Evaluating Impact of Cross-platform Frameworks in Energy Consumption of Mobile Applications

Matteo Ciman and Ombretta Gaggi
Department of Mathematics, University of Padua, Via Trieste 63, 35121 Padua, Italy

Keywords: Mobile Applications Development, Cross-platform Frameworks, Energy Consumption.

Abstract: In this paper we analyze energy consumption of mobile applications using 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 use of frameworks for mobile cross-platform development may influence the amount of required energy for the same operation. We use an hardware and software tool to measure energy consumption of the same application, using different sensors, when developed natively 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

With the advance of mobile technologies, modern mobile devices are equipped with an ample set of sensors, like GPS, accelerometer, light sensor, etc. The presence of these sensors allows the implementation of more and more attractive applications, which can analyze the surrounding environment to better answer to the user's need. 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. The software implementing the acquisition of data from sensors and the adaptation process must conserve battery power since energy is a vital resource for mobile computing and smartphones may operate for days without being recharged.

Battery life is, in fact, a critical performance and user experience metric on mobile devices. As stated by Bloom et al. (Bloom et al., 2004), 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. Unfortunately, it is difficult for developers to measure the energy used by their apps both before and after its implementation.

In this paper, we analyze 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, with different samples frequency. We use the Monsoon Power Monitor (Monsoon Solutions Inc., 2013) 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.

Moreover, we have studied how the use of frameworks for cross-platform development may influence the amount of required energy for the same operations. Our experiments have shown that 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 60% 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

Ciman M. and Gaggi O..
Evaluating Impact of Cross-platform Frameworks in Energy Consumption of Mobile Applications.
DOI: 10.5220/0004857604230431
In *Proceedings of the 10th International Conference on Web Information Systems and Technologies* (WEBIST-2014), pages 423-431
ISBN: 978-989-758-023-9

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., 2009) 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.

(Thompson et al., 2011) propose a model-driven methodology to emulate the power consumption of smartphone application architectures. They develop SPOT, *System 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., 2012) 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 (Flinn and Satyanarayanan, 1999a; Flinn and Satyanarayanan, 1999b) 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 (Yoon et al., 2012) 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 (Pathak et al., 2012a), 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. (Pathak et al., 2012b) 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. (Heitkötter et al., 2013) compare jQuery Mobile (Firtman, 2012), Sencha Touch (Sencha Inc., 2013), The-M-Project (Panacoda GmbH., 2013) and Google Web Toolkit combined with mgwt (Kurka, 2013) 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., 2012) evaluate Rhodes (Motorola Solutions, Inc, 2013), PhoneGap (Apache Software Foundation, 2013), dragonRAD (Seregon Solutions Inc., 2013) and MoSync (MoSync Inc., 2013) 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.

(Ciman et al., 2014) evaluate different cross-platform frameworks, i.e. Phonegap, Titanium,

jQuery Mobile and MoSync, with the focus 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 (Charland and Leroux, 2011). 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 (Raj and Tolety, 2012), 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 the programmer 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. In this way, a full devices support is guaranteed (limited only by the maturity of the native browser respect to the standard technologies specifications), but the problems arise when the application needs to access to the smartphone's sensors, because this feature is not provided for web applications. Moreover, the user does not interact with a mobile application, but he/she must run the browser to access a web page containing the web application. An example of framework using the WA is *jQuery Mobile* (Firtman, 2013).

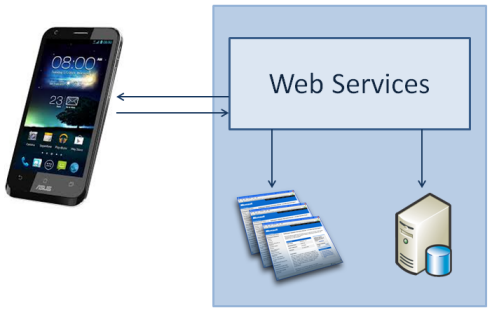

Figure 1: Architecture of an application using the WA.

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 site, 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 (Apache Software Foundation, 2013), and the architecture of the final application is shown in Figure 2.

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 the device features (how

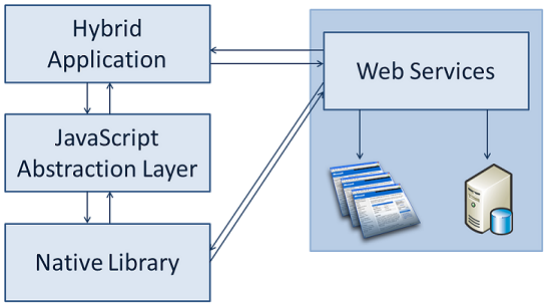

Figure 2: *Hybrid approach* application architecture.

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 (Appcelerator Inc., 2013a) is an example of this kind of framework and Figure 3 shows the architecture of the framework.

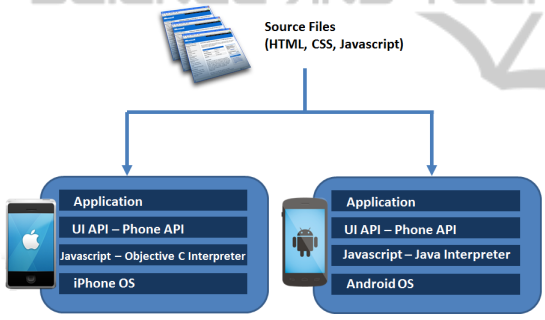

Figure 3: *Interpreted Approach* architecture.

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 is 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 (Monologue Inc., 2013).

4 EXPERIMENTS

Our goal is to provide objective information about energy consumption of the most common sensors with which smartphones are usually equipped. 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 (Monsoon Solutions Inc., 2013). The hardware device comes with a software tool, the Power Tool, which gives the possibility to analyze data of 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 increasing battery life, an extremely important aspect when working with mobile devices. Figure 4 shows the hardware setup of the system.

Our goal is to compare energy consumption of applications developed in a native way or using a

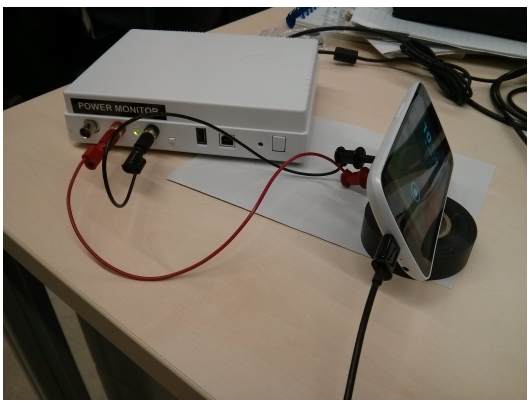

Figure 4: Hardware setup of the system.

cross-platform framework, and to compare performances, in terms of energy consumption, among different frameworks. The application used for our analysis allows to choose between several sensors and the frequency with which they retrieve data and update the user interface showing the retrieved data. This 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.

To compare energy consumption of mobile cross-platform frameworks, 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. In this way, first of all we can compare the native approach and its consumption with two different cross-platform frameworks. Then, we can even compare two different frameworks, getting information about the less expensive, in terms of power consumption, information that can influence the framework choice when considering mobile application development.

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 know, after our 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 used three different applications: a native mobile application, one built with PhoneGap and another one used to test the Titanium framework. We developed from scratch the application used to test the native solution and the Phonegap framework. These applications let the user chooses the sensor to test and the sampling frequency. Instead, we tested the Titanium framework using the Kitchen Sink app provided by Appcelerator (Appcelerator Inc., 2013b), a sample application built to show the different APIs and features provided by the framework.

All the applications retrieve data from the accelerometer, the compass and the GPS tests 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 use for the camera, all the developed applications show the images captured from the camera for a predefined time interval, and take a picture at its end. Each test lasted two minutes. Previous experiments show that, after an initial interval, the duration of a test does not influence the final value of the expected battery life, and we chose this amount of time to get a stable final value from the system. The version of the Android API was 4.2.2, PhoneGap was version 3.0.0 while Titanium SDK was version 3.2.

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.

Measures are repeated several 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 this 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 with 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 the test smartphone, the consumed energy in two minutes is 6047,31 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 shows the results of our experiments, i. e., it compares how the consumed energy increases when the smartphone uses its sensors with a native application, or with applications developed using the 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 Δ of Table 1 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.

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 denoted with the label “Only App”. As we can see, the energy consumption increases of about 7% for PhoneGap and slight more than 2% for Titanium, 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 60% 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. 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 applications for each platform.

The result reported for the Titanium framework needs a particular attention. Data reported on the table could lead to the wrong conclusion that it cost less energy to retrieve data with the Titanium framework in respect to the PhoneGap solution. The reason for this lower consumption is that the update frequency for the native (and the PhoneGap) application is about a sample every 64ms, while the Titanium frequency is about a sample every 600ms. To read data and update the user interface with a lower frequency (about 10 times lower) clearly reduces the amount of required energy. To be able to compare the performances of the Titanium framework, we used the PhoneGap application to acquire data at the right frequency (600ms) and we compared the final results (see Table 2). What we got is that, with the same update frequency, the delta of PhoneGap is about +40%, while for Titanium is about +98%. This means that, if compared together, the PhoneGap framework works better than the Titanium framework (about 60%).

All the cross-platform frameworks perform worst than the native solution also for data acquisition from compass and GPS: about 45% more power for acquisition from compass and about 10% more power for acquisition from the GPS using the PhoneGap framework; a little less than 5% more power for acquisition of data from GPS using Titanium, compass is not supported by Titanium yet.

The only sensor for which the difference in power consumption is very low is the use of camera to take pictures. In this case, the amount for energy consumed with a native application and with an application developed with a cross-platform framework differs of 8% - 12%, which is very low if we consider the total amount of energy required to use the camera (see the “Camera” row of Table 1). 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) and so the difference between the native and the cross-platform solution is really low.

Table 1: Energy consumption comparison between native applications and apps developed with a framework for cross-platform development. Note that data marked with (*) use a different updating frequency.

	Native		PhoneGap		Titanium	
Sensor	**Consumed Energy (μAh)**	**Δ (%)**	**Consumed Energy (μAh)**	**Δ (%)**	**Consumed Energy (μAh)**	**Δ (%)**
Only App	7705,54	+27,42%	8130,85	+34,45%	7860,97	+29,99%
Accelerometer	9179,99	+51,80%	12849,82	+112,49%	11972,16*	+97,97%*
Compass	9489,85	+56,93%	12124,6	+100,50%	-	-
Microphone (Rec)	8120,92	+34,29%	8404,71	+38,98%	-	-
GPS	9301,48	+53,81%	9947,60	+64,50%	9577,27	+58,37%
Camera	21857,38	+261,44%	22347,52	+269,54%	22576,45	+273,33%

A similar situation takes place for data acquired from the microphone (although test data are not available for the Titanium framework) and from the GPS. In the last case the difference, in terms of energy consumption, ranges between 5 and 11%.

Comparing the frameworks PhoneGap and Titanium, the overhead of energy consumed introduced by the two frameworks is particularly 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 performances 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 20%. 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 80% for both the framework. This essentially means that, without concern on the chosen cross-platform framework, the most expensive task for it 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 2. As it is easy to see, if the update frequency decreases, the consumed energy decreases, with a difference that can reach 70% 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

Table 2: Consumed energy using different sampling frequencies to capture data with PhoneGap.

	Consumed Energy increase (%)			
Sensor	**60ms**	**150ms**	**300ms**	**600ms**
Accelerometer	+112,49%	+70,06%	+49,84%	+40,25%
Compass	+100,50%	+75,31%	+52,92%	+46,62%

devices, in particular for the *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. Moreover, the two analyzed frameworks, PhoneGap and Titanium, have very similar performances for Camera and GPS.

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 60% 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 that 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 60%, 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.

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

As future works, we plan to increase the total number of analyzed frameworks and to add HTML5 applications in our comparisons, 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

Apache Software Foundation (2013). Phonegap, http://phonegap.com/.

Appcelerator Inc. (2013a). Titanium, http://www.appcelerator.com/platform/titanium-platform/.

Appcelerator Inc. (2013b). Titanium Mobile Kitchen Sink Demo, https://github.com/appcelerator/KitchenSink.

Balasubramanian, N., Balasubramanian, A., and Venkataramani, A. (2009). 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*, IMC '09, pages 280–293.

Bloom, L., Eardley, R., Geelhoed, E., Manahan, M., and Ranganathan, P. (2004). Investigating the relationship between battery life and user acceptance of dynamic, energy-aware interfaces on handhelds. In *Proceedings of the International Conference Human Computer Interaction with Mobile Devices & Services*, pages 13–24.

Charland, A. and Leroux, B. (2011). Mobile application development: web vs. native. *Communications of ACM*, 54(5):49–53.

Ciman, M., Gaggi, O., and Gonzo, N. (2014). Cross-platform mobile development: A study on apps with animations. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, SAC'14.

Firtman, M. (2012). *jQuery Mobile: Up and Running - Using HTML5 to Design Web Apps for Tablets and Smartphones*. O'Reilly Media.

Firtman, M. (2013). jquery mobile, http://jquerymobile.com/.

Flinn, J. and Satyanarayanan, M. (1999a). Energy-aware adaptation for mobile applications. In *Proceedings of*

the seventeenth ACM Symposium on Operating Systems Principles, SOSP '99, pages 48–63.

Flinn, J. and Satyanarayanan, M. (1999b). 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, Washington, DC, USA. IEEE Computer Society.

Heitkötter, H., Hanschke, S., and Majchrzak, T. (2013). Evaluating cross-platform development approaches for mobile applications. In Cordeiro, J. and Krempels, K.-H., editors, *Web Information Systems and Technologies*, volume 140 of *Lecture Notes in Business Information Processing*, pages 120–138. Springer Berlin Heidelberg.

Kurka, D. (2013). mgwt - Making gwt Work with Mobile. http://www.m-gwt.com/.

Mittal, R., Kansal, A., and Chandra, R. (2012). Empowering developers to estimate app energy consumption. In *Proceedings of the 18th annual International Conference on Mobile Computing and Networking*, MobiCom '12, pages 317–328.

Monologue Inc. (2013). Mono framework, http://www.mono-project.com/.

Monsoon Solutions Inc. (2013). http://www.msoon.com/LabEquipment/PowerMonitor/.

MoSync Inc. (2013). MoSync http://www.mosync.com.

Motorola Solutions, Inc (2013). Rhodes http://www.motorolasolutions.com/us-en/rhomobile+suite/rhodes.

Palmieri, M., Singh, I., and Cicchetti, A. (2012). Comparison of cross-platform mobile development tools. In *16th International Conference on Intelligence in Next Generation Networks*, ICIN '12, pages 179–186.

Panacoda GmbH. (2013). The-m-project http://www.the-m-project.org/.

Pathak, A., Hu, Y. C., and Zhang, M. (2012a). 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*, EuroSys '12, pages 29–42.

Pathak, A., Jindal, A., Hu, Y. C., and Midkiff, S. P. (2012b). 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*, MobiSys '12, pages 267–280.

Raj, R. and Tolety, S. (2012). A study on approaches to build cross-platform mobile applications and criteria to select appropriate approach. In *Annual IEEE India Conference*, INDICON '12, pages 625–629.

Sencha Inc. (2013). Sencha touch, http://www.sencha.com/products/touch.

Seregon Solutions Inc. (2013). dragonrad http://dragonrad.com/.

Thompson, C., Schmidt, D. C., Turner, H. A., and White, J. (2011). Analyzing mobile application software power consumption via model-driven engineering. In Benavente-Peces, C. and Filipe, J., editors, *PECCS*, pages 101–113. SciTePress.

Yoon, C., Kim, D., Jung, W., Kang, C., and Cha, H. (2012). 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, pages 36–36, Berkeley, CA, USA. USENIX Association.