Performance Dissimilarities
Between Hybrid and Native Windows Phone Applications

Robert Gustavsson
M.Sc. Software Engineering
Department of Innovation, Design & Engineering, Mälardalen University
rgn09003@student.mdh.se

Dimitrios Symeon Kostopoulos
M.Sc. Software Engineering
Department of Innovation, Design & Engineering, Mälardalen University
dks12001@student.mdh.se

ABSTRACT
The need of deploying a mobile application on to multiple mobile platforms is increasing, since enterprises are aiming for as many users as possible. Therefore, a way of speeding up and simplify this process is required. Nowadays there are three different technologies, which support multiplatform deployment. This paper targets one of these technologies, named hybrid applications. The main goal is to measure performance weaknesses, thus one hybrid and one native application were built targeting Windows Phone. Performance contains various properties and this paper focuses on the property execution time. The results are displayed using a matrix containing three statistical representations. By analyzing the results the native application seems to be the fastest and the most stable when it comes to execution time.

1. INTRODUCTION
Smartphone sales in the second quarter of 2013 suggests that currently six mobile operating systems (OS) dominate the mobile market [10]. Thus, companies that intend to cover a bigger part of the market, might have to hire one developer for each aimed OS. Moreover, the chances of more operating systems to show up are high since Ubuntu's counterpart for mobile platforms is already underway [9]. The fact that each mobile platform has its own native software development kit (SDK) forces the developers to create a unique application per application store. Hence, this makes it hard to deploy an existing application on different devices, in regards of cost and needed knowledge.

In order to facilitate the work of developers, some new techniques were presented that differentiated from the standard native development. The main idea behind these new techniques is to provide a core structure that can be reused instead of redeveloping. Three technologies are most commonly used to increase portability. The first is referred to as cross compiling and translates one source language to another platform's framework [14]. Next there are mobile web applications, which use web technologies and can be viewed as normal web pages [14]. The third category, which is the main area of interest in this paper, is called hybrid mobile application [14]. By taking advantage of Web technologies, this technique increases the variety of applications that can be created, because of a web-to-native translation layer. Hybrid applications can reveal the true potential of a smartphone, by giving access to specific features. These features can be related to hardware like accelerometer sensor, or software such as contact lists.

Developers might benefit from hybrid applications in sense of time saving. Thus, they only need to be familiar with web technologies, for example HTML CSS and JavaScript, to deploy to multiple platforms. In a case study that has been conducted, measurements were taken regarding the decrease in cost related to work hours, for a company that wants to deploy their iOS application to the Android and Windows Phone market. That company would need twelve people as native developers, but only seven to form a hybrid team [12]. The degree of portability is a metric that represents whether redevelopment is more efficient or not [11]. In this case study, the measured value for degree of portability is equal to 0.41, as the hours needed by the native team are 24000 and the hybrid 14000. The value of the metric shows that the enterprise will benefit twice if they use a hybrid application instead of various native. Although, hybrid applications seem more beneficial when it comes to expenditures, one of their main consequences has been reported to be lack of performance [7].

In this paper, PhoneGap framework [15] will be used as the hybrid development tool for developing an application aiming on Windows Phone. This application will be compared to another application based on the Windows Phone SDK [16] in the sense of execution time. The selection of windows phone was made due its rapid spread over the last years. Moreover, to the best of our knowledge there has not been a comparison between a native and a hybrid application deployed on Windows Phone in the mean of execution time, so far. The tests described later corresponds to the tests by Corral et al. [1], to receive clearer understanding on how much the developers lose in performance when they embrace a hybrid development tool instead of a native one.

The following section covers the related work in the research area. Section 3 contains necessary background information of Windows Phone, Windows Phone SDK and PhoneGap. Then Section 4 refers to the method used to test the performance and section 5 summarizes the results that were gathered and their clarification. The last part, section 6, provides the conclusion and thoughts of future work.

2. RELATED WORK
There have been multiple comparisons between a native application and a hybrid application in different areas, such as user experience [7], the development differences and similarities of the applications [6, 5]. Corral et. al. [1] tested the performance differences between two applications developed using PhoneGap and Android SDK [17]. Hardware, software and also network access were tested by deploying those applications on a HTC Nexus One. The tests were run 1000 times on the device and the results were presented using arithmetic mean, standard deviation and geometric mean. As mentioned they tested different features that are supported by both Android SDK and PhoneGap, and the
outcome of the test revealed that the native application had better performance than the hybrid, when it comes to execution time.

Xanthopoulos and Xinogalos [4] tested another hybrid development framework, called Titanium [18]. Their work was a comparison on performance and development between a native and hybrid application. The outcome revealed that the native applications have better performance than hybrid, but the hybrid applications are more rapid in development, in circumstances when the application is meant for various platforms.

3. CORE TECHNOLOGIES

Windows Phone is a mobile OS that uses metro as style for its graphical user interface (GUI). Metro [2] is a style that Microsoft developed and it uses tiles to display the applications shortcuts on the home screen. It also contains new logic in the GUI that was not present in windows mobile, e.g. when a user scrolls, the scroll-speed is dependent on the force used by the user on the touch screen.

Windows Phone SDK is used for development of applications targeting Windows Phone and supports two choices for developers to use as API, SilverLight or XNA. The SDK provides an emulator for the developer to debug on and also several starting templates to endorse the easiness of the starting process. Silverlight uses Extensible Application Markup Language (XAML) for the UI and the programming language C# for the logic, and XNA uses C# only. The developer uses Visual Studio 2010 or later [19] together with the Windows Phone SDK to develop an application and it is also required to have a subscription in windows App Hub to enable uploading applications on the marketplace or debugging on devices.

Figure 2 shows the high-level architectural style of Windows Phone 7. The top layer is the application which contains either managed Silverlight or XNA, the next layer is the .NET Compact Framework which manages the communication between the managed application and the OS and the lowest layer is the O.S which controls the device.

PhoneGap is a framework developed by Nitobi in 2008 [13] as a solution for simplifying cross platform mobile development. PhoneGap is built on top of Apache Cordova and it allows developers to develop applications using current web technologies, like HyperText Markup Language (HTML) 5 for the user interface, Cascading Style Sheets (CSS) for structuring the layout and JavaScript for the logic.

When a normal web application is initiated in a smartphone a specific component is triggered, named web view. A web view is a container used to interpret HTML code and permits the user to interact with a web application. The actions of the user in such applications are limited to navigation of the web pages included in the index.html file. At this point PhoneGap acts and changes the boundaries of a web application and reduces its limitations, by providing access to a device’s hardware and software. Therefore the device specific modules can be used, as in a real native application.

A PhoneGap application structure, which is shown in Figure 2, allows the use of native activities. When a developer needs to access a device module, he utilizes the PhoneGap Application Programming Interface (API) with JavaScript. The JavaScript calls the framework’s API and then a special bridge acts as a translator between PhoneGap API and the native API. Hence, the mobile components are used implicitly and the developer is not involved in the background actions. [13]

PhoneGap supports the six dominant mobile operating systems, iOS, Android, Blackberry, Windows Phone, Bada and Symbian. For convenience, the PhoneGap framework is common for every mobile OS, which means that the code is created once and can be used for every distribution. However, one noteworthy detail exists which can be seen as a disadvantage and is related to the building process of an application. Thus, each mobile application is built with specific tools and uses different patterns for configuration. Also, taking into consideration that PhoneGap libraries are unique for every platform. Therefore, there is a need for device-specific building tools. As a result, an android application requires Android SDK, an iOS project needs XCode [21] and a Windows Phone application requires Microsoft’s Visual Studio 2010 or later, and so on.
4. ANALYSIS OF EXECUTION TIME ON HYBRID AND NATIVE APPLICATIONS

4.1 Experimental Setup
To be able to test the performance of two different SDKs two applications were needed, one per technology. One of the applications is built using PhoneGap 2.9 and the other one is simply built with the native Windows Phone SDK 8 together with Silverlight. However, during the configuration of the working environment, Windows Phone SDK must be installed in both cases. Therefore, the implementation phase for both applications was performed in Microsoft’s Visual Studio 2012 Interactive Development Environment (IDE). As explained in the previous section, PhoneGap is related to the building tools of each mobile OS and this is the main reason for using this IDE. Specifically, setting up the environment for using PhoneGap in Visual Studio is not time consuming, because its SDK can easily be integrated as a plugin template inside it [8].

As Figure 3 displays, the applications has one button per feature to trigger and in the lower end of the application, there are three labels, where the results of the tests are shown. The functionality of both applications is equivalent, they call hardware, software and retrieve a result in the same manner. Furthermore, the graphical design appearance is as similar as possible.

![Figure 3. Graphical User Interface of the hybrid application on the left and native on the right.](image)

4.2 Subject
The tests ran on a LG E900 Optimus 7, which runs WP7.8 OS. The fact that the device is running WP7.8 instead of WP8, is not an issue, since all the hardware and software is not said to be improved performance wise for the latter [3]. The tests were also ran on the 7.1 emulator that Windows Phone SDK 8 provides, to check that the results were close to ones of the real device.

4.3 Research Design
This paper tested, as it was explained, two different SDKs, which use different languages and techniques. The goal of the test procedure is to measure performance data related to each of the SDKs. As it is known performance can be measured as execution time, throughput and system availability. This research focuses on execution time calculation. The way that we collect data is similar to event-based measurement which focuses on the calculation of the time difference between the timestamp set at the beginning and the one at the end of a method.

When the data are collected, we analyze them using mathematical methods, in order to get the results. The data are analyzed using the same methods as in [1]. As a result, information about the average value will be presented both in the type of arithmetic and geometric mean and the standard deviation will be included.

4.4 Method
To measure execution time, timers were used inside the applications. These timers were timestamps that stamped when a method was called and when the result was received. With these two stamps the elapsed time could be calculated by subtracting the latest stamp from the first. As shown in Figure 4 the first stamp is made directly after the user pushes the button to call the wanted hardware or software, and the other stamp is taken when the device has returned a value. The elapsed time between the stamps shows the time it takes for the API to make a call and get a response back.

![Figure 4. Flowchart of the approach for execution time calculation.](image)

This paper will focus on testing a part of the smartphone’s features, listed below:
- **Accelerometer**: Get the current accelerometer values of the device
- **Vibrator**: Make the device vibrate
- **Notification**: Play a notification sound through the devices speaker
- **Global Positioning System (GPS)**: Retrieve the current location of the device
- **Contact List**: Retrieve contact information from the contact list

The areas in the device that are tested are hardware, software and network. Accelerometer, vibrator and notification is considered as hardware, GPS as network and the contact lists is software. By covering these areas the execution time could be measured for most of the device.

4.5 Measurements
Here a deeper insight will be given about the time measurements and the thoughts that led us to this solution. First the number of measurements had to be defined for our approach. Since, we are following the method used by Corral et. al. [1], we repeated the procedure the same amount of times and the sample size was set to 1000 values, a fair big number that ensures the plausibility of our results.
The data in all cases were collected by using a pattern of an automated function containing loops, which was repeated every 100 ms. The delay was added to provide a more realistic feeling that an user is pressing the button. In each iteration there is a start timer, then the sensor or software is triggered and we calculate the time difference. However, some of the calls to hardware or software parts, follow a slightly different sequence. Those calls are commonly known as asynchronous. Hence, when the appropriate module is initiated, we wait for its response before the calculation. In Table 1 below, all the tested features are included and they are categorized as synchronous or asynchronous, depending on the call nature. A first look reveals that 3 out of 5 modules in PhoneGap rely on callback functions, which is a statistical representation of 60%. While only 1 out of 5 modules (20%) is working asynchronously in the native SDK.

<table>
<thead>
<tr>
<th>Application type</th>
<th>Mobile module(s)</th>
<th>Call nature</th>
</tr>
</thead>
<tbody>
<tr>
<td>hybrid</td>
<td>Accelerometer, GPS, Contacts</td>
<td>asynchronous</td>
</tr>
<tr>
<td>hybrid</td>
<td>Vibrator, Sound notification</td>
<td>synchronous</td>
</tr>
<tr>
<td>native</td>
<td>Accelerometer, GPS, Vibrator, Sound notification</td>
<td>synchronous</td>
</tr>
<tr>
<td>native</td>
<td>Contacts</td>
<td>asynchronous</td>
</tr>
</tbody>
</table>

Even though the automated way of collecting sample values was a solution for most of the features, a problem arose for the GPS. When we tried to apply this pattern, we got precise values for the very first iterations, but really small or negative time values later on, which could not be taken into consideration. We were able to overcome this obstacle, by using the emulator. The emulator could load a pre-saved location data file which was set to change location every second, depending on the coordinates. Observing the sensor’s reaction in this one second interval, helped us notice our fault, which was that the device could not handle checking for a location in a shorter period of time without giving invalid results.

However, the real GPS sensor of the phone receives two type of values, continuous and single. When the sensor is set to check continuously for values, it reacts to every location change inside a time limit, with a minimal distance set. This approach seems valid, but it does not let us calculate the timestamp, as the sensor is triggered only once and the starting time is known only before the first call. When it gets a single value, it represents the current location. In that case we would have to trigger the sensor 1000 times. In this solution the current location is gathered, using the single value approach, multiple times with an interval of 1000ms for the native, and 1200ms for the hybrid.

5. RESULTS

The elapsed time was calculated by using the difference between two timestamps, one in the start and on in the end of the access. To make the data more understandable, three mathematical methods were used. The first was arithmetic mean (Equation 1) that displays the mean of the elapsed times in milliseconds and the second is standard deviation (Equation 2) which displays whether the timings are widely distributed or close to the mean, the higher the result is the more spread the values have. Respectively, if the result is low, the distribution between the values is close.

\[
\bar{a} = \frac{a_1 + a_2 + \ldots + a_n}{n}
\]

Equation 1: Arithmetic mean

\[
s = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (a_i - \bar{a})^2}
\]

Equation 2: Standard Deviation

The third and last is geometric mean (Equation 3) and it shows the typical value of the elapsed times. The results of the calculated geometric mean are presented in a different manner to distinguish the most efficient approach. Wherefore, the geometric mean is divided by the known machines geometric mean, in this case the native application, to get the relation of the execution time between the two applications. In this paper we used the logarithmic version of the geometric mean because the number of test runs generated too big data to be handled while using the product version in C#.

\[
g = \exp\left[\frac{1}{n} \sum_{i=1}^{n} \ln a_i\right]
\]

Equation 3: Geometric mean using logarithms

As mentioned in the previous section, tests were prearranged to measure the performance dissimilarities between a native Windows Phone application and a PhoneGap application deployed on a Windows Phone device. Table 2 shows the results of the tests after the executions. Arithmetic mean presents the mean in milliseconds, standard deviation displays the dissemina-

<table>
<thead>
<tr>
<th>Trigger</th>
<th>Arithmetic Mean (ms)</th>
<th>Standard Deviation</th>
<th>Geometric Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Notification</td>
<td>1.978</td>
<td>4.299</td>
<td>1.132</td>
</tr>
<tr>
<td>Vibrator</td>
<td>0.049</td>
<td>3.959</td>
<td>0.022</td>
</tr>
<tr>
<td>Accelerometer</td>
<td>0.039</td>
<td>48.05</td>
<td>0.033</td>
</tr>
<tr>
<td>Contact</td>
<td>25.470</td>
<td>240.871</td>
<td>4.552</td>
</tr>
<tr>
<td>GPS location</td>
<td>304.315</td>
<td>403.147</td>
<td>29.778</td>
</tr>
</tbody>
</table>
tion of the data. Geometric mean represents the relation, in sense of execution time, between the applications. The native application has a value of 1 and the PhoneGap application has either lower value than 1 or higher, meaning that it has a faster or a slower execution time respectively.

By examining the table, we distinguish important patterns. All the tested features execute quicker on the native application than the hybrid one. Sound notification is one of the quickest executed in both applications, with values for arithmetic mean less than 2 milliseconds (ms) for the native and close to 4ms for PhoneGap. The same execution time is visible when we tested vibrator on PhoneGap, where the arithmetic mean is close to 4ms. On the other hand the vibrator sensor was responding a lot quicker (67 times faster) in the native environment and together with accelerometer, they are the most rapid tested features with running times less than 0.05ms. In contrast, accelerometer sensor was one of the slowest on the PhoneGap application with average running time 48ms, but this value reveals one of the most surprising findings in the comparison of native and hybrid performance, as the PhoneGap application is 1321 times slower than the native one. When we executed the feature to find a contact with our applications there was a time difference, which reveals that the hybrid is 9.5 times slower when it comes to execution time. Finally, getting GPS coordinates takes the longest time for both applications and the native was 1.2 times faster than the hybrid application.

Moreover, it can be seen that most values of the samples are distributed close to their arithmetic average. Each of the tested features ran for approximately the same time in each iteration, excluding GPS for both and Contacts for the hybrid. Notification, vibrator and accelerometer have few variations in their timings and especially each of the last two sensors have returned almost exact same value times on the native. The time measures for getting data out of the contact list vary 4.5ms over or under the mean for native and 23.45ms for PhoneGap. GPS of native is also showing a high deviation of about 29.7 in the execution time, but the highest is the deviation of hybrid’s GPS (137,792). Thus, by observing the deviation results we can conclude that the native application is more consistent than the hybrid one. This means we can be more confident when we run a native application that the running time would be the same.

6. CONCLUSION AND FUTURE WORK

In this paper, two applications were developed to measure the differences regarding execution time between a native windows phone application and a hybrid windows phone application using PhoneGap. There has been previous work supporting that hybrid applications consist a good solution for an enterprise’s economy, but before this important development decisions, the drawbacks should be known. Corral et al. [1] tested the performance dissimilarities between a native android application and a PhoneGap application and their results revealed a gap and proved the former type as faster. The research of this paper aims to expose variations between a native windows phone application and a PhoneGap.

The tested features were grouped in three categories, hardware software and network. Each of those tests was repeated 1000 times before returning outputs. As presented in the table in the result section the relation, in sense of the execution time, could be interpreted, with the help of geometric mean, between the applications. Based on those values, the native application is faster than the hybrid. The native application is also steadier in execution time than hybrid, because of results of standard deviation. The values are lower in the native in comparison with the hybrid, which means that the execution time is almost always the same for the native application but could vary slightly in the hybrid.

A brief comparison with [1], makes clear that their result is equal regarding performance dissimilarities between a native and a hybrid application. In their paper they used Android as the native system, with the same features tested. In the same way, they ascertain that native android is more effective in execution time than an application developed using PhoneGap. With this knowledge we can then conclude that a native application developed using android or windows phone is faster, in sense of execution time, than an application developed using the hybrid framework PhoneGap.

As future work there is multiple things that could be extended, for example there are various ways of testing the performance on devices, for example by testing how fast an algorithm runs, how long the battery lasts, memory usage etc. This paper only covers the execution time area and for upcoming work it would make perfect sense to cover other areas as well, to get a better overview of the performance differences between the technologies. This paper along with [1], covers windows phone and android devices, it would also be beneficial to conduct test for the rest of platforms that PhoneGap supports to get a general answer about the performance of a PhoneGap application. Moreover, further focus on a hybrid application’s architecture seems promising too. This way a deeper insight for the reasons of lack of performance in sense of execution time can be acquired.

Furthermore, there is a company named AppGyver [20] who is trying to fill the gap between hybrid and native applications. They have a project called Steroid [20] and it was created with PhoneGap as its core. The goal of the project was to improve the performance of applications built on top of PhoneGap and actually reach native performance. Thereby, a comparison could be made between an application build with native SDK and an application build on Steroid, in order to find out for real if the performance values are close. The results of such a comparison may change the way of developing mobile applications in the future.

7. ACKNOWLEDGMENTS

Our thanks to Jasmina Andersson and Kyriakos Malamas for contributing with thoughts and ideas on how to improve the quality of the format of the sentences.

8. REFERENCES


