Abstract

The variety of operating platforms in mobile devices involves separate standards, programming languages, and distribution markets. This poses a challenge on software developers, as to select what platform to develop first for. Web-based multiplatform development tools offer a solution under the principle of developing once using target-agnostic technologies, able to be deployed in multiple platforms; nonetheless, it has been reported that web-based applications suffer significant performance decreases. In this paper, we present a study to analyze the performance of mobile web applications using PhoneGap and Android OS to understand the most relevant performance matters raised by multiplatform tools. We report an experiment focused on evaluating execution time, to characterize the performance overhead found in a web app with respect to an identical native application.

Keywords: Mobile; Multiplatform; PhoneGap; Android; Performance; Execution Time

1. Introduction

The evolution of mobile systems has witnessed the change of handset terminals from simple communication devices to high-end computer equipment. Now, cellular phones are able to perform complex and critical tasks that require increasing computing capabilities, high availability, efficient performance, and much more [1]. Since the introduction of smartphones, these devices are driven by powerful operating systems that allow users to add and remove applications employing an architecture similar to a regular personal computer. Depending on the device, there is a variety of operating platforms (e.g. Android, iOS, RIM, etc.) that imply separate standards, programming languages, development tools, and even distribution markets through which users can purchase and download applications. This variety

* Corresponding author. Tel.: + 39 389 4580955.
E-mail address: luis.corral@stud-inf.unibz.it.

1877-0509 © 2012 Published by Elsevier Ltd.
doi:10.1016/j.procs.2012.06.094

poses a challenge on software developers: each platform has a vast amount of users representing potential customers of a new application. Business strategies may require software developers to target wider ranges of users, and developing just for one platform would dismiss a large number of prospective clients. Meanwhile, developing an independent software product for each platform, requires to conduct significant parts of the software life cycle several times for each released application, which may become redundant and expensive.

An efficient approach to solve this issue is the introduction of multiplatform development tools (e.g. PhoneGap, Appcelerator, SenchaTouch, etc.) that offer a solution under the principle of "develop once, deploy everywhere". These tools use technologies that are common to different platforms, such as HTML, CSS and JavaScript programming, operating the functionality of the mobile device through a set of application program interfaces (API).

The discussion on target-agnostic development on mobile devices has been covered by works that forecast a promising growth on the use of the web browser as execution environment [2, 3, 4, 5]. Development-oriented surveys and case studies [6, 7, 8] described programming experiences and lessons learned, highlighting that although mobile applications can be effectively built for more than one platform, tools still present drawbacks that prevent them from offering an integral cross-platform solution. The most important shortcomings shown are restrictions to access hardware features, constraints integrating the application with native components, and variations in user experience. It is also reported that web-based development leads to a substantial decrease in performance; however, there are no available studies that endorse this affirmation by describing in detail the level or extent of such performance decays.

In this paper, we present a study to analyze the performance of mobile web applications created on PhoneGap and deployed on Android Operating System (OS), with the objective of shedding light to understand important performance matters raised by the utilization of web-based multiplatform development tools for mobile software. We report an experiment for evaluating performance in terms of execution time, characterizing the overhead found in a web app with respect to an identical native application. The rest of this paper is organized as follows: section 2 introduces performance analysis, section 3 gives an overview on the selected development tool, section 4 describes our experimental setup and its results, and section 5 presents directions for future work and draws conclusions.

2. Performance analysis

The performance of an application can be measured in a number of ways: execution time, memory usage or battery consumption are parameters that typically yield useful values for performance assessment [9]. Our study focuses on application's execution time, as this parameter illustrates overhead that is felt directly on the user's experience when he uses an application, or when such application interacts with hardware and software resources in the device.

Evaluating the execution time is not a task that merely requires sampling the time used by a routine to run. The task of appraising two machines, languages or techniques shall bear in mind the creation of a proper environment to maximize fairness, and make use of appropriate procedures for data interpretation [10]. To have a better understanding on the impact of the use of web technologies in the performance of a mobile application, we used a set of software routines that utilize different hardware and software resources in the mobile device. After that, we implemented these routines in applications identically coded in two versions, the first one using a web-based development tool, and the second one using programming technologies native to a mobile OS. With this, we count with an experimental setting that let us fairly compare and appraise the two approaches.

We selected for this task PhoneGap as development tool and Android OS as target platform, thanks to their openness, flexibility and availability. To complete our work, we executed the two applications in an

experimental environment, measuring and comparing the execution time as the selected performance parameter.

3. PhoneGap framework

PhoneGap [11] is an open source development framework for mobile applications, incorporated to the Apache Incubator named as "Apache Cordova". PhoneGap's approach is to use the device's web browser as an intermediate level of abstraction that permits to implement the logic layer based on JavaScript and the presentation layer on HTML and CSS. Like in desktop computing, this structure is easily portable to different web browsers. Nonetheless, this only allows to create script-based applications to be executed within the web browser's runtime, and in this scope JavaScript cannot exploit all the functionality (e.g. managing hardware features) of the mobile device.

To address this problem, PhoneGap delivers a set of APIs to manipulate low-level components like telephony management, and hardware features through a native engine. These APIs are accessible to JavaScript after being exposed to the browser through the PhoneGap JavaScript engine. Thus, developers should simply concern on using web programming, since the logic layer will count on the necessary extensions and interfaces to access further resources by means of methods (Figure 1). This architecture simplifies the creation of applications that can be deployed in more than one OS, considering that each one has the capacity of running a web browser and execute a logic layer within it.

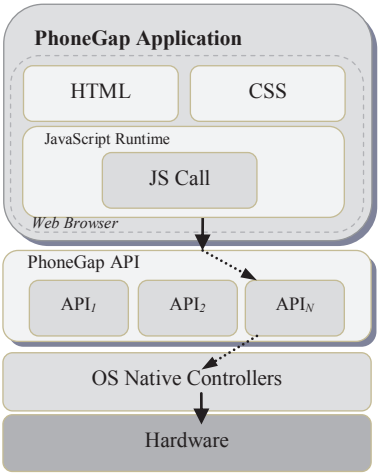

Fig. 1. PhoneGap application architecture

As of version 1.3.0, PhoneGap supports all the major mobile operating systems (e.g. Android, iOS, RIM, Windows Mobile, etc.), although in some of them it does not provide full management of device's functionality [11].

4. Analysis of execution time on web and native applications

4.1. Experimental setup

Two Android applications were coded using JavaScript programming, and regular Java programming respectively, and they were deployed in a real mobile terminal. Each application is able to call subroutines

that exercise specific features in the mobile device. Each time a routine is launched, the application records the time required to complete the job. To acquire this value, we instrumented the code by adding sentences to take a time sample immediately before calling the routine (t_1), and promptly after obtaining a successful answer of its completion (t_2).

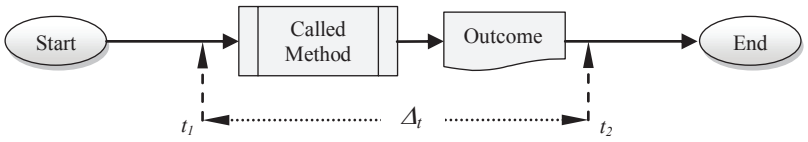

Fig 2. Operational definition of the measured job

Figure 2 represents the operational definition to determine the starting and the ending points that bound the measured task. In this manner, we assure to take into consideration the complete amount of time that implies exercising each feature, from its original invocation to its successful completion, including a tangible response. The final execution time is calculated from the difference between the two time samples (Δ_t).

4.2. Methodology

We based our analysis and data interpretation on the recommendations proposed in [10] to compare performance between two machines according to a metric. First, it is necessary to normalize the results to a "known machine" to represent relative system performance; then, calculate the geometric mean to average the normalized numbers; finally, use this geometric mean to compare relative performance and draw conclusions. Since Java is the language natively supported by Android, Java was considered as known machine. Thus, normalized values are calculated taking the time samples achieved by Java and JavaScript, and dividing each one by the Java value.

4.3. Mobile application

The mobile application consists of a graphic user interface furnished with buttons through which an operator can launch a routine to access a given feature. Both mobile applications were implemented in a way to provide the same user experience (Figure 3). The goal of each routine is to send a request to a hardware or software resource, and retrieving a response or information as means of acknowledgement that the resource was successfully accessed. The application reports and records the amount of time that was required to complete the job.

To accomplish a comprehensive analysis in the mobile terminal, we considered resources from different categories:

- *Hardware access:* access to accelerometer, launch a sound notification, trigger vibrator.
- *Network access:* request data from GPS, request network information, and
- *Data access:* write data into a file, read data from a file, retrieve data from a content provider.

When the button is pressed, a method attempts to access the selected device or resource. As per our operational definition, the application samples the time data in the instant before calling each method and after its successful completion, storing the time sample in a file for future use. To obtain the time samples, we selected the method that returns the value of the most precise system timer available. Java is able to obtain time samples in a resolution of nanoseconds, but the highest resolution of the JavaScript timer is of milliseconds, so the last unit was chosen. Each routine was exercised 1000 times to achieve statistically significant results.

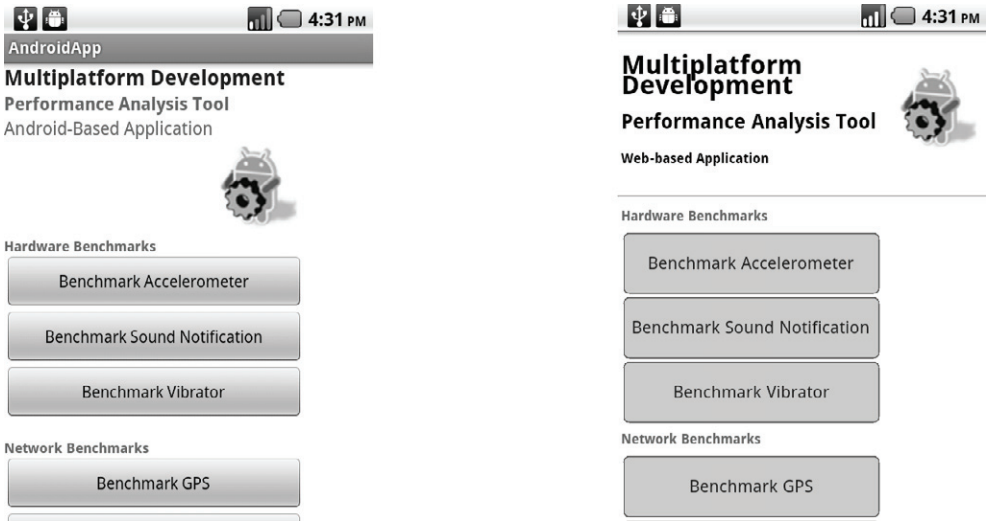

Fig 3. (a) Android native application ; (b) PhoneGap web application

The selected test bed was a cellular phone HTC Nexus One, equipped with Android OS 2.2. To ensure repeatability and reproducibility of this experimentation, we also run our tests in a HTC Magic cellphone where we could observe consistent results, yet they were kept for reference only and are not reported in this paper.

4.4. Data analysis

Results are summarized in Table 1. We report arithmetic means in milliseconds and standard deviation to illustrate the distribution of the data. To conduct performance analysis, we considered exclusively the data in relative time units obtained after normalizing all time samples with respect to the Java application, and their corresponding geometric means. As known machine, the geometric mean of Java tasks must remain constant in 1. For each JavaScript job, the geometric mean will show a value less than 1 if it is statistically more efficient, or greater than 1 if it is statistically less efficient than the same task executed by the known machine.

Table 1. Comparison of execution time between Android native application and PhoneGap web application

Measured Job	Arithmetic Mean (milliseconds)		Standard Deviation		Geometric Mean (relative)	
	Native App	Web App	Native App	Web App	Native App	Web App
Access to accelerometer	0.7136	2.0021	0.9984	3.0025	1.0000	2.5974
Launch sound notification	18.4835	26.7481	13.3665	47.5036	1.0000	0.6534
Trigger vibrator	1.5134	3.2222	1.2234	4.1248	1.0000	2.2593
Request data from GPS	2.1881	809.2352	6.7244	12.5523	1.0000	528.9298
Request network information	1.1015	1.01419	1.2052	0.6096	1.0000	1.1044
Write a file	4.7146	7.9221	9.2085	6.4558	1.0000	3.3657
Read a file	13.3036	255.7381	13.8829	74.1943	1.0000	29.9005
Retrieve data from contact list	95.8686	1841.4689	13.8747	491.5454	1.0000	18.7518

Values in Table 1 show that the web application only succeeded in executing equal or better than the native application in one routine (e. g. launching a sound notification, 35% faster). For the rest, there is a performance decay that goes from slight (e.g. requesting network information, 10% slower) to very significant (e.g. accessing the GPS sensor).

4.5. Discussion

To approximate the root cause of this difference, we analyzed the inner structure of the resource call at code level for each version, noticing that while Java programming typically uses native methods to directly access the specified resource, JavaScript is allowed to access resources only by following an execution path that implements at least one callback. This introduces waiting and execution times for calling the method, going through the callback path and waiting for the result to reach the original requestor. Execution time becomes especially high when accessing a resource that involves a complex call series to invoke the associated API.

According to PhoneGap's architecture, a JavaScript method defined at user space is sent as parameter to a foreground executive method called PhoneGap.exec(); this executive method invokes a JavaScript function (i.e. prompt()), with a twofold objective: to generate an event that may be caught by the PhoneGap's native engine, and to send the necessary parameters via a JSON string. Then, PhoneGap's native layer implements the capture of the JavaScript method and its arguments, and delegates the call to the corresponding controller or API (see Figure 4).

At the same time, a callback defined on the application's web view is notified when the JavaScript method is called, before the actual result (a prompt dialog) is shown. Finally, the native Java method checks the parameters sent, executes the request, and passes the return value back to original JavaScript caller using a string message.

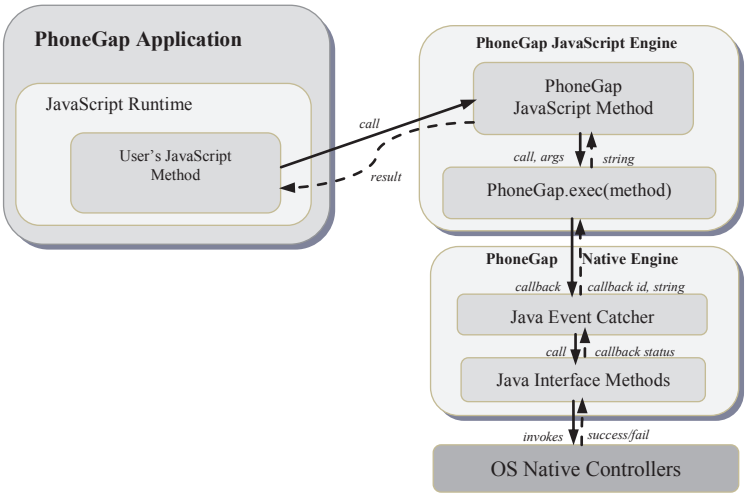

Fig 4. PhoneGap's method call flow path

Tracing the method through the callback tree and delivering the result, in combination with the resource's usual response time, introduces overhead that might make this architecture prone to become expensive, especially when resources need a complex execution tree to be accessed. Other web-based frameworks that rely on this structure to expose resource-specific methods to the web browser view

should expect similar performance losses in their outcome applications.

Although it is acknowledged that execution time increases on web applications (and in certain conditions critically), it is important to note that the experimental setup also shown that the performance penalty on a number of features commonly used is minor. These results agree with the discussion proposed in [12], where it is said that web-based mobile applications are more suitable for business applications, or applications that do not make extensive use of resource-hungry code (for example, rendering 3D graphics, or performing other heavy hardware-consuming operations).

5. Conclusions

Mobile development based on web technology is a rich field for discussion, presenting both advantages and disadvantages: It opens the opportunity of broadening the scope of a single application in a wider range of potential targets, overcoming the need of repeating platform-specific efforts through the software life cycle. On the other hand, current development tools still present limitations, particularly accessing device-specific features and interacting with other software resources. Moreover, mobile applications based on web technologies have been reported as showing important losses in performance, affecting the overall user experience.

In this work, using PhoneGap and Android OS, we analyzed specific performance matters on web-based mobile applications, showing the extent in which the execution time of a task coded using web-based programming increases with respect to an identical job developed using native, target-specific tools. In our experimentation, we exercised hardware and software features in a cellular phone, acquiring a dataset that allowed us to identify the level and cases in which execution time rises.

Using principles of machine benchmarking, we determined that in 7 out of 8 routines, web-based implementation was slower than the native one, observing that the execution time increases due to an architecture that requires to invoke methods using at least one callback and waiting for its response. This overhead grows when resources need a complex execution tree to be accessed and to send an answer back to the requestor. For general-purpose business applications, even though it is expected a performance penalty, it is noted that such penalty will be slight.

Developing using a multiplatform framework is a strategic decision that should consider different tradeoffs: while it permits to follow a “develop once, deploy anywhere” approach, the performance of the final product may not be as good as in a native application. This work contributes on analyzing and characterizing the degree and impact of such performance losses in a test application. As the focus of this paper is evaluating performance from the execution time perspective, further research should be conducted to assess other performance analysis variables (e.g. memory consumption, battery usage, user experience surveys, etc.) to consolidate a well-grounded comparison.

User experience is critical for the success of any mobile application. Developers using the web-based paradigm should understand relevant performance matters to strive for better design and coding practices, and for the improvement of multiplatform development tools to accomplish a true cross-platform, unified user experience.

References

[1] Corral L, Sillitti A, Succi G. Preparing mobile software development processes to meet mission-critical requirements. *Proceedings of the 2nd Annual Workshop on Software Engineering for Mobile Application Development, in connection with MOBICASE 2011*, pp. 9-11. 2011.

[2] Taivalsaari A, Mikkonen T, Anttonen M, Salminen M. The death of binary software: End user software to the web. Proceedings of the 9th International Conference on Creating, Connecting and Collaborating through Computing, pp.17-23. 2011.

[3] Mikkonen T, Taivalsaari A. Apps vs. open web: The battle of the decade. *Proceedings of the 2nd Annual Workshop on Software Engineering for Mobile Application Development, in connection with MOBICASE 2011*, pp. 22-26. 2011.

[4] Taivalsaari A, Mikkonen T. The web as an application platform: The saga continues. *Proceedings of the 37th EUROMICRO Conference on Software Engineering and Advanced Applications*, pp. 170-174. 2011.

[5] Corral L, Sillitti A, Succi G, Garibbo A, Ramella P. Evolution of mobile software development from platform-Specific to web-Based multiplatform paradigm. *ACM Symposium on New Ideas in Programming and Reflections on Software, (ONWARD! 2011)*, pp. 181-183. 2011.

[6] Blom S, Book M, Gruhn V, Hrushchak R, Köhler A. Write once, run anywhere – A survey of mobile runtime environments. *Proceedings of the 3rd Int. Conference on Grid and Pervasive Computing Workshops*, pp. 132-137. 2008.

[7] Geisler S, Zelazny M, Christmann S, Hagenhoff S. Empirical analysis of usage and acceptance of software distribution methods on mobile devices. *Proceedings of the 10th International Conference on Mobile Business (ICMB)*, pp. 210-218. 2011.

[8] Duarte C, Afonso AP. Developing once, deploying everywhere: A case study using JIL. *Proceedings of the 8th International Conference on Mobile Web Information Systems MobiWIS 2011*, pp. 641-644. 2011.

[9] Frisiani A. Evaluation of computer machinery (In Italian: La valutazione dei calcoatori) *Lecture notes of the Seminar on computer platforms engineering*. University of Genoa, Faculty of Engineering, Italy, 2011.

[10] Fleming P, Wallace J. How not to lie with statistics: The correct way to summarize benchmark results. *Communications of the ACM*. vol. 29, no. 3, pp. 218-221. 1986.

[11] PhoneGap version 1.3.0. Retrieved January 20th, 2012 from http://www. phonegap.com. 2012.

[12] Charland A, LeRoux B. Mobile Application Development: Web vs. Native. *Communications of the ACM*, vol. 54 , no. 5, pp. 49-53. 2011.