Measuring energy consumption of cross-platform frameworks for mobile applications

Matteo Ciman and Ombretta Gaggi

Department of Mathematics, University of Padua
Via Trieste 63, 35121 Padua, Italy
{mciman, gaggi}@math.unipd.it

Abstract. In this paper we analyze frameworks for mobile cross-platform development according to their influence on energy consumed by the developed applications. We consider the use of different smartphones sensors, e.g., GPS, accelerometer, etc., and features, e.g., acquiring video or audio from the environment. In particular, we have studied how the amount of required energy for the same operation changes according to the used framework. We use an hardware and software tool to measure energy consumed by the applications developed natively, as a web application or using two frameworks, Titanium and PhoneGap. Our experiments have shown that frameworks have a significant impact on energy consumption which greatly increases compared to an equal native application. Moreover, the amount of consumed energy is not the same for all frameworks.

1 INTRODUCTION

Modern mobile devices are equipped with an ample set of sensors, like GPS, accelerometer, light sensor, etc, which are able to "sense" the real world, thus allowing the implementation of more attractive applications. As an example, applications can use GPS to provide information about interesting places near the user, the accelerometer to understand the current user activity (e. g., if he/she is walking, riding a bike or using a car), and the light sensor to adapt the screen brightness.

Acquiring data from all these sensors have a cost in terms of energy and power is a vital resource, so the software implementing the data acquisition from sensors must conserve battery power to allow for smartphones to operate for days without being recharged.

Battery life is a critical performance and user experience metric on mobile devices. Even the best ever app can be deleted by the user if it wastes all smartphone power within few hours. Unfortunately, it is difficult for developers to measure the energy used by their apps both before and after its implementation. As stated by Bloom et al. [5], battery life is one of the most important aspects considered by users when dealing with mobile devices, and most of the time users request a longer battery life. For this reason, energy consumption is an extremely important factor to consider when developing applications for smartphones.

In this paper, we analyze the impact of frameworks for cross-platform development on energy consumption of mobile applications which acquire data using different smartphones sensors, e.g., GPS, accelerometer, etc., and features, e.g., acquiring video

or audio from the environment. We compare the request in terms of power consumption of different sensors, using different frameworks and with different samples frequency. We use the Monsoon Power Monitor [16] during our experiments to reduce the impact of external events. We must note here that this tool requires to have direct access to the battery, therefore cannot be used with iOS devices without opening them.

We studied both native applications and applications developed with PhoneGap and Titanium to verify whether these frameworks influence the amount of required energy for the same operation. Moreover we analyze power consumption for HTML5 applications accessed using different browsers.

Our experiments have shown that the frameworks have a significant impact on the total amount of consumed energy, since an application developed using a cross-platform framework for mobile development can consume up to 50% more energy than an equal native application, which means, a strong reduction in terms of battery life. Moreover, the amount of consumed energy is not the same for all frameworks. This means that power consumption can be one of the factors to be considered in the choice between native implementation and using a framework, or in the choice between two different frameworks.

The paper is organized as follows: related works are discussed in Section 2. Section 3 describes the frameworks for cross-platform mobile development already present in literature. The experiments made are presented in Section 4. We conclude in Section 5.

2 RELATED WORKS

Other works in literature analyze power consumption of mobile applications but they usually do not cover all sensors/features available on smartphones, and do not consider the use of cross-platform framework for development of mobile applications.

Balasubramanian et al. [4] measure energy consumption of mobile networking technologies, in particular 3G, GSM and WiFi. They find out that 3G and GSM incur in tail energy overhead since they remain in high power states also when the transfer is complete. They developed a model for energy consumed by networking activity and an efficient protocol that reduces energy consumption of common mobile applications.

A model-driven methodology to emulate the power consumption of smartphone application architectures is proposed by Thompson et al [26]. They develop SPOT, *Power Optimization Tool*, a tool that automates code generation for power consumption emulation and simplifies analysis. The tool is very useful since it allows to estimate energy consumption of potential mobile architecture, therefore *before* its implementation. This is very important since changes after the development can be very expensive. Moreover the tool is able to identify which hardware components draw significantly more power than others (e.g, GPS).

Mittal et al. [14] propose an energy emulation tool that allows to estimate the energy use for mobile applications without deploying the application on a smartphone. The tool considers the network characteristics and the processing speed. They define a power model describing different hardware components and evaluate the tool through comparison with real device energy measurements.

PowerScope [10,11] is a tool to measure energy consumption of mobile applications. The tool calculates energy consumption for each programming structure. The approach combines hardware instrumentation to measure current level with software to calculate statistical sampling of system activities. The authors show how applications can modify their behavior to preserve energy: when energy is plenty, the application allows a good user experience, otherwise it is biased toward energy conservation.

AppScope [27] is an Android-based energy metering system which estimates, in real-time, the usage of hardware components at a microscopic level. AppScope is implemented as a kernel module and provides an high accuracy, generating a low overhead. For this reason, the authors also define a power model and measure energy consumption with external tools to estimate the introduced error, which is, in the worst case of about 5.9%.

Eprof [21], is a fine-grained energy profiler for mobile apps, which accurately captures complicated power behavior of smartphone components in a system-call-driven Finite State Machine (FSM). *Eprof* tries to map the power drawn and energy consumption back to program entities. The authors analyzed the energy consumption of 21 apps from Android Market including AngryBirds, Android Browser, and Facebook, and they found that third party advertisement modules in free apps could consume up to 65-75% of the total app energy, and tracking user data (e.g., location, phone stats) consumes up to 20-30% of the total energy. Moreover, smartphone apps spend a major portion of energy in I/O components such as 3G, WiFi, and GPS.

Pathak et al. [22] study the problem of no-sleep energy bugs, i. e., errors in energy management resulting in the smartphone components staying on for an unnecessarily long period of time. They develop a static analysis tool to detect, at compile-time no-sleep bug in Android apps.

In other papers, the authors compare different framework for cross-platform mobile development according to a set of features. [12] compare jQuery Mobile [9], Sencha Touch [24], The-M-Project [20] and Google Web Toolkit combined with mgwt [13] according to a particular set of criteria, which includes license and costs, documentation and support, learning success, user interface elements, etc. They conclude that jQuery Mobile is a good solution for simple applications or as first attempt in developing mobile apps, while Sencha Touch is suited for more complex applications.

Palmieri et al. [19] evaluate Rhodes [18], PhoneGap [1], dragonRAD [25] and MoSync [17] with particular attention to the programming environment and the APIs they provide. The authors provide an analysis of the architecture of each framework and they conclude highlighting Rhodes over other frameworks, since this is the only one which supports both MVC framework and web-based services.

Phonegap, Titanium, jQuery Mobile and MoSync, are evaluated in [7] focusing on applications with animations, i.e. games. Besides the standard evaluation parameters like IDE, debug tools, programming complexity, etc., they evaluate more mobile and games-related aspects like APIs for animations, mobile devices supported, support for native user interface, performances etc. They conclude that, according to the actual state of art of the frameworks, Titanium is the best framework, since it supports animations and transitions, and its performances are good even in case of complex applications.

Issues about performances are discussed in [6]. They stated frameworks based on web technologies experience a decrease in performances when the applications implement transition effects, effects during scrolling, animations, etc., but this problem affects essentially games, while the loss in performances are unnoticeable in business application, i. e., applications which support business tasks.

All the addressed works make a critical analysis of the chosen frameworks according to criteria which never include power consumption. In some case they include performances which are considered in terms of user experience. In this paper we want to study how the use of a cross-platform framework for mobile device may affect the energy consumption of the final application with respect to native development.

3 FRAMEWORKS FOR CROSS-PLATFORM MOBILE DEVELOPMENT

According to Raj and Tolety [23], frameworks for cross-platform development can be divided into four approaches: *web*, *hybrid*, *interpreted* and *cross compiled*. They highlight strength and weakness of each approach, concluding that a preferred solution for each kind of application does not exist, but the decision about which framework to use should be made considering the features of the application to be developed. To help the developers in this decision, they provide a matrix which shows which are the best choices to develop a specific feature. This classification can help to correctly interpret the results of our tests.

The *Web Approach (WA)* allows programmers to develop a web application using HTML, CSS and Javascript. Given the new emerging features of HTML5 and CSS3, these technologies allow the creation of rich and complex applications, therefore their use cannot be considered a limitation. The application can be accessed through a mobile device using only its integrated browser, connecting to the right public internet address. Figure 1 shows this approach. Thanks to the new features introduced by HTML5 for mobile devices, web applications can now access device sensors, i.e. accelerometer, gyroscope etc., or user and device specific information, i.e. contacts. Device support for

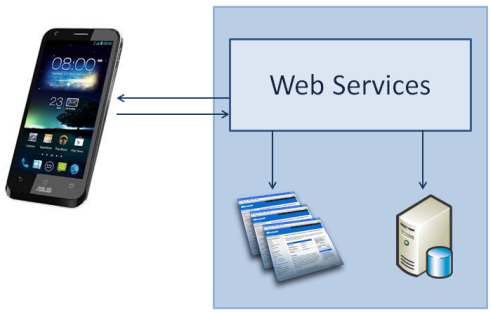

Fig. 1: Architecture of an application using the WA.

this features depends on the browser maturity used by the user, meaning that the adoption of recent features could reduce the number of final possible users. Since between different browsers the implementation of the W3C Recommendations could be different, i.e. accelerometer frequency update, the developer has to take into account these differences when developing web applications. *jQuery Mobile* [8] and *Sencha Touch* [24] are examples of this framework.

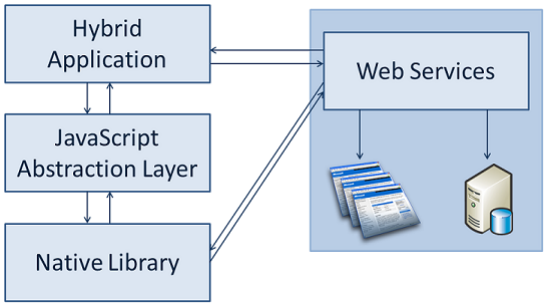

Fig. 2: *Hybrid approach* application architecture.

The *Hybrid Approach (HA)* is a middle way between *Native* and *Web Approach*. In this case, the application operates in two different ways. It uses the webkit rendering engine to display controls, buttons and animations. The webkit engine is therefore responsible to draw and manage user interface objects. On the other side, the framework and its APIs provide access to device features and use them to increase the user experience. In this case, the application will be distributed and installed and appears on the user device as native mobile applications, but the performances are often lower than native applications because its execution requires to run the browser rendering engine. An example of this kind of frameworks is PhoneGap, also known as Apache Cordova [1], and the architecture of the final application is shown in Figure 2.

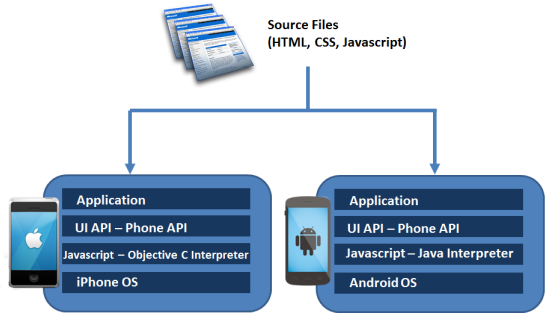

Fig. 3: *Interpreted Approach* architecture.

With the *Interpreted Approach (InA)*, the developer has the possibility to write the code of the application using a language which is different from languages natively supported by the different platforms. An example can be Javascript: developers who already knows this language are simply required to learn how to use the APIs provided by the framework. When the application is installed on the device, the final code contains also a dedicated interpreter which is used to execute the non-native code. Even in this case, the developer has access to device features (how many and which features depend on the chosen framework) through an abstraction layer. This approach allows the developer to design a final user interface that is identical to the native user interface without any additional line of code. This approach can reach an high level of reusable code, but can reduce a little the performances due to the interpretation step. Titanium [2] is an example of this kind of framework and Figure 3 shows the architecture of the framework.

The last approach, the *Cross Compiled Approach (CCA)*, lets the developer write only one application using a common programming language, e. g. C#. Frameworks which follow this approach generate, after compilation, different native applications for different mobile platforms. The final application uses native language, therefore can be considered a native mobile application to all intents and purposes. Therefore, this approach lets the programmer have full access to all the features provided by smartphones and the performances can be considered similar (even if not equal) to the native approach for simple application. In fact some tests have shown that for complex applications the native solution remains better since the generated code gives worst performances of the resulting application, compared to code written by a developer. An example of this framework is Mono [15].

4 EXPERIMENTS

Our goal is to provide objective information about the amount of consumed energy by the most common sensors available on the smartphones. To perform the best measure of energy consumption, it is important to avoid influences from external factors. A simple possibility is to run a background application that measures battery power at fixed intervals of time. This solution is adopted by some of the related works discussed in Section 2, but clearly introduces an overhead and must consider the possibility that other external events like call, messages, network problems and connectivity may influence energy consumption. Moreover these external events are almost unpredictable and unlikely reproducible, thus leading to unequal test sets.

4.1 Study Setup

For the reason mentioned before, during our experiments we use the Monsoon Power Monitor [16]. This hardware device comes with a software tool, the Power Tool, which gives the possibility to analyze data about energy consumption of any device that uses a single lithium battery. The information retrieved are energy consumption, average power and current, expected battery life, etc. These data are extremely important because can help the developer to understand which tasks use most of the energy of the

battery, for how much time, etc. Moreover, it helps to analyze and improve the application code according to its power consumption. Using data acquired by Power Monitor, it is possible to understand if and where some energy can be saved, thus increasing battery life, an extremely important aspect when working with mobile devices. Figure 4 shows the hardware setup of the system.

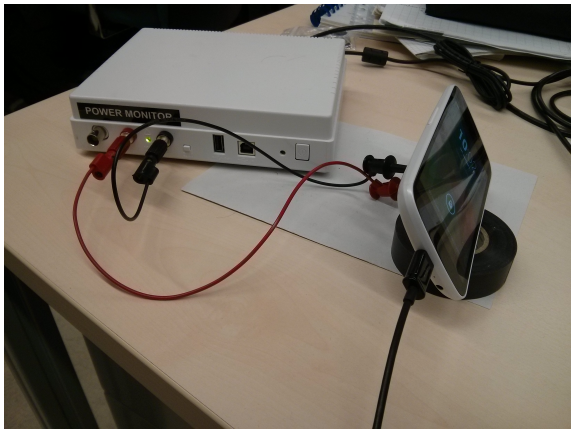

Fig. 4: Hardware setup of the system

Our goal is to compare energy consumption of applications developed in a native way or using a cross-platform framework, and to compare performances, in terms of energy consumption, among different frameworks. For each sensor that we want to analyze, we develop a basic application that simply retrieves and show data. Each application can be considered a sort of lower bound for applications behavior in terms of energy consumption, because usually applications that collect data from sensors perform several computation before updating the user interface.

As explained before, to use the Power Monitor tool it is necessary to have direct access to the battery of the smartphone, in order to be able to connect the entire system and to measure the consumed energy. For this reason, we made our comparison using an Android device with removable battery, because the iOS devices do not allow to have direct access to the battery in a safe way.

For our experiments we compared three different approaches: *Web*, *Hybrid* and *Interpreted*. To test the *Web Approach*, we developed web applications and used the most common mobile browsers available for Android: Google Chrome, Mozilla Firefox and Opera. Each web application was tested offline, with the smartphone in "flight" mode, using the new HTML5 offline feature, that lets the developer specify, through a *cache manifest* file, a list of file that the browser has to download and store in memory to show the web application even without internet connectivity. In this way we avoid interference due to the operating system which can connect to the Internet to answer to request from other apps, or to search for new updates.

For the other two approach, we select Titanium and PhoneGap. As already explained in Section 3, these two frameworks belong to two different categories. Titanium belongs to the *Interpreted* frameworks, while PhoneGap follows an *Hybrid* approach.

Our test smartphone is a Samsung Galaxy i9250. The (theoretical) capacity of the battery is 1750mAh. Even if this smartphone is not an up-to-date device, what is important for us is if the usage of a cross-platform framework has negative effects on energy consumption, and in which measure, for applications that use smartphone's sensors. With this question in mind, is easy to see that what we want to retrieve from the experiments is how much (in percentage) the energy consumption increases. Even if characteristics and performances of sensors may vary between different smartphones, our tests are performed on the same device and comparisons are made on the percentage increase of energy consumption, to limit the influence of the chosen device on our final results. We must note here, that many devices, although different in terms of brand and model, share the same hardware components for sensors.

To make our tests, we developed one application for each sensor for each framework: a native mobile application, a web application, an application built with PhoneGap and another one used to test the Titanium framework. We developed from scratch the native, the web and the Phonegap application. These applications let the user chooses, when possible, the sampling frequency. Instead, we tested the Titanium framework using the Kitchen Sink app provided by Appcelerator [3], a sample application built to show the different APIs and features provided by the framework.

All the applications retrieve data from the accelerometer and the compass, and display this information simply as a numerical value with a label. To test the microphone API, we record audio from the microphone and save it on the device as a 3GPP file. To test energy consumption due to the usage of the camera, all the developed applications show the image captured from the camera for a predefined time interval, and take a picture at its end. Since satellite signal is not available inside buildings, to test the GPS sensor we simply start a location update without showing anything.

Each test lasted two minutes. Previous experiments show that, after an initial interval, the duration of a test does not influence the final results in terms of differences of energy consumption, and we chose this amount of time to get a stable final value from the system. For our tests we used: Android API v4.2.2, PhoneGap v3.5.0, Titanium SDK v3.2., Mozilla Firefox v30.0, Google Chrome v35.0 and Opera 22.0.

4.2 Results

In this section we report the final results of our analysis. We analyze the most common sensors which can be used through the APIs provided by at least one the two analyzed frameworks. For each sensor, our application (either native or developed using a cross-platform framework) starts to acquire data coming from the selected sensor and display them on the user screen.

We made our experiments keeping the screen on, since this situation is much more adherent to reality where it's very difficult that an user interact with an application keeping the screen off: consider, as an example, Google Maps, which uses the GPS sensor to capture the position of the user and to give the correct directions.

	Native		PhoneGap		Titanium	
Sensor	**Consumed Energy (μAh)**	**Δ (%)**	**Consumed Energy (μAh)**	**Δ (%)**	**Consumed Energy (μAh)**	**Δ (%)**
Only App	5129	+0,06%	5281	+3,03%	5180	+1,06%
Accelerometer	7001	+36,58%	8971	+75,01%	9191	+79,29%
Compass	6708	+30.87%	7572	+47,72%	-	-
Microphone (Rec)	6180	+20,56%	6241	+21,76%	-	-
GPS	7320	+42,80%	7338	+43,15%	7355	+43,48%
Camera	17653	+244,37%	18062	+252,36%	17950	+250,17%

Table 1: Energy consumption comparison between native applications and apps developed using the Interpreted or Hybrid Approach.

Measures are repeated three times, in order to get a mean final value of each test. The main value we are interested in is the consumed energy, which shows how much energy is used by that particular task during the two minutes test. The smartphone was in "flight" mode (thus unable to receive any phone call or message that could arbitrarily increase energy consumption) and WiFi and Bluetooth connectivity are turned off.

To get an idea about energy consumption of an application, we need a base value to compare with. For our purpose, we decided to measure the consumed energy when the smartphone is turned on, with the screen on and without any running applications. This is the base value for our analysis. Clearly, it is quite impossible to measure the same value from another smartphone, but, since our discussion does not deal with absolute values of energy consumption, but with the increment in energy consumption in term of percentage, this initial value is important for our computation. Using our test smartphone, the consumed energy in two minutes is 5126 μAh. Moreover, the analysis of the increment in energy consumption reported in percentage helps to give results which are not tightly related to the chosen hardware.

Table 1 and Table 2 show the results of our experiments, i. e., they compare how the consumed energy increases when the smartphone uses its sensors with a native application, with a web application accessed using the different browsers and with applications developed using the other two frameworks, PhoneGap and Titanium. We must note here that native development provides full access to all the available sensors of the device, while the number and type of supported sensors may vary between different cross-platform frameworks, depending on the state of the art of the frameworks themselves. For example, Titanium provides access to the microphone and record audio data only for iOS devices and not for Android devices, while PhoneGap supports this features for iOS, Android and Windows Phone.

The columns denoted with Δ in Table 1 and Table 2 show how the consumed energy increases compared to the consumed energy of our base value, i.e. the smartphone with the screen on, without running applications. This value gives an idea of how much more expensive is to perform a particular task among our initial idle state. Table 3 presents a resume of the increments in energy consumption reporting all the Δ columns.

	Firefox		Chrome		Opera	
Sensor	**Consumed Energy (μAh)**	Δ (%)	**Consumed Energy (μAh)**	Δ (%)	**Consumed Energy (μAh)**	Δ (%)
Only App	5255	+2,51%	5315	+3,69%	5318	+3,74%
Accelerometer	13738*	+168,01% *	11040*	+115,38% *	11469*	+123,74%*
Compass	-	-	-	-	-	-
Microphone (Rec)	7862	+53.57%	7636	+48,97%	7649	+49,22%
GPS	7488	+46,07%	8329	+62,49%	8209	+60,15%
Camera	15801	+208.26%	15386	+200,15%	18354	+258,06%

Table 2: Energy consumption using the Web Approach. Note that data marked with (*) use a faster updating frequency.

	Native Δ (%)	**Firefox** Δ (%)	**Chrome** Δ (%)	**Opera** Δ (%)	**PhoneGap** Δ (%)	**Titanium** Δ (%)
Only App	+0,06%	+2,51%	+3.69%	+3,74%	+3,03%	+1,06%
Accelerometer	+36,58%	+168,01%*	+115,38%*	+123,74%*	+75,01%	+79,29%
Compass	+30,87%	-	-	-	+47,42%	-
Microphone (Rec)	+20,56%	+53.57%	+48,97%	+49,22%	+21,76%	-
GPS	+42,80%	+46,07%	+62,49%	+60,15%	+43,15%	+43,48%
Camera	+244,37%	+326,25%	+200,15%	+258,06%	+252.36%	+250,17%

Table 3: Resume table with all the tested framework. Data marked with (*) uses an higher update frequency.

As it is easy to see, the differences between power consumed by the native app and the solutions developed with a framework for cross-platform development is very high.

Let us begin the analysis with the comparison between applications which do not capture any data from sensors. The purpose of this test is essentially to investigate if the adoption of a cross-platform framework instead of a native development requires more energy consumption simply to "show" the application without any computation behind. The results are shown in the first row of Table 1 (and Table 2) denoted with the label "Only App". As we can see, the energy consumption increases from at least 1,06% (Titanium) to at most 3,74% for Opera, i.e. the adoption of a cross-platform framework produces, basically, a little more expensive applications in terms of power consumption, and this increment is not equal for all frameworks.

Considering applications that use smartphones sensors to retrieve data, we can measure differences only for sensors that are supported at least from one of the two frameworks. The measurements made show that the usage of the accelerometer requires about 39% more energy using the PhoneGap application instead of the native one. This is an extremely high value, which means that the usage of this sensor in a cross-platform application is really a battery consuming task that can decrease user experience. This value is even bigger for browsers (and a bit more for Titanium), as we will discuss later. Therefore, if the application needs to retrieve data from the accelerometer it is necessary to consider if it would be better to develop different, more performing, native applica-

tions for each platform. Even with the *Interpreted Approach*, the energy consumption is much more higher (about 43%). Moreover, we can compare PhoneGap with Titanium, showing how the latter is more expensive than the first one.

The results reported for the *Web Approach* require particular attention. Web Applications do not let the developer specify the preferred update frequency of a particular sensor. This means that new data is available at not controllable intervals by the developer. For example, with Web Applications the update frequency is in general $20Hz$ (1 sample every 50ms), while with Mozilla Firefox it is even faster (25Hz).

Analyzing the *Web Approach*, it is clear that to read data and update the user interface faster leads to an higher energy consumption with respect to the PhoneGap framework. However, the difference between PhoneGap and *Web Approach* in terms of update frequency ($15Hz$ against $20Hz$ or $25Hz$) does not justify an increase of about 40% (PhoneGap vs Chrome) to 90% (PhoneGap vs Firefox) of energy consumption, meaning that this approach is much more expensive[1].

PhoneGap performs worst even for data acquisition from compass, since it requires 17% more battery than the native solution (Titanium and web browsers do not support this feature at the time of writing).

The only two sensors where the difference between the native solution and the cross-platform framework is lower are the GPS and the microphone. Despite the *Web Approach*, where there could be several performances problems due to a not complete optimization of the sensor (Chrome and Opera with the GPS), the energy consumption of the frameworks is quite the same of the native solution, as we can see in Table 2. This behavior can be explained because in this case the cross-platform application makes only one call to the system API, and waits from the system the result (an image or location coordinates) and so the difference between the native and the cross-platform solution is really low.

Considering the energy consumption for the camera using web browsers, Mozilla Firefox and Google Chrome seem to be better that the native solution. This strange data is explained from the fact that, differently from the native solution, images coming from the camera do not cover the entire window, but only a small portion of it. This makes the application much less expensive since it collects less data. Opera, differently from the other two browsers, shows the images from the camera in the entire window, and, as we can see from Table 2, its consumption is higher that the native solution.

Comparing together all the frameworks, it is clear that the nature of the framework influences energy consumption. In particular, Web Applications are the most expensive applications, both for a not complete optimization of the API for sensor data acquisition, and because user interface updates for a browser is an expensive task. Comparing together PhoneGap and Titanium, the overhead of energy consumed introduced by the two frameworks is different when the user interface has to be updated frequently. This difference comes from the nature of the two frameworks. As already mentioned in Section 3, PhoneGap belongs to the *hybrid* family, while Titanium to the *interpreted* frameworks. This means that if we compare the two different platforms in terms of per-

[1] Since PhoneGap does not allow to have data faster than $20Hz$, it was not possible to have an objective comparison of this data.

	Consumed Energy increase (%)			
Sensor	**60ms**	**150ms**	**300ms**	**600ms**
Accelerometer	+75,01%	+38,60%	+24,09%	+15,55%
Compass	+47,72%	+41,02%	+29,36%	+19,33%

Table 4: Consumed energy using different sampling frequencies to capture data with PhoneGap.

formances and energy consumption, the *hybrid* application is better, since the consumed energy by this application is lower, meaning that the overhead introduced is lower.

Another possibility to compare different development frameworks, and in particular how sensors usage affects application performances and energy consumption, would be to test these applications without updating the values of the retrieved information on the screen or turning off the screen while performing operations. In this case, the differences between the native solution and the cross-platform solutions when acquiring data from accelerometer or compass are much more lower, about 5% (at most 20% for Web Approach). Unfortunately, this is only a theoretical result. In fact, every application that retrieves data from sensors does not use this raw data immediately, but it usually elaborates the data and updates the user interface accordingly. Therefore, to use this theoretical results to promote the use of cross-platform framework would not adhere to the reality because we would not consider two extremely important parts of the applications, i. e., data elaboration and user interface management.

If we compare together the same cross-platform application, what we can note is that the difference in terms of consumed energy to show or not to show data from sensors is about 70% (PhoneGap and Titanium) to 30% (Web Approach). This means that, without concern on the chosen cross-platform framework, the most expensive task is to update the user interface.

Unlike Titanium, PhoneGap allows the developer to decide the frequency to retrieve data from sensors. This is an extremely important option, because lets the developer to define the right update frequency depending on the target application and the needs of data. This possibility is available for both the accelerometer and the compass sensor. In this cases, we made several test at predefined update frequencies (60ms, 150ms, 300ms, 600ms) to see how, and in which measure, changing the update frequency affects energy consumption. The results are shown in Table 4. As it is easy to see, if the update frequency decreases, the consumed energy decreases, with a difference that can reach 60% between 60ms and 600ms update frequency. This means that the developer has to pay extremely attention when developing a cross-platform application, because if the user experience do not requires an extremely fast update, it is useful to reduce the update frequency of sensor data in order to save energy, and so battery life.

5 CONCLUSION

Due to the diffusion of different smartphones and tablet devices from different vendors, the cross-platform development approach, i. e., to develop only one single application and distribute it to different devices and mobile platforms, is growing and becoming extremely important. This cross-platform development incorporates several approaches,

i. e., *web*, *hybrid*, *interpreted* or *cross compiled*. All these approaches have several positive and negative aspects, either from a developer or a user point of view.

Many papers address the problem of how to choose the best framework for the development of a particular application, but, to the best of our knowledge, no one considers the power consumption as one of the key issue to make a correct choice.

In this paper we analyze the influence of the different approaches on energy consumption of mobile devices, in particular for the *Web* (Google Chrome, Mozilla Firefox and Opera), *Hybrid* (PhoneGap) and the *Interpreted* (Titanium) approach. We compared the performances of a simple application, built with a particular framework, that retrieves data from sensors and updates the user interface, to a native application with the same behavior, to measure differences in terms of power consumption. Despite other previous analysis, we do not use a software to measure energy consumption since it introduces a not valuable overhead; for this reason we used the Monsoon Power Tool which allows to measure, through hardware links, consumed energy and to understand how this consumption increases with the two frameworks.

Our comparison shows that, visualization of data (business applications) perform better on the *interpreted* approach (Titanium), that can be a good solution for simple applications that do not retrieve data from sensors, e. g., on-line shopping, home banking, games which do not use accelerometer, etc.

A particular sensor needs specific discussion. Even if Titanium seems to be the right choice if energy consumption is a key issue, our experiments have shown that it fails in case of retrieval of data from accelerometer for two reasons: the available API does not allow to impose an update frequency, and compared to other framework with the same update frequency it consumes about 5% more energy than PhoneGap. This result derives from the *Interpreted approach*: the necessary interpretation step can be extremely expensive and lead to more energy consumption. We must note here that this sensor consume a lot of energy, therefore if the application make a strong use of the accelerometer, the developer has to consider also the native solution.

The results we got show that a cross-platform approach involves an increase in terms of energy consumption that is extremely high, in particular in presence of an high usage of sensors data and user interface updates. This increase can vary even in order of about 40%, meaning that it is important to choose the right framework to preserve the battery duration and to avoid negatively affecting user experience with usage of a cross-platform framework during development. In fact, several research studies have shown that energy consumption and battery life are the most important aspects considered by mobile devices users.

Talking about the *Web Approach*, it is clear that at the moment this kind of approach is not ready for a wide adoption, since the energy required to perform the same task of a native application is extremely higher, reducing battery life and stressing of the user.

The results provided are clearly related to the state of art of the frameworks and browsers under examination at the time of writing. This means that, they can change in the future according to the improvements of the frameworks and browsers.

As future works, we plan to increase the total number of analyzed frameworks considering event the *Cross-compiled Approach*, trying to cover all the different approaches. Moreover, we will try to follow the development of the different frameworks

in order to reach a complete analysis of the different sensors provided by the smartphones and their consumption in user applications.

References

1. Apache Software Foundation: Phonegap, http://phonegap.com/ (2013)
2. Appcelerator Inc.: Titanium http://www.appcelerator.com/platform/titanium-platform/ (2013)
3. Appcelerator Inc.: Titanium Mobile Kitchen Sink Demo, https://github.com/appcelerator/KitchenSink (2013)
4. Balasubramanian, N., Balasubramanian, A., Venkataramani, A.: Energy consumption in mobile phones: a measurement study and implications for network applications. In: Proceedings of the 9th ACM SIGCOMM conference on Internet Measurement Conference. pp. 280–293. IMC '09 (2009), http://doi.acm.org/10.1145/1644893.1644927
5. Bloom, L., Eardley, R., Geelhoed, E., Manahan, M., Ranganathan, P.: Investigating the relationship between battery life and user acceptance of dynamic, energy-aware interfaces on handhelds. In: Mobile Human-Computer Interaction - MobileHCI 2004, pp. 13–24. Lecture Notes in Computer Science (2004)
6. Charland, A., Leroux, B.: Mobile application development: web vs. native. Communications of ACM 54(5), 49–53 (May 2011), http://doi.acm.org/10.1145/1941487.1941504
7. Ciman, M., Gaggi, O., Gonzo, N.: Cross-platform mobile development: A study on apps with animations. In: Proceedings of the 29th Annual ACM Symposium on Applied Computing. SAC'14 (2014)
8. Firtman, M.: jquery mobile, http://jquerymobile.com/ (2013)
9. Firtman, M.: jQuery Mobile: Up and Running - Using HTML5 to Design Web Apps for Tablets and Smartphones. O'Reilly Media (2012)
10. Flinn, J., Satyanarayanan, M.: Energy-aware adaptation for mobile applications. In: Proceedings of the seventeenth ACM Symposium on Operating Systems Principles. pp. 48–63. SOSP '99 (1999), http://doi.acm.org/10.1145/319151.319155
11. Flinn, J., Satyanarayanan, M.: Powerscope: A tool for profiling the energy usage of mobile applications. In: Proceedings of the Second IEEE Workshop on Mobile Computer Systems and Applications. WMCSA '99, IEEE Computer Society, Washington, DC, USA (1999), http://dl.acm.org/citation.cfm?id=520551.837522
12. Heitkötter, H., Hanschke, S., Majchrzak, T.: Evaluating cross-platform development approaches for mobile applications. In: Web Information Systems and Technologies, Lecture Notes in Business Information Processing, vol. 140, pp. 120–138. Springer Berlin Heidelberg (2013)
13. Kurka, D.: mgwt - making gwt work with mobile http://www.m-gwt.com/ (2013)
14. Mittal, R., Kansal, A., Chandra, R.: Empowering developers to estimate app energy consumption. In: Proceedings of the 18th annual International Conference on Mobile Computing and Networking. pp. 317–328. MobiCom '12 (2012), http://doi.acm.org/10.1145/2348543.2348583
15. Monologue Inc.: Mono framework, http://www.mono-project.com/ (2013)
16. Monsoon Solutions Inc.: http://www.msoon.com/LabEquipment/PowerMonitor/ (2013)

17. MoSync Inc.: MoSync
http://www.mosync.com (2013)
18. Motorola Solutions, Inc: Rhodes
http://www.motorolasolutions.com/us-en/rhomobile+suite/rhodes (2013)
19. Palmieri, M., Singh, I., Cicchetti, A.: Comparison of cross-platform mobile development tools. In: 16th International Conference on Intelligence in Next Generation Networks. pp. 179–186. ICIN '12 (2012)
20. Panacoda GmbH.: The-m-project
http://www.the-m-project.org/ (2013)
21. Pathak, A., Hu, Y.C., Zhang, M.: Where is the energy spent inside my app?: Fine grained energy accounting on smartphones with eprof. In: Proceedings of the 7th ACM European Conference on Computer Systems. pp. 29–42. EuroSys '12 (2012), http://doi.acm.org/10.1145/2168836.2168841
22. Pathak, A., Jindal, A., Hu, Y.C., Midkiff, S.P.: What is keeping my phone awake?: Characterizing and detecting no-sleep energy bugs in smartphone apps. In: Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services. pp. 267–280. MobiSys '12 (2012), http://doi.acm.org/10.1145/2307636.2307661
23. Raj, R., Tolety, S.: A study on approaches to build cross-platform mobile applications and criteria to select appropriate approach. In: Annual IEEE India Conference. pp. 625–629. INDICON '12 (2012)
24. Sencha Inc.: Sencha touch,
http://www.sencha.com/products/touch (2013)
25. Seregon Solutions Inc.: dragonrad
http://dragonrad.com/ (2013)
26. Thompson, C., Schmidt, D.C., Turner, H.A., White, J.: Analyzing mobile application software power consumption via model-driven engineering. In: Benavente-Peces, C., Filipe, J. (eds.) PECCS. pp. 101–113. SciTePress (2011)
27. Yoon, C., Kim, D., Jung, W., Kang, C., Cha, H.: Appscope: Application energy metering framework for android smartphones using kernel activity monitoring. In: Proceedings of the 2012 USENIX Conference on Annual Technical Conference. USENIX ATC'12 (2012)