Energy-Aware Design Patterns for Mobile Application Development (Invited Talk)

Abhijeet Banerjee
National University of Singapore, Singapore
abhijeet@comp.nus.edu.sg

Abhik Roychoudhury
National University of Singapore, Singapore
abhik@comp.nus.edu.sg

ABSTRACT

Developing energy-efficient application is crucial for mobile platforms such as smartphone and tablets, since such devices operate on a limited amount of battery power. However, until recently most of the smartphone applications have been developed in an energy-oblivious fashion. This is increasingly becoming a concern due to the fact that smartphone applications are progressively becoming complex and energy-intensive, whereas the battery technology is unable to keep up. Existing studies have proposed a number of testing and re-factoring techniques that can be used to increase the energy-efficiency of such applications, after the development has been completed. However, we feel that maximum level of energy-efficiency can be achieved only if energy-efficient design practices are used in the software development process.

In this study, we propose a set of energy-aware design patterns, specifically targeted at smartphone applications. These design patterns can be applied to huge number of real-life scenarios for energy-efficient information gathering and processing, within the smartphone application. We also present some examples of design patterns for application development for the Android platform.

Categories and Subject Descriptors: D.2.2 [Software Engineering]: Design Tools and Techniques

Keywords: Mobile Apps; Energy-Aware Design Patterns

1. INTRODUCTION

A recent report from a technology research and advisory firm estimated that the total revenues from mobile apps grew from $18 Billion in 2012 to $26 Billion in 2013 [1]. The report also observed that in 2013 the number of application downloads increased to 102 Billion. These numbers indicate that the market for mobile application is growing at a very rapid rate. Such growth presents opportunities as well as challenges to mobile application developers. To excel in a such a competitive market, application developer must ensure that their applications are of better quality. Quality of an application can be judged based on a number of factors such as functionality, performance, energy-efficiency, etc. We feel that energy-efficiency is a crucial factor for mobile application development because mobile devices (like smartphones) operate on a limited amount of battery power. However, until recently most of the mobile applications have been developed in an energy-oblivious fashion. Exiting studies [2] have uncovered the presence of energy-inefficiencies in a number of real-life smartphone applications. Some of the more recent works [3] have proposed energy-aware testing frameworks to uncover energy-inefficiencies in mobile applications. In this study we propose a four-pronged approach for energy-aware mobile application development. Our approach consists of four design patterns (Figure 1) (i) Energy-aware design practices for energy-heavy hardware components (ii) Optimal design choices for information gathering and transmission (iii) Trade-offs between energy consumption and Quality-of-Service and (iv) Increasing energy efficiency by using asynchronous tasks.

DeMobile'14, November 17, 2014, Hong Kong, China
ACM 978-1-4503-3225-5/14/11
http://dx.doi.org/10.1145/2661694.2661698

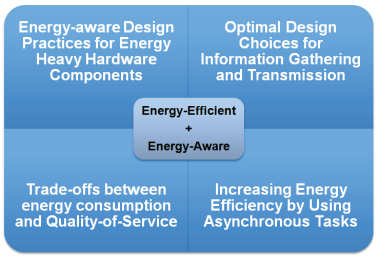

Figure 1: Design Patterns for Energy-Aware Application Development

1.1 Energy-Aware Design Practices for Energy Heavy Hardware Components

Existing studies [2] have suggested that use of I/O components and power management utilities play an important role in influencing the energy consumption behaviour of a smartphone application. Experiments in our previous work [3] have lead us to believe that not only do such components/utilities play an important role in influencing the energy consumption behaviour of the device but their misuse can lead to serious energy-inefficiencies. It is worthwhile to know that smartphone operating systems (like Android) require an application to acquire a hardware resource (such as sensor, GPS receivers, camera, Wifi) before it can start using the resource. Often application developers would acquire the required resource much before the information gathered from that hardware resource is needed. This causes excessive but avoidable power consumption. For instance, in Android to use the camera, an application must first acquire it and only then it can start the video preview. Interestingly, we observed that while the camera was acquired (but the preview not started) the additional power consumption was approximately 47% more than the idle power consumption whereas starting the preview increases the power consumption by an additional 10% (to a total of 57%) only. Energy-inefficiencies can also manifest as a result of misuse of power management utilities. For instance, Android provides a set of power management utilities, such as *Wakelocks*, using which the power-state transitions of the device can be controlled. Suboptimal usage of such power management utilities also leads to excessive, avoidable power consumption.

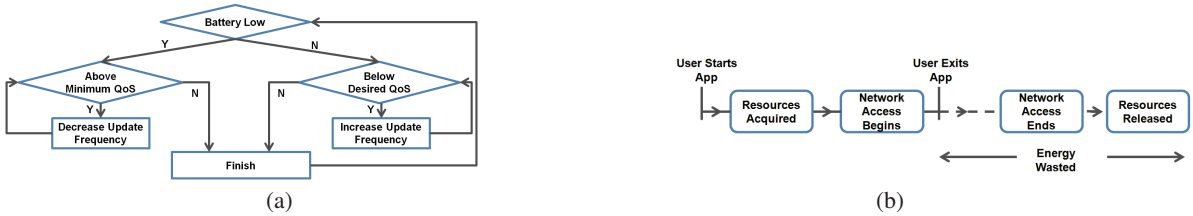

Figure 2: (a) Energy-Aware Information Gathering (b) Energy-inefficiency Due to Asynchronous Power Loss

1.2 Optimal Design Choices for Information Gathering and Transmission

In context of mobile applications, certain kind of information can be obtained/transmitted over multiple hardware components. For instance, modern smartphones have multiple network components, such as Wifi, 3G and GSM. The decision to choose a network components should be based on a number of factors such as transmission size, frequency of transmission, delay tolerance, etc. Existing studies [4] have observed that for smaller transmissions Wifi is less energy efficient than GSM because the energy consumed for establishing the connection in Wifi is often higher than that of GSM. However, energy efficiency of Wifi increases dramatically with bigger transmission sizes. Comparatively, the cost of establishing the connection in 3G is low, however 3G transmission has a high overhead due to *Tail Energy* [4]. In general, Wifi is energy-efficient for large transfer sizes, 3G is efficient for small transfer sizes as long the transfers take place within the *Tail Energy* duration and GSM is more energy-efficient than 3G but offers a lower data transfer rate. Another such scenario where an applications can choose between hardware components is in location estimation. The location can be either be estimated through GPS based locations updates or through Wifi based locations updates. GPS based updates are more precise than Wifi based updates but consumes more power than the later. Such design choices heavily depend on the application scenario and must be made by the developer *before* the development of the application begins. Additionally, since such application specific requirements are seldom explicitly written in the application code, therefore suboptimal energy behaviour due to such design choices are very hard to detect using automated testing techniques only.

1.3 Trade-off between Energy-Consumption and Quality-of-Service

Applications intended for smartphone platforms are often real-time but non-mission-critical in nature. Therefore, subtle deviation in the Quality-of-Service(QoS) are often acceptable. Since most of the energy consumed on mobile devices is spent on information gathering or information processing, there can be two ways in which QoS can be traded for energy-efficiency.

(i) *Frugal information gathering*: As the name suggests, this approach involves collecting as less information as possible to achieve the pre-defined QoS. For instance, in Android, applications can gather information from a variety of sensors (such as accelerometer, gyroscope *etc*) with the varying update rates such as SENSOR_DELAY_FASTEST, SENSOR_DELAY_GAME, SENSOR_DELAY_NORMAL, SENSOR_DELAY_UI. Faster updates rates, as expected, provide more precise information but incurs a heavier energy consumption. Other components such as GPS receivers also allow the developer to specify the minimum frequency of location updates. Such flexibility can be exploited to dynamically adjust the QoS/energy-efficiency of the device. The idea (as shown in 2(a)) is to maintain the desired QoS while minimizing the energy consumption. Another way of frugal information gathering is to share information between multiple components within an applications. For example, many real-life Android applications [3] require location information for the application functionality as well as to show location-based advertisements. In such scenario location information can be shared between both the components, instead of fetching the information twice.

(ii) *Approximated information processing*: Several types of applications such as games, video/audio players, data compression/decompression tools are computationally intensive. Such applications derive much of their functionality from energy-intensive loops and functions. For such cases, applications can be equipped with energy-saving features such as runtime loop perforation and function approximation [5]. The choice of the using a particular energy saving feature should be made at runtime based on the battery status of the device and the desired QoS.

1.4 Increasing Energy Efficiency by Using Asynchronous Tasks

Android applications are executed on a single thread, by default. This means that all the components of the application have to execute on the same thread. This however can create not only performance issues but energy-inefficiencies. For example, (as shown in Figure 2(b)) assume there is an application that executes user-interface (UI) actions, resource acquires/releases as well network operation on the same thread. In such a scenario, if the network operations are delayed, all subsequent operations, including release of resource is delayed thereby causing energy-inefficiency indirectly.

Android provides a mechanism named AsyncTask (asynchronous task), to overcome delay to the UI thread due to time-consuming tasks. Android website explains that "AsyncTask enables proper and easy use of the UI thread." However, one can use AsyncTask to achieve a better UI performance as well as remove energy inefficiencies. This can be achieved by executing all time consuming tasks as AsyncTasks. This frees the main UI thread for user interaction as well as provides the opportunity to release any acquired resources and cancel all AsyncTasks instantly, whenever the user exits the application. For example in the Figure 2(b), if the network access were initiated as an AsyncTask, when the user exits the application the resource can be released instantaneously.

2. ACKNOWLEDGEMENT

The work was partially supported by a Singapore MoE Tier 2 grant MOE2013-T2-1-115 entitled "Energy aware programming".

3. REFERENCES

[1] Gartner. {www.gartner.com/newsroom/id/2592315}.
[2] 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 *EuroSys*, 2012.
[3] A. Banerjee, L. K. Chong, S. Chattopadhyay, and A. Roychoudhury. Detecting energy bugs and hotspots in mobile apps. In *FSE (to appear)*, 2014.
[4] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani. Energy consumption in mobile phones: a measurement study and implications for network applications. In *SIGCOMM*, 2009.
[5] W. Baek and T. M. Chilimbi. Green: a framework for supporting energy-conscious programming using controlled approximation. In *PLDI*, 2010.