ISSN XXXX XXXX © 2016 IJESC

Research Article Volume 7 Issue No.3

GreenDroid: Automated Diagnosis of Energy Inefficiency for SmartPhone Application

Ashwini S.G[1], Sowmyashree .K.M[2]
Student[1], Assistant professor[2]
Department of MCA
PESCE, Mandya, India

Abstract:
Smartphone applications' energy efficiency is vital, but many Android applications suffer from serious energy inefficiency problems. Locating these problems is labour-intensive and automated diagnosis is highly desirable. However, a key challenge is the lack of a decidable criterion that facilitates automated judgment of such energy problems. Our work aims to address this challenge. We conducted an in-depth study of 173 open-source and 229 commercial Android applications, and observed two common causes of energy problems: missing deactivation of sensors or wake locks, and cost-ineffective use of sensory data. With these findings, we propose an automated approach to diagnosing energy problems in Android applications. Our approach explores an application's state space by systematically executing the application using Java PathFinder (JPF). It monitors sensor and wake lock operations to detect missing deactivation of sensors and wake locks. It also tracks the transformation and usage of sensory data and judges whether they are effectively utilized by the application using our state-sensitive data utilization metric. In this way, our approach can generate detailed reports with actionable information to assist developers in validating detected energy problems. We built our approach as a tool, GreenDroid, on top of JPF. Technically, we addressed the challenges of generating user interaction events and scheduling event handlers in extending JPF for analyzing Android applications. We evaluated GreenDroid using 13 real-world popular Android applications. GreenDroid completed energy efficiency diagnosis for these applications in a few minutes. It successfully located real energy problems in these applications, and additionally found new unreported energy problems that were later confirmed by developers.

1. INTRODUCTION

Mobile computing is the discipline for creating an information management platform, which is free from spatial and temporal constraints. The freedom from these constraints allows its users to access and process desired information from anywhere in the space. The state of the user, static or mobile, does not affect the information management capability of the mobile platform. A user can continue to access and manipulate desired data while traveling on plane, in car, on ship, etc. Thus, the discipline creates an illusion that the desired data and sufficient processing power are available on the spot, where as in reality they may be located far away. Otherwise **Mobile computing** is a generic term used to refer to a variety of devices that allow people to access data and information from where ever they are.

Smartphone applications' energy efficiency is vital, but many Android applications suffer from serious energy inefficiency problems. Locating these problems is labor-intensive and automated diagnosis is highly desirable. However, a key challenge is the lack of a decidable criterion that facilitates automated judgment of such energy problems. Our work aims to address this challenge. We conducted an in-depth study of 173 open-source and 229 commercial Android applications, and observed two common causes of energy problems: missing deactivation of sensors or wake locks, and cost-ineffective use of sensory data. With these findings, we propose an automated approach to diagnosing energy problems in Android applications. Our approach explores an application's state space by systematically executing the application using Java PathFinder (JPF). It monitors sensor and wake lock operations to detect missing deactivation of sensors and wake locks.

1.2 Motivation for Project

Our approach explores an application's state space by systematically executing the application using Java PathFinder (JPF). It monitors sensor and wake lock operations to detect missing deactivation of sensors and wake locks. It also tracks the transformation and usage of sensory data and judges whether they are effectively utilized by the application using our state-sensitive data utilization metric. In this way, our approach can generate detailed reports with actionable information to assist developers in validating detected energy problems.

SYSTEM DESIGN

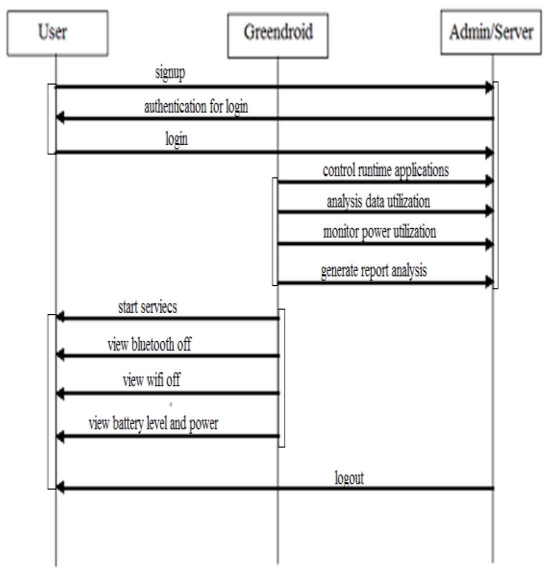

Figure.1. Sequnce Diagram

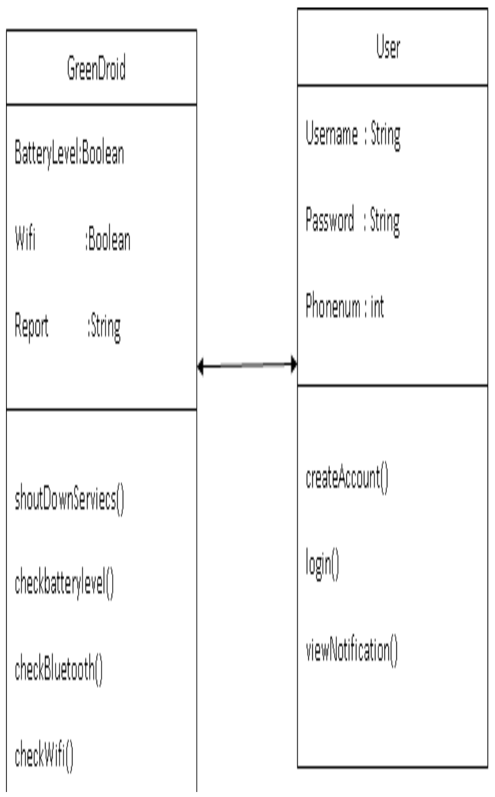

Figure.2. Class diagram

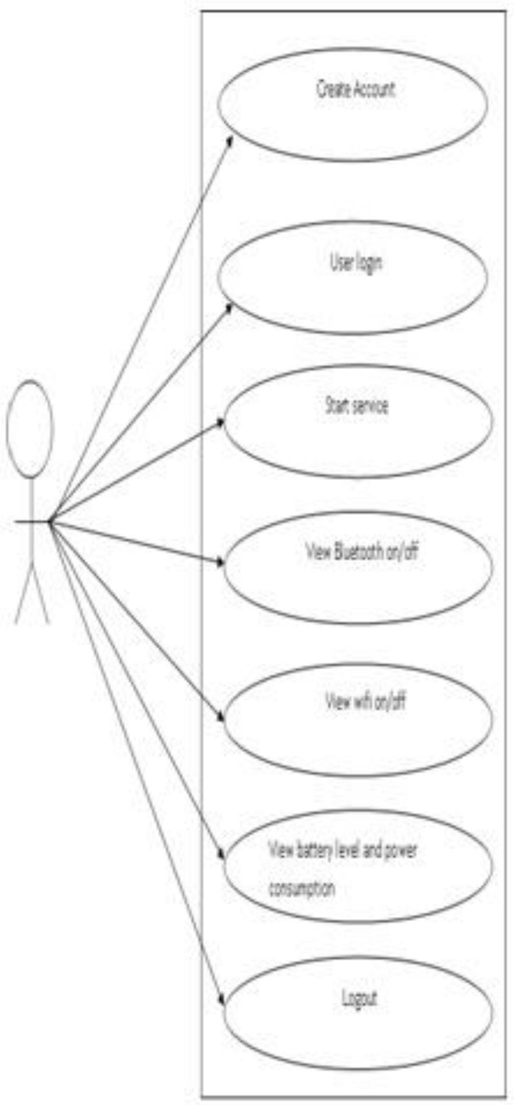

Figure.3. Use Case Diag

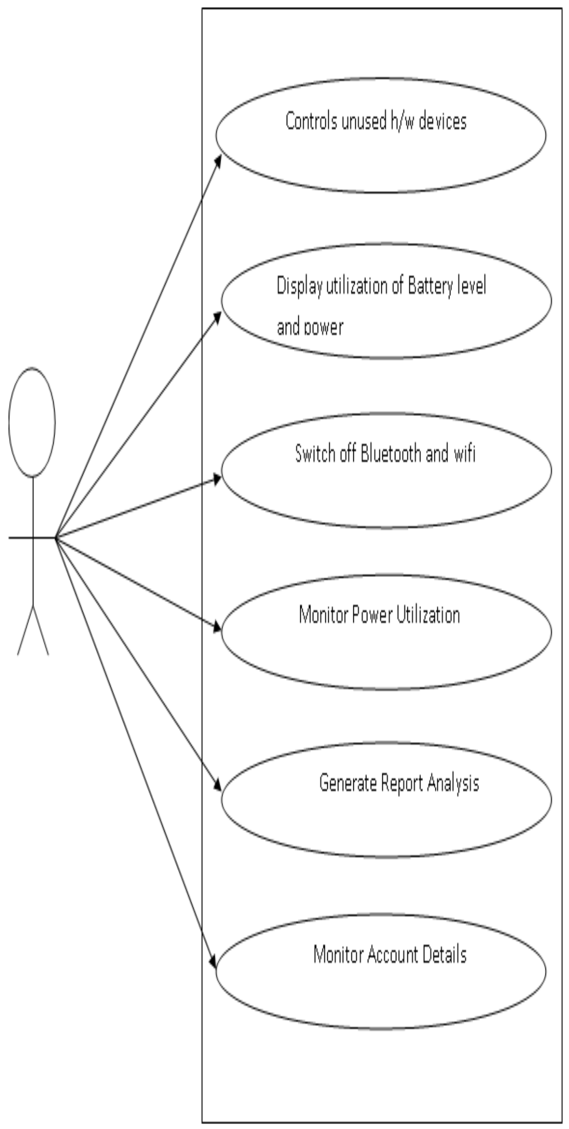
Controls unused h/w devices
Display utilization of Battery level and power
Switch off Bluetooth and wifi
Monitor Power Utilization
Generate Report Analysis
Monitor Account Details

Figure.4. Use Case Diagram for GreenDroid:

II. SYSTEM ANALYSIS

Existing System

Existing studies show that many Android applications are not energy efficient due to two major reasons. First, the Android framework exposes hardware operation APIs (e.g., APIs for controlling screen brightness) to developers. Although these APIs provide flexibility, developers have to be responsible for using them cautiously because hardware misuse could easily lead to unexpectedly large energy waste. Second, Android applications are mostly developed by small teams without dedicated quality assurance efforts. Their developers rarely exercise due diligence in assuring energy savings.

2.3 Proposed System

In this work, we set out to mitigate this difficulty by automating the energy problem diagnosis process. A key research challenge for automation is the lack of a decidable criterion, which allows mechanical judgment of energy inefficiency problems. As such, we started by conducting a large-scale empirical study to understand how energy problems have occurred in real-world smartphone applications. By examining their bug reports, commit logs, bug-fixing patches, patch reviews and release logs, we made an interesting observation: Although the root causes of energy problems can vary with different applications, many of them (over 60%) are closely related to two types of problematic coding phenomena.

They are Missing sensor or wake lock deactivation and Sensory data underutilization.

III. MODULES

1.Application Execution and State Exploration

Android applications are mostly designed to interact with smartphone users. Their executions are often triggered by user interaction events. Typically, an Android application starts with its main activity, and ends after all its components are destroyed. During its execution, the application keeps handling received user interaction events and sys-tem events (e.g., broadcasted events) by calling their handlers according to Android specifications. Each call to an event handler may change the application's state by modifying its components' local or global program data. As such, in order to execute an application and explore its state space in JPF, we need to: (1) generate user interaction events, and (2) guide JPF to schedule corresponding event handlers.

2. Missing Sensor or Wake Lock Deactivation

We next discuss how to detect energy problems when exploring an application's state space. As mentioned earlier, missing sensor or wake lock deactivation is one common cause of energy problems. This shares some similarity with traditional resource leak problems, where a program fails to release its acquired system resources (e.g., memory blocks, file handles, etc.) Resource leak problems can cause system performance degradation (e.g., slower re-sponse), and similarly missing deactivation of sensors or wake locks can also waste valuable battery energy. Besides, according to Android process management policy, senors and wake locks are not automatically deactivated even when the application components that activated them are destroyed (e.g., onDestroy() handler is called). We will give an example and details. Based on the preceding state exploration efforts, we can now adapt existing resource leak detection techniques to detect missing sensor or wake lock deactivation

3.Sensory Data Utilization Analysis

During an Android application's execution, its collected sensory data are transformed into different forms and consumed by different application components. We need to track these data usages for energy efficiency analysis. We do it at the byte code instruction level by dynamic tainting. Our technique contains three phases: (1) tainting each collected sensory datum with a unique mark; (2) propagating taint marks as the application executes; (3) analyzing sensory data utilization at different application states. We elaborate on the three phases in the following.

a.) Preparing and tainting sensory data

In the first phase, we generate mock sensory data from an existing sensory data pool, which is controlled with different precision levels. They are then fed to the application under analysis after each event handler call. The object reference to each sensory datum is initialized with a unique taint mark before the datum is fed to the applica-tion. The taint mark will be propagated with the datum together for later analysis.

b.) Propagating taint marks

At runtime, an Android application's collected sensory data are transformed into different forms by assignment, arithmetic, relational, and logical operations. For example, the Osmdroid application in has its loc object (Line 38) transformed to another formatted Loc object (Line 39), which further affects the intent object (Line 42). Later, by message communication, this intent object is propagated to a broadcast receiver and converted back to the loc object (Line 9), which may or may not affect database con-tent, depending on the variable tracking ModeOn's value (Line 11). Such data flows need to be tracked to propagate taint marks so as to identify which program data depend on the collected sensory data. Based on this information, one is then able to analyze sensory data utilization.

c.)Analyzing sensory data utilization

With program data tainted with marks associated with sensory data, we can analyze how sensory data are used in an Android application and whether the uses are effective with respect to energy cost.

5.4 Usecase Diagram

A use case diagram is a simple technique of an internal representation of a user's interaction with the system and distracting the specifications of a use case. A use case diagram can be shows the different types of users of that can be help system and the different ways that they are overcome between with the system. This type of diagram is can be used in different with the code use case and often be published by other types of sketch as well it can be designed. Other a use case can be itself might roll into a lot of reservation about every system with some specification; a use-case diagram can help to provide a higher-level show of the system. It has been says some that "Use case diagrams which it shows always the blueprints for the system". They give the simple and sketch representation of how the system do it and behave different use case which shows the actual flowing of the procedure form data flowing.

IV. CONCLUSIONS

In this project, we presented an empirical study of real energy problems in Android applications, and identified two types of coding phenomena that commonly cause energy waste: missing sensor or wake lock deactivation, and sensory data underutilization. Based on these findings, we proposed an approach for automated energy problem diagnosis in Android applications. Our approach systematically explores an application's state space, automatically analyzes its sensory data utilization, and monitors the usage of sensors and wake locks. It helps developers locate energy problems in their applications and generates actionable reports, which can greatly ease the task of reproducing energy problems as well as fixing them for energy optimization. We implemented our approach into a tool GreenDroid on top of JPF, and evaluated it using 13 real-world popular Android applications. Our experimental results confirmed the effectiveness and practical usefulness of GreenDroid.

V. REFERENCES

[1]. Beginning Android Programming by Haseman.

[2]. Android Application Development by James C. Sheusl.

[3]. Programming Android by Zigurd Mednieks, Laird Dornim, G. Blake Meike, Masumi Nakamura.

[4]. Android Programming by Bill Phillips, Brian Hardy.

[5]. The Complete Reference JAVA by Herbert Schildt.

[6]. Ian Sommerville "Software Engineering" Person Education Ltd, 9th Edition

[7]. MichaelBlaha,James Rumbaugh "OOMD" Person Education Ltd, 2nd Edition

[8]. Elmasri and Navathe "Fundamentals of Database System" Person Education Ltd, 5h Edition

[9]. Mauro Pezze,Michal Young "Software Testing" John Willy and Sons Pte. Ltd Edition

[10]. http://developer.android.com/guide/index.html

[11]. http://developer.android.com/training/basics /firstapp/ index.html

[12]. http://www.codeproject.com/Articles/102065/Android-A-beginner-s-guide

[13]. http://mobile.dzone.com/articles/fundamentals-android-tutorial