Self-Adaptive Battery and Context Aware Mobile Application Development

Soumya Kanti Datta, Christian Bonnet, Navid Nikaein
Mobile Communication Department
EURECOM
Biot, France
Emails: {dattas, bonnet, nikaein}@eurecom.fr

***Abstract*— Overall high power consumption in the mobile applications forces the mobile users to recharge frequently. Most of the Android applications do not implement any self-adaptive strategies that react to the battery level, status and context. Thus the applications continue to consume power even when battery is critically low. Intelligent control of hardware and software optimization based on the battery level is the key to power saving. This paper introduces a self-adaptive application development framework which proposes three profiles with various self-adaptive features for mobile applications. The framework employs an analyzer engine which decides the activation of appropriate profile based on battery and context information. The self-adaption takes place in four levels – hardware & software features adaption, user features adaption and additional optimization. When the battery is critically low, priority is given to maximize the battery life until next charging opportunity. Such implementation is highly desirable for mobile applications with high dependency on display hardware (e.g. games) and/or on network operations (e.g. YouTube, Dropbox). Prototype Android applications are developed and results show up to 40% reduction in application power consumption. Power Tutor has been used to get the power consumption results.**

Keywords—Android; Battery level and status; Context; Power consumption; Self-adaptive application.

I. INTRODUCTION

Over the recent past, we have witnessed exponential growth in smart device capabilities. These devices fashion high processing power, storage, rich sensors, superior screen resolution and communication over Wi-Fi, 3G and LTE. Mobile applications have also evolved at the same pace with applications supporting complex image and audio processing, augmented reality, e-health and more. Mobile applications being the key elements in the ecosystem, pushes the demand of battery life at exponential order. Although the newer generation of devices includes batteries with increased capacity, the overall higher power consumption forces the user to recharge frequently. Limited battery life makes the functionalities of 900 million active Android devices totally dependent on the availability of battery resource.

During Google I/O 2013 event, the company stated that Google Play registers 2.5 billion of Android application download every month. But current literature suggests that, power consumption in the popular applications is very high. An investigation of internal power consumption of Android application is explained in [1]. It is found that, popular free applications consume high amount of power in displaying third party advertisements, location based user tracking and clean termination of long lived TCP connection. Also, prolong utilization of power-hungry components (display, Wi-Fi, 3G, GPS) from the applications increases power consumption [2]. In order to study the usage pattern of smart devices and get a better understanding of how application usage limits the battery life, we have developed an Android application "Power Monitor". It is seen from the usage patterns that the users who depend on applications heavily using network operations, display or location information will spend high amount of power. A plethora of researches have been conducted to minimize power consumption in Android devices [4].

This paper proposes a self-adaptive framework to facilitate the development of power efficient mobile applications. The developers should always consider power efficiency across the entire lifecycle of the applications to minimize overall consumption in mobile devices. The framework employs lightweight monitoring and analyzer engines. The former gathers battery and context information while the later analyzes them to determine the degree of self-adaption necessary. Such applications respond to the battery level, status (AC/USB charging or discharging) and context information by modifying their behavior, optimizing resource usage & performance and user experience (UX). The strategies are useful for optimizing performance without compromising UX at higher battery level. When the battery is critically low, the application will offer energy efficient alternatives to the user to minimize the battery consumption while compromising the UX.

The rest of the paper is organized as follows. Section II provides the motivation of developing self-adaptive power efficient applications. Section III describes the self-adaptive framework while Section IV presents prototype implementation along with results. A comparison of the mentioned strategies with current literature is provided in Section V and Section VI concludes the paper.

II. MOTIVATION: POWER DISSIPATION IN ANDROID DEVICES

This section motivates the reasons for self-adaptive applications by providing comprehensive overview high power dissipation in Android devices. Firstly, we report about the

three major components – display, network interface and CPU which are most commonly used in applications and accounts for very high power consumption. Then we present two usage patterns of real smart devices that confirm overall high energy dissipation in Android devices. The application Power Tutor is used to determine the results.

A. *Display hardware*

The energy spent in display hardware is sufficiently high and increases as device brightness goes up. Figure 1 lists the energy consumption over a minute in two Android devices with varying brightness level.

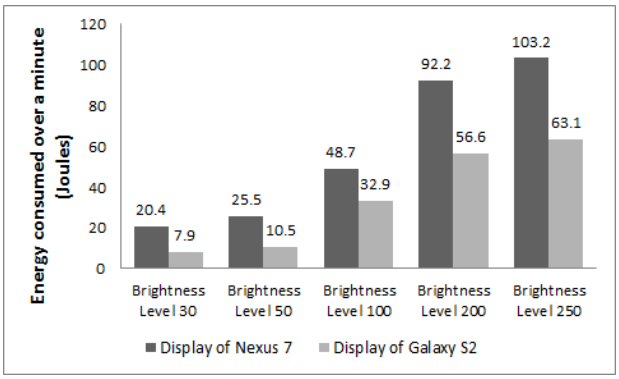

Figure 1. Energy consumption at display hardware.

B. *Network interfaces*

Table I lists the battery consumption in network interfaces in both active and idle mode [5]. Several Android applications depend on network operations. Therefore applications depending on bulk data transfer, audio/video streaming, browsing for long time will dissipate more battery.

TABLE I. BATTERY CONSUMPTION IN NETWEOK INTERFACES

Network Interface	Battery Consumption (mA)	
	Active mode	***Idle mode***
EDGE	300	5
3G	225	2.5 – 3
Wi-Fi	330	12 - 15

C. *CPU*

Existing research suggest that a CPU operating with higher frequency draws more battery. Application executing complex algorithms will consume more CPU cycles and force the operating frequency towards the higher range, thus minimizing the battery life.

Other hardware components and software implementations can also drive the power consumption. A complete overview is presented in [3].

D. *Usage Patterns from Power Monitor*

Table II presents two usage patterns obtained using "Power Monitor" from end users.

TABLE II. USAGE PATTERNS WITH HIGH POWER CONSUMPTION

Device Feature	Usage Patterns	
	User 1	***User 2***
Device description	Samsung GT-P3100, Android 4.1.1 and CPU 1MHz	LG-E400, Android 2.3.6, CPU 800KHz
Brightness level	107.1	144.0
Mobile data	On 24 hours and total usage ~ 100 MB	On 24 hours and total usage ~ 44 MB
Wi-Fi	Not used	Not used
GPS	Used for 35 – 40 minutes	Used for ~60 minutes
Frequently used applications	Facebook, Gmail, Talk, Chrome, YouTube, MyFiles	Browser, Contacts, Facebook, Wikipedia, Temple run
CPU load	Greater than 70% most of the time	Load is moderate i.e. 22% – 30%
Average battery life	7 - 8 hours	12 – 14 hours

As seen from the above table, the applications used by the users are network-centric, computationally expensive and device brightness is very high. As a result of prolong use of power hungry hardware the battery lasts for short duration.

III. SELF-ADAPTIVE FRAMEWORK

In this section we describe the self-adaptive framework to develop Android application that reacts to battery level, status and context. While the choice of battery level and status are obvious, context information is necessary to learn more about the surroundings. For example, while travelling users might want to conserve battery as charging opportunities are not very frequent. Thus combining the battery and context information provides added advantage as the framework can make better decision. Figure 2 depicts the framework and its components.

A. *Battery and Context Monitoring Engine*

A battery and context monitoring engine is employed to extract the remaining battery level, battery status of the device and context information at real time. The context module extracts the system date & time, current coarse location and determines if the user is roaming or travelling abroad. Such information is associated with the battery level and status and stored at the statistics module. This functions as a database which stores the charging and discharging pattern of the mobile device. Using the database, the occurrence of next charging opportunity can be determined.

B. *Analyzer Engine*

The analyzer engine is the heart of the framework. The engine receives the battery and context related information at the real time. The received information are examined at four levels – (i) current battery level, (ii) battery status, (iii) occurrence of next charging opportunity and (iv) travelling information. The engine is configured with predefined rules which take into account combinations of the above four

conditions to determine the appropriate self-adaptive profile. There are three such profiles as mentioned below.

- **Light self-adaptive profile**: This basically contains minimum self-adaptive behavior for the application. The developer can continue to provide rich user experience and it will provide power saving in the application at a minimal level.
- **Medium self-adaptive profile**: This caters to a lot of application features being optimized at hardware, software and user experience level. Our suggestion to the developers is to keep the important functionalities of the application intact and stop the rest. The power saving will consequently be higher than the above profile.
- **Strong self-adaptive profile**: This provides maximum power saving and as a consequence the degree of self-adaption is the highest. Thus only the main core function of the application should be active and additional power saving features should be adopted.

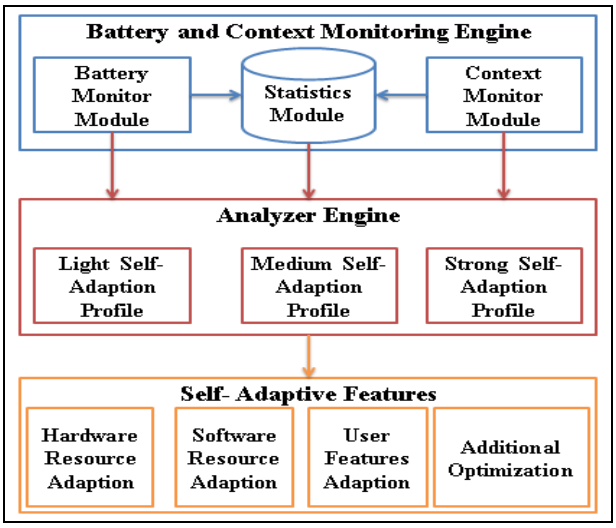

Figure 2. Self-adaptive framework for battery and context aware application development.

All the three profiles contain several self-adaptive features which can be broadly classified into four categories.

- **Hardware resource adaption**: It caters to four main hardware units which consume most of the battery, display, CPU, sensors and network interface. Depending on the profile, various configurations of the hardware resources are to be set. For example, for the strong self-adaption profile (i) the brightness level should be set at the minimum to reduce the power dissipation at display and (ii) turn off GPS and the network operations.
- **Software resource adaption**: Such features include how functionalities developed at the software level should be optimized. For example, location can be determined from both GPS and network technologies. As a measure of such adoption, only network technologies may be preferred to obtain the location. It saves quite amount of power but the trade-off is accuracy of the location. Another example might be not showing in-app advertisements which the battery is critically low.
- **User features adaption**: Each application is composed of one primary and several secondary features. Adapting user features for different profiles will results in providing a subset of secondary features in low battery and only the primary feature when battery is critically low. Also the user experience will undergo various changes depending on which profile is active.
- **Additional optimization**: The developers are advised to further optimize the application for medium and strong self-adaption profiles. Such measures will ensure the application is consuming less power while user experience will remain intact. This basically deals with optimization in the background processes, distributing CPU intensive tasks etc.

The detail features of self-adaption will depend on the actual application and the developer.

IV. Prototype Implementation and Results

To validate the proposed self-adaptive framework of battery and context aware design and related power saving strategies, we have implemented two applications for Android platform. One of the applications does not follow the proposed strategies and the other follows them. The functionalities of the applications are as described below.

i. Display several texts in the screen with default brightness level being 200.

ii. Stream videos from a server and enabling background services to download and upload images, video and text files to a server. These are the primary functionalities of the application.

iii. Display location in a map using both GPS and network based locations to compare and determine the best location. This is a secondary feature in the application.

iv. Show in-app advertisements.

The development of the first application without any self-adaption is straight forward. We focus on describing the self-adaptive Android application development. The self-adaptive framework is explained in details as applicable for the particular application development.

A. *Monitoring Engine*

The monitoring engine is implemented using a service which runs in the background without any user interface. In order to get the battery related information (level and status), the "BATTERY_STAT" permission is required. The context module determines the coarse location from the network connection. The necessary permissions needed are to access

network and coarse location. This module also employs a database which stores the battery level, status, system date & time and location. The database entry is done when there is 1% change (increase or decrease) in battery level. The database forms a part of the battery statistics module which helps in determining discharging and charging pattern of the device. This in turn computes the next available charging opportunity based on the day, time and location.

B. Analyzer Engine

The analyzer engine is also developed using another service and is in-charge of examining the data coming from battery module and context module at real time. There are four levels of examination done (as shown below) in the engine to determine the most appropriate self-adaptive profile for the application.

- Battery level (B)
- Battery status (S)
- Time for next available charging opportunity (C)
- Location information (T)

The battery statistics module is necessary to determine the next charging opportunity. The analyzer engine compares the current time with the time when charging was initiated by the user in recent past to determine the time necessary before the device can be charged. The location information is necessary to determine if the user travelling or not.

Based on above inputs, the engine applies the following conditions to trigger the self-adaptive profiles.

1) Conditions for No Self-Adaptive Profile

When the battery level is sufficiently high, the application does not need any self-adaptive modifications. The following condition must be satisfied for this:

- Battery level lies within 76-100 and battery status can be either discharging or charging and irrespective of the occurrence of charging opportunity and location information.

2) Conditions for Light Self-Adaption Profile

This profile is trigger at the following conditions:

- Battery level is between 51 and 75 and battery is discharging and irrespective of next charging opportunity or location information.
- Battery level is between 11 and 50 and battery is charging from AC source and irrespective of next charging opportunity or location information.
- Battery level is between 11 and 50 and battery is discharging and next charging opportunity will occur within 11 to 30 minutes and user is not roaming or travelling abroad (as obtained from location information).

3) Conditions for Medium Self-Adaption Profile

This profile is activated if any of the following conditions is true:

- Battery level is within 10 to 50 and battery is either discharging or USB charging and next charging time will occur beyond 31 minutes and user is not roaming or travelling abroad (as obtained from location information).
- Battery level is within 1-10 and the device can be plugged to charger within 10 minutes and irrespective of battery status and location.
- Battery level is within 51-75 and the user is travelling abroad or roaming and irrespective of battery status.

4) Conditions for Strong Self-Adaption Profile

- Battery level is less than 10 and next charging opportunity occurs beyond 31 minutes and irrespective of status of battery.
- Battery level is within 11-50 and the user is travelling or roaming.

Once the analyzer engine decides the necessary profile to activate, the application modifies its behavior according to the configuration of the profile. It should be noted that the ranges of the battery level and occurrence of next charging time depends on the developers. The values mentioned here are used for this particular prototype and may be different for another application.

C. Self-Adaptive Features

Following are the descriptions of the self-adaptive behavioral features configured for the particular application. When no profile is active the functionalities of the two applications will be the same.

1) Light Self-Adaption Profile

- The brightness is set to 125.
- For bulk-data transfer, compress the data before uploading to a server. The server must decompress the data after receiving it.
- Check for high speed network before streaming video contents or transferring bulk data.
- If highly accurate location is not needed then use coarse location information for the mapping functionality.

2) Medium Self-Adaption Profile

- Tone down brightness to 75. This degrades the user experience to some extent if the user prefers higher brightness level.
- Network operations are preferred to be done on high speed networks like Wi-Fi/3G. If none of them are available and EDGE has to be used, notify the user about the same and ask if the user would like to continue to use EDGE.
- Transferring bulk data over EDGE should be avoided. This is a user feature adaption in the application.
- Establishing multiple connections to stream HD videos or downloading high volume data should be avoided.
- For scheduling background updates, inexact timer must be used. In this case, Android system will couple several

network operations together to conserve battery and bandwidth.

- Use of GPS should be avoided unless very precise location is needed.
- Applications must use stream parser over tree parser to reduce parsing time and have faster UI response.
- The background processes should be made as short-lived as possible. The processes should sleep between successive calls to reduce memory footprint and CPU usage. AlarmManager API should be used to call the processes at definite interval of time.
- Attempts should be made to distribute CPU intensive tasks and avoid complex computation to reduce CPU power expenditure.

3) Storng Self-Adaption Profile

- Reduce brightness to 30 which minimizes the power consumption at display.
- Defer network operations from the application and notify the user about the same. This includes taking backup in remote server, streaming audio or video files and downloading bulk data. This is another user feature adaption done to make the dying battery last longer.
- By default the mapping feature is not offered in this profile as network operations are turned off. But there is an option where the user can override the settings and use only wireless network location to get the location.
- Stop background updates or any less important services.
- Stop computationally expensive parts of the applications.
- Do not show in-app advertisements.
- Avoid using sensors if possible. For example, do not change the view based on rotation of the screen.
- For video playback from the memory, reduce the size of viewing window. For audio playback, reduce the volume.

D. Results

Figure 3 compares the power consumption of the two applications, one without any self-adaption and another with the self-adaptive features.

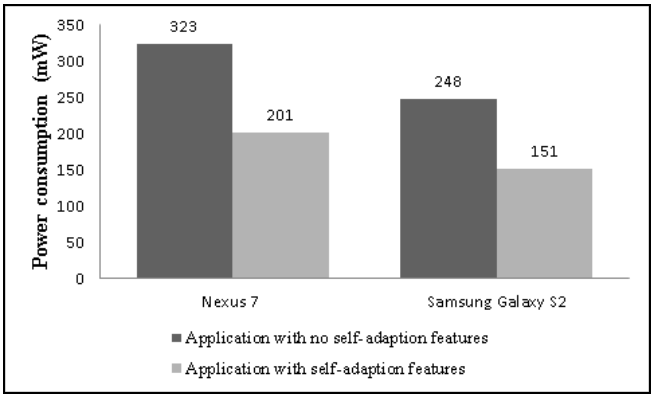

Figure 3. Comparison of power consumption between the two applications.

To further examine the power saving of the three individual profiles, we activated each of them while deactivating two others and compared the resulting power consumption values. The results are portrayed in Figures 4, 5 and 6.

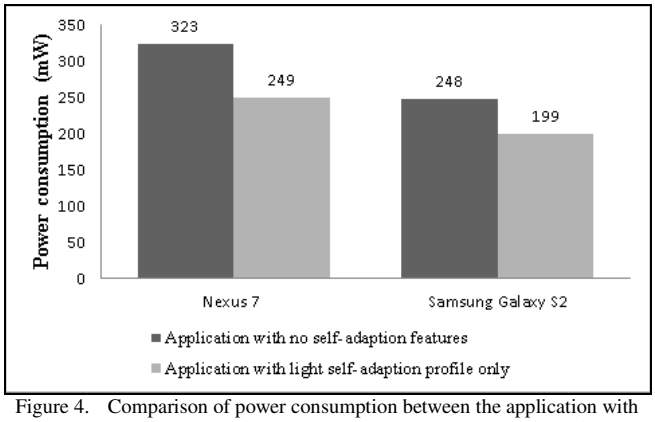

Figure 4. Comparison of power consumption between the application with no self-adaption and application with only light self-adaption profile.

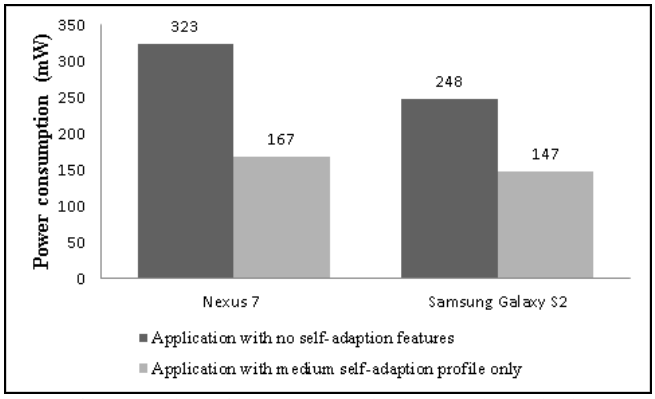

Figure 5. Comparison of power consumption between the application with no self-adaption and application with only medium self-adaption profile.

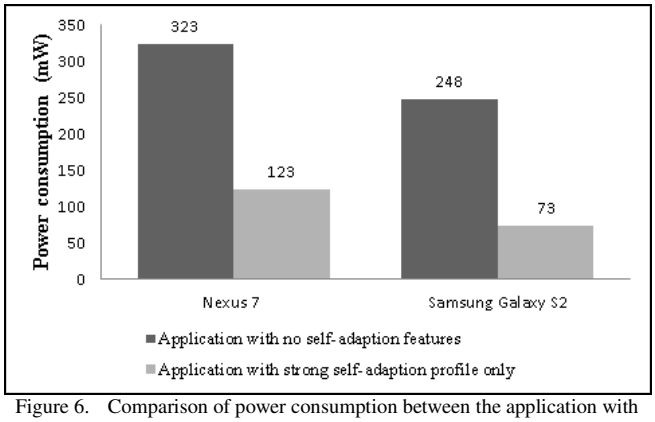

Figure 6. Comparison of power consumption between the application with no self-adaption and application with only strong self-adaption profile.

V. State-of-the-Art

In this section, we review the current literature in connection with our research and compare the strategies.

Context aware computing for mobile devices has received a lot of attention from researchers. Authors Moghimi et al [6] described context awareness in relation to mobile power management and have showcased reduced energy expenditure for periodic and streaming applications. Context aware battery management (CABMAN) is proposed in [7] which can predict the next charging opportunity for mobile devices. The mentioned architecture argues that location information can be used in order to predict the next opportunity to charge mobile devices. Authors Zhao et al [8] presented a system with context-aware approach for predicting battery lifetime. The solution is based on collection of context information from the mobile device (HTC G1). Then a quantitative relation is established between the information collected and battery discharge rate. Preuveneers and Berbers have presented a context-driven and resource-aware middleware for mobile applications. The middleware architecture is composed of three layers as application layer, context layer and runtime layer [9]. The applications are capable of behavioral and structural self-adaption in each layer. Then they mention resource driven component selection algorithm.

On the other hand, self-adaptive mobile application development is studied in depth in [10]. The authors of [11] explore the possibilities of power saving in mobile applications. Their work explains the static behavior of mobile applications which continue to consume power even if the battery level is very low. They introduce an architecture which extracts battery information to process that in battery-awareness integrator to dynamically modify application functionalities. The dynamic self-adaption of the applications are clustered into three layers: (i) External users feature availability - which defines the availability of application features based on the remaining battery level and priority is given to core algorithm of the applications, (ii) Internal feature behavior adaption – which provides several power efficient implementation of the same functionalities and shall be triggered based when battery level reaches predefined levels and (iii) Data consumption adaption – which explores the power efficient communication with an external server. But the architecture does not take into consideration the battery charging status and context information. Also the work does not provide guidelines to implement power efficient implementations.

In our research, we have taken into account battery level, status and context in order to decide a self-adaption profile. Furthermore, the analyzer engine is intelligent enough to compute the next available charging opportunity and travelling information. Thus the self-adaptive features are not solely determined based on battery information, rather more conditions are checked to accurately determine which self-adaptive profile to be triggered. This is the main innovative aspect of the proposed framework.

VI. CONCLUSION

In this paper we have presented a self-adaptive framework for Android application development. The primary goal of the framework is to monitor the battery level, status and context information at real time, analyze them and trigger the appropriate self-adaptive profile. The higher power consumption of applications in smart devices motivates us to develop such framework. A prototype application has been developed using the approach. The results clearly establish that such development strategies can immensely benefit the smart devices and applications can become power efficient.

ACKNOWLEDGMENT

The work is sponsored by French research project Smart-4G-Tablet Pole SCS. The authors thank the Android application beta testers for helping to evaluate the framework.

REFERENCES

[1] A. Pathak, Y. C. Hu and M. Zhang. “Where is the energy spent inside my app? Fine grained energy accounting on smartphones with eprof.” In Proc. of ACM EruoSys’12, Bern, Switzerland, 2012.

[2] L. Zhang, et al. “Accurate online power estimation and automatic battery behavior based power model generation for smartphones.” In Proc. Of ACM CODES+ISSS’10, Arizona, USA, 2010, pp. 105-114.

[3] Datta, S.K.; Bonnet, C.; Nikaein, N., "Minimizing energy expenditure in smart devices," IEEE Conference on Information and Communication Technologies (ICT 2013), Tamil Nadu, India, April, 2013.

[4] Datta, S.K.; Bonnet, C.; Nikaein, N.; , "Android power management: Current and future trends," First IEEE Workshop on Enabling Technologies for Smartphone and Internet of Things (ETSIoT), 2012, pp.48-53.

[5] http://www.google.com/events/io/2009/sessions/CodingLifeBatteryLife.html.

[6] Mohammad Moghimi, Jagannathan Venkatesh, Piero Zappi, Tajana Rosing. “Context-Aware Mobile Power Management Using Fuzzy Inference as a Service”, In 4th International Conference on Mobile Computing, Applications and Services (MobiCASE) 2012, pp. 314-327.

[7] Ravi, N.; Scott, J.; Lu Han; Iftode, L., "Context-aware Battery Management for Mobile Phones," *Pervasive Computing and Communications, 2008. PerCom 2008. Sixth Annual IEEE International Conference on* , vol., no., pp.224,233, 17-21 March 2008.

[8] Xia Zhao, Yao Guo, Qing Feng, and Xiangqun Chen. 2011. A system context-aware approach for battery lifetime prediction in smart phones. In *Proceedings of the 2011 ACM Symposium on Applied Computing* (SAC '11). ACM, New York, NY, USA, 641-646.

[9] Davy Preuveneers and Yolande Berbers. 2007. Towards context-aware and resource-driven self-adaptation for mobile handheld applications. In *Proceedings of the 2007 ACM symposium on Applied computing* (SAC '07). ACM, New York, NY, USA, pp. 1165-1170.

[10] Nearchos Paspallis, Frank Eliassen, Svein Hallsteinsen, and George A. Papadopoulos. “Developing Self-Adaptive Mobile Applications and Services with Separation-of-Concerns” In “At Your Service:Service-Oriented Computing from an EU Perspective”, MIT Press, Edition 1, pp. 129 – 158, 2009.

[11] Mizouni, R.; Serhani, M.A.; Benharref, A.; Al-Abassi, O., "Towards Battery-Aware Self-Adaptive Mobile Applications," IEEE Ninth International Conference on Services Computing (SCC), 2012, pp.439 - 445, 24-29 June 2012.