A Visualization Method of Inter-module Communications for Profiling Energy Consumption of Android Applications

Hiroki Furusho[1], Kenji Hisazumi[2], Takeshi Kamiyama[3], Hiroshi Inamura[3], Shigemi Ishida[1] and Akira Fukuda[1]

[1]Graduate School/Faculty of Information Science and Electrical Engineering, Kyushu Univ., Fukuoka, Fukuoka, Japan

[2]System LSI Research Center, Kyushu Univ., Fukuoka, Fukuoka, Japan

[3]Research Laboratories, NTT DOCOMO INC., Yokosuka, Kanagawa, Japan

Abstract— *We propose a method for visualizing the relationship between software modules of applications running on the Android OS. Existing energy estimation methods can analyze energy consumption for each modules of an application. However, it is difficult for application developers to choose a module as tuning target by the above profiling result.*

Our proposed method observes data modules communicating each other, and visualizes the relationship between a large energy-consuming module and other modules. In this study, we analyzed a verification application with proposed method and showed the relationship between these application modules.

Keywords: Energy consumption, Profiling method, Mobile application, Android

1. Introduction

An important task for Android application developers is reducing the energy consumed by their Android app in order to prolong the battery life of the Android smartphones. Although this problem can be addressed at both hardware and software levels, it is important to reduce the energy consumption of individual applications that vary significantly in behavior. Considering the foregoing, it is not sufficient to identify modules to be tuned by individually visualizing energy consumption for each module.

The simplest method for reducing the energy consumption of smartphone applications is to eliminate problems such as excessive creation instances, loop statement errors, communication process errors, and bugs. Energy-profiling methods can identify the points at which the applications consume excessive energy and determine methods to reduce their overall consumption. Existing methods can estimate energy consumption of the entire smartphone by using data obtained from the OS (such as CPU time, amount of file system access, and traffic). The authors have proposed a profiling method that analyzes energy consumption of modules of an application running on the Android OS [1].

However, a module that must be tuned might be different from modules that consume energy excessively. For instance, when module A consumes a large amount of energy, there are two possible causes: module A itself consumes the energy, or other modules use module A excessively. In the latter case, it is not enough to visualize energy consumption for each module individually to identify modules to tune.

Hence, we propose a method to visualize relationships between modules to assist developers in determining modules that must be tuned. The proposed method is applied to a simple application for testing.

The remainder of the paper is organized as follows. Section **2** describes related work. Section **3** explains about the type of communication monitored by our proposed method and specifies the logging process. Experimental results are reported in Section **4**. Section **5** concludes with a summary and specifies the direction of our future works.

2. Related Work

Most smartphone energy analysis methods employ model-based estimation. The basic form of the energy-consumption model can be represented by the following linear equation

$$E_{estimate} = \sum_{m \in M} C_m \cdot V_m \tag{1}$$

$E_{estimate}$, M, C_m and V_m represent estimated energy, a set of the factors related to energy consumption (such as CPU time, data communication, access to storage and display), usage of a factor $m \in M$, and its coefficient, respectively. C_m is calculated by regression analysis of resource usage and measured energy consumption.

Several researchers present methods that use a energy model to estimate the energy consumption of the entire device, using operating times of each part of the device as parameters [2][3]. However, it is difficult to identify the contribution of an application to the total energy consumption because smartphones can run several processes simultaneously.

An estimation method using values obtained from a Linux process file system [4] overcomes the issue [5] because the process file system records the device usage (hereinafter referred to as "resource usage") for each process separately. Mittal et al. proposed an energy consumption profiling

method for the CPU, wireless communication (3G, Wi-Fi) and display[6]. The display energy is consumed by the application because the interface of the application itself usually occupies the smartphone display.

The authors have proposed profiling method that analyzes energy consumption of modules of an application running on the Android OS[1]. However, a module that must be tuned might be different from modules that consume energy excessively as mentioned above.

3. Profiling of relationships between modules

3.1 Overview

This section describes the monitoring process of inter-module communication. Our method provides application developers with profiling result based on the actual usage of users. Regardless of a developer's understanding of an application, the profiling results are helpful.

Our proposed method monitors behavior of an application and records the behavior in the log file. To hook various method in an application, we implemented the logging code using AspectJ[7]. Fig. 1 depicts code weaving using AspectJ. AspectJ includes an additional object named Aspect, which is not a part of Java. The Aspect object contains the conditions of the embedding point (pointcuts) in the source code and the embedding codes (advice). When an application is generated, Java bytecode, which generated from the advice,is embedded into the application.

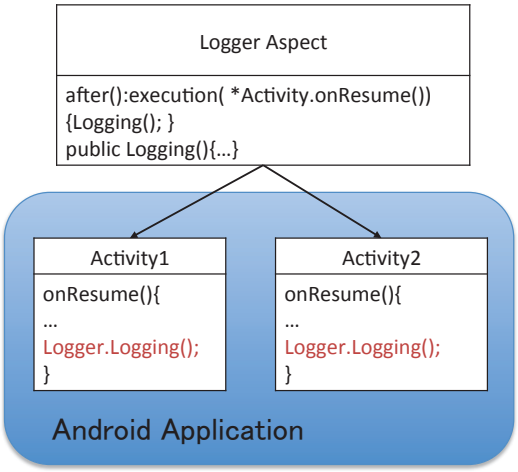

Fig. 1: Code embedding using AspectJ.

3.2 Inter-module communication of an Android application

We classify communication type of Android application modules. Table 1 shows monitoring target modules and calling patterns of modules. The target modules are `Activity` class, `Service` class, `BroadcastReceiver` class and `AlarmManager` classes. Activity provides the GUI for the functions of an application. Service runs longer than the Activity class and performs background processing. These application components call each other's method with data called `Intent`. `Broadcast Intent` is a kind of Intent. Broadcast Intent is sent to all of modules that have a particular attribute value. A BroadcastReceiver can receives a Broadcast intent. AlarmManager sends an Intent in a constant cycle.

Table 1: Calling pattern of Android module.

Activity class and Service class
• Normal calling
• Calling via AlarmManager
BroadcastReceiver class
• Calling from Android application
• Calling from Android system

3.2.1 Normal invocation

Fig. 2 shows the logging process of normal invocation of an Activity. This case occurs when switching screen of an application or using a function of Service. In Android system, we send an Intent to invoke the Activity from other Service or Activity. Our proposed method records communication log between sender module(Activity1) and receiver module(Activity2). When an Intent instance is created, our proposed method gives a hash value to the Intent, which enables us to trace a sender from a receiver. After that, when Intent is sent by `startActivity()` ,`startService()` and `bindService()`, the sender module's name and hash value are recorded in the log file. The receiver module receives the Intent and starts processing. Our proposed method obtains the receiver module's name and the hash value given by sender and records them.

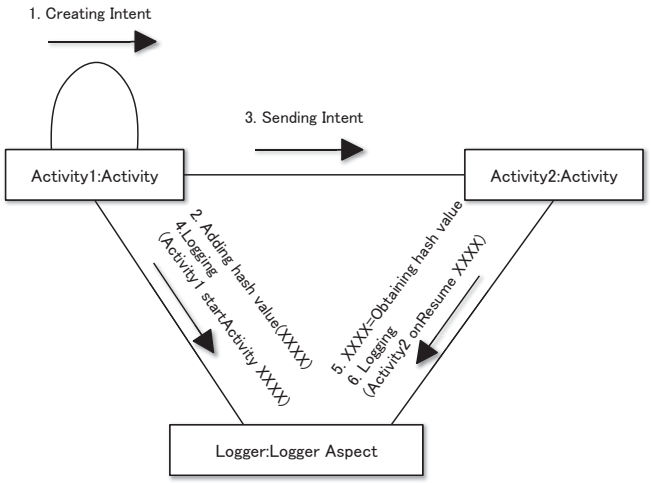

Fig. 2: Log collection of a normal invocation.

3.2.2 Periodic invocation

This section describes how to determine the periodic invocation of services. AlarmManager sends Intents to Services according to settings in advance. For energy saving, it is important to identify which service received the intents from AlarmManager and the number of times they were received. Fig. 3 depicts a method to log such Intents from the AlarmManager to Services. Our proposed method also gives a hash value to the Intent as in the previous section. Our method also gives a hash value to the Intent to trace the sender and receiver in the same way as described in the previous section. The Intent is used to create an instance of a Pending Intent. Unlike in the Activity invocation case, we cannot see the hash value of the sent Intent because we cannot obtain the Intent instance from the `Pending Intent`. To check the hash value of the sent Intent, our proposed method collects the ID numbers of the Pending Intents when creating and sending a Pending Intent. The ID number of Pending Intent is a return value of java.lang.Object.hashCode()[8]. The receiver can log the same way that we mentioned in Section 3.2.1.

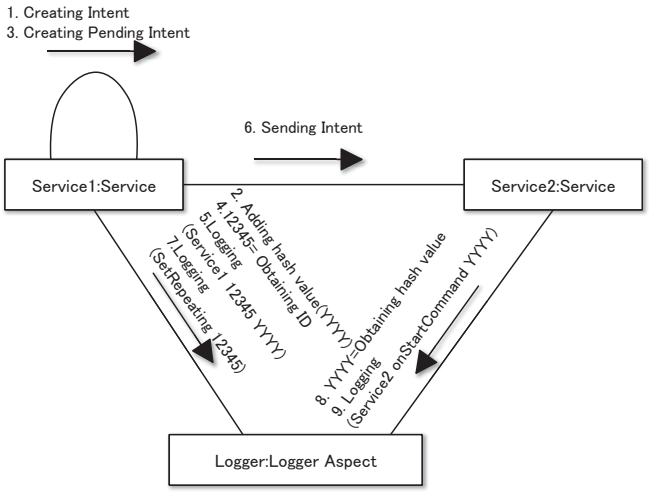

Fig. 3: Log collection of a periodic invocation.

3.2.3 Invocation by Broadcast Intents

The Android system or applications send Broadcast Intents to applications to notify certain events. The application can receive Broadcast Intent to implement BroadcastReceiver and specify a type of Intent that the application wants to receive. The analysis method of this type of communications determines whether frequency and type of Broadcast Intent are appropriate.

Fig. 4 shows how to log for inter-application communication. The logger gives a hash value to an Intent and logs it as a sender, as mentioned in Section 3.2.1. The Android system calls the onReceive() method implemented in BroadcastReceiver whenever the Broadcast Intent is sent. We can hook the onReceive() method to identify the Intent. Fig. 5 shows how to log for receiving a Broadcast Intent from the Android system. We can annotate an attribute called "action" to an Intent. We can distinguish whether the Broadcast Intent is sent from the Android system because a Broadcast Intent sent by the Android system has a value that begins with a particular string, such as "android.action".

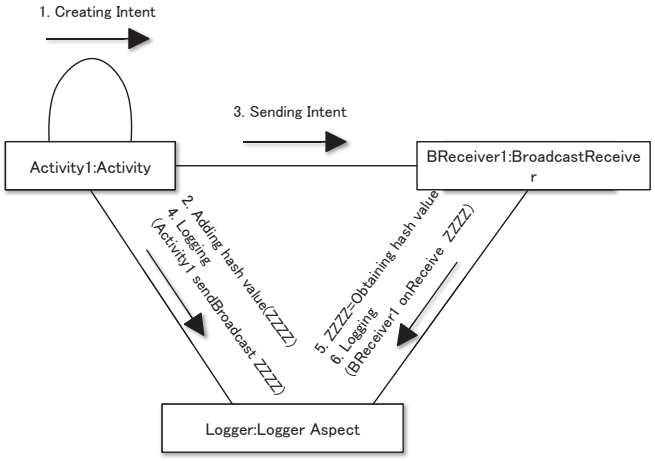

Fig. 4: Log collection of inter-application Broadcast Intent.

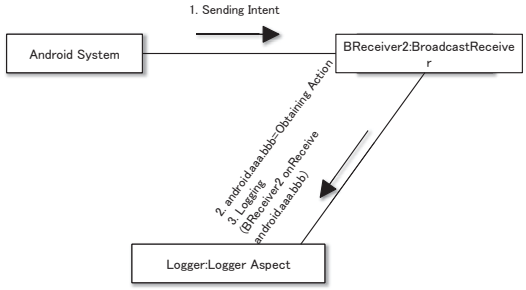

Fig. 5: Log collection of Broadcast Intent from the Android system.

3.3 Log analysis for visualization

This section shows a method, which analyzes and visualizes the collected logs, as mentioned previously. Our method generates a directed graph to visualize communication using Intent between Activities, Services, and the Android system, and annotates information such as energy consumption for each module and frequency of communication for each edge. The analyzer collects all necessary data from an Android terminal. It classifies these logs according to type of module and communication and totals them as shown as in Fig. 6. The type of communication is mentioned in Table 1.

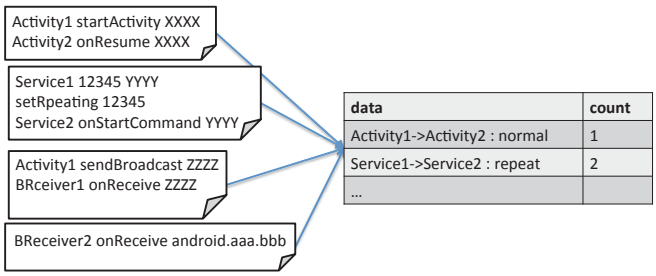

Fig. 6: Counting log files.

4. Evaluation

4.1 Environment

This section demonstrates the proposed method preliminarily to apply to a simple application for evaluation. The authors implemented a log collection function of our method and analyzed applications that are running on a smartphone with our method. Fig. 8 shows a screenshot of the application for verification. The application has three Activity classes, three Service classes, and one BroadcastReceiver class. Each Activity class can transition to another Activity except itself. There are three Service classes, and these classes are: MyService, MyService2, and MyService3, invoked from the Activity classes using startService(), setRepeating(), and bindService(), respectively. The BroadcastReceiver implemented in the application receives Intents from Service classes and the Android system. We implemented the log collection function of our method and analyzed applications that are running on a smartphone with our method.

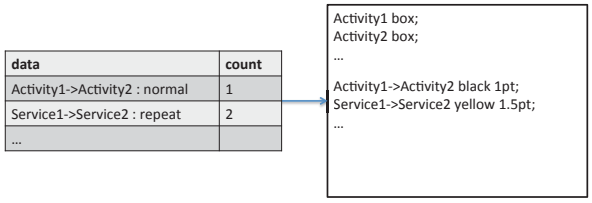

Fig. 7: Generating the DOT file.

4.2 Result

Fig. 9 shows relationships of the testing application modules. We generated this directed graph using the developed logging software and `dot`, which is a program in `Graphviz`[9]. We collected a log, as mentioned in Section 3, and converted the log to the DOT language that dot interprets. Black edges indicate normal invocation. Yellow edges show periodic invocation and Broadcast Intents. The numbers in the edge labels indicate the communication count between pairs of modules. And line thickness of these edges shows the percentage of communication count.

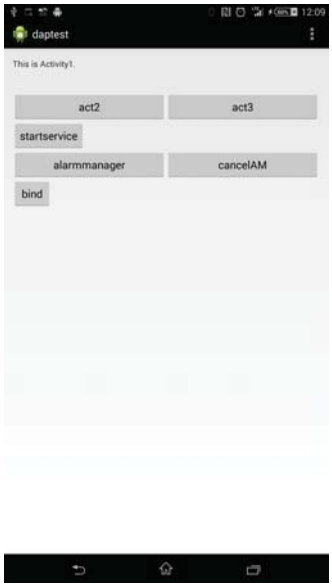

Fig. 8: Screenshot of the application for verification.

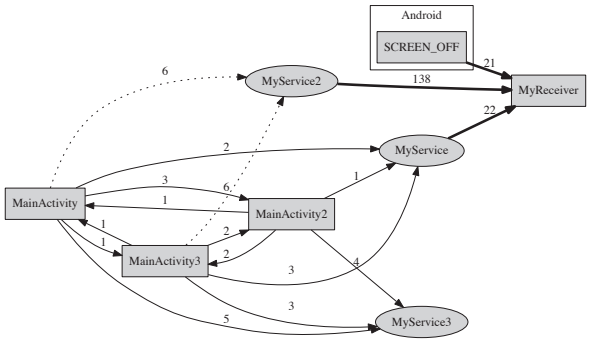

Fig. 9: Relationship graph of the testing application modules.

4.3 Discussion

$E_{processing}$, energy consumption of processing, can be estimated from information that integrates the call graph and module' s energy consumption. A subgraph of call graph indicates a processing of an application. Our proposed method[1] can estimate energy consumptions of modules themselves.

$E_{processing}$ is represented by the summation of module's energy of a processing.

$$E_{processing} = \sum_{m \in Ms} e_m \qquad (2)$$

Ms and e_m represent a set of module elements of a processing and energy of $m \in Ms$, respectively.

5. Conclusion

In this paper, we presented a profiling method identifying relationships between Android application modules. Our method monitors and records communication between modules in an Android application, and visualizes them. Our proposed method helps application developers in identifying hidden energy consumption problems that are caused by communication in the application. We identified types of communication that should be visualized in an Android application. We also preliminarily demonstrated the method using a simple application for verification and showed that it can depict communications in the application in the form of a directed graph.

Our future work will include planning to identify communication patterns in a more complicated module structure. At present, we can identify relatively simple communication patterns. We will also demonstrate our method to be applicable in real applications.

References

[1] H. Furusho, K. Hisazumi, T. Kamiyama, H. Inamura, T. Nakanishi and A. Fukuda: Power Consumption Profiling Method based on Android Application Usage,Lecture Notes in Electrical Engineering, Vol. 339, pp.891–898, Springer Berlin Heidelberg(2015)

[2] L.T. Cignetti, K. Komarov and C.S. Ellis: Energy estimation tools for the Palm, Proc. the 3rd ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWIM '00), pp.96–103, ACM, New York, NY, USA(2000)

[3] L. Zhang, B. Tiwana, Z. Qian, et al: Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones, Proc. the eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES/ISSS '10), pp.105–114, ACM, New York, NY, USA(2010)

[4] T. J. Killian: Processes as Files, USENIX Summer Conf. Salt Lake City(1984)

[5] Y. Kaneda, T. Okuhira, T. Ishihara, K. Hisazumi, T. Kamiyama and M. Katagiri: A Run-Time Power Analysis Method using OS-Observable Parameters for Mobile Terminals, 2010 International Conference on Embedded Systems and Intelligent Technology (ICESIT 2010), Vol.1, pp.39–44(2010)

[6] R. Mittal, A. Kansal and R. Chandra: Empowering Developers to Estimate App Energy Consumption, Proc. the 18th Annual International Conference on Mobile Computing and Networking (Mobicom '12), pp.317–328, ACM, New York, NY, USA(2012)

[7] The Eclipse Foundation, The AspectJ Project, available from(http://www.eclipse.org/aspectj/) (accessed 2015-05-04)

[8] PendingIntent | Android Developers, `http://developer.android.com/reference/android/app/PendingIntent.html#hashCode()` (accessed 2015-05-10)

[9] Graphviz | Graphviz - Graph Visualization Software, `http://www.graphviz.org/` (accessed 2015-05-10)