Model-based Power Consumption Analysis of Smartphone Applications

Shin NAKAJIMA

National Institute of Informatics**
Tokyo, Japan
nkjm@nii.ac.jp

Abstract. Unexpected power consumption of smartphone applications is a nuisance because the battery capacity is limited. Such energy bugs (ebugs) are currently detected only at runtime. Some ebugs, however, are desirable to detect at the early stage of the development because they are design faults. This paper proposes a formal model, the power consumption automaton, to account for the power consumption, and discusses how the tool-assisted analysis is conducted.

Keywords: Android, Energy Bugs, Stopwatch Automaton, UPPAAL, Realtime Maude, Sampling Abstraction

1 Introduction

Smartphones are compact and small, but are a complex system to consist of many components. All of them have impact on the power consumption of the battery, but their causal relationships are not clear owing to the complexity of the multi-layered architecture [1]. A functionally correct application may suffer from unexpected power consumption, which is called *energy bugs (ebugs)* [14]. Detecting such ebugs currently uses energy profilers (cf. [15]), which monitor the program execution at runtime to see whether power drains exist.

Smartphones adapt an aggressive power saving strategy to reduce the battery power consumption. Some applications, however, keep the smartphone awake even during the inactivity periods. To meet this demand, the Android framework provides application interfaces (API) for the power management [1]. The API is a two-edged sword; it is usable while it may introduce further ebugs due to improper uses of the API. These ebugs are desirable to detect at the early stage of the development because they are design faults.

This paper proposes a new model-based approach to the representation and analysis of the asynchronous power consumption of Android applications. We introduce a formal model, the power consumption automaton (PCA), show how the PCA is analyzed with existing tools and present some discussions based on our experience.

** Also affiliated with The Graduate University for Advanced Studies (SOKENDAI)

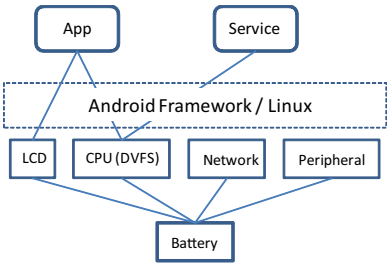

Fig. 1. Configuration

2 Power Consumption in Android

2.1 Power Management

Figure 1 shows the configuration of Android smartphones focused on the power consumption. The Android framework together with the Linux kernel provides basic features to the application programs, app and service. Android app is an application process to provide GUI to the user, while Android service does not use GUI and is a backgound process. All of them may be multi-threaded to have concurrency. In addition, the application processes use the hardware components such as networks or peripherals. These, except for the docking station, are the consumers of the battery power.

The Android framework encapsulates the underlying components for programming tasks easy. It, however, makes it hard to know the power consumption behavior precisely. The framework adapts an aggressive power saving strategy; the power control subsystem automatically forces the system to sleep when the user does not touch the screen for some periods of time.

Some app may prohibit the power saving, and the Android framework provides the wake locks for the power management. An app can request CPU resources with wake locks (`WakeLock`), and CPU is kept awake while some active locks remain. An acquired wake lock should have a matching release call. Otherwise, the wake lock is kept alive even after the caller program is destroyed. Such improper uses of wake locks result in ebugs. These ebugs can, in principle, be eliminated at the early stage of the development because they are design faults.

2.2 Wi-Fi Subsystem

This paper uses the Wi-Fi component as a concrete example to discuss the motivation and technical details of the proposal. Figure 2 is an example measured energy profile of a Wi-Fi client in Nexus One operating in the power save mode (PSM) [3]. It is taken from [11] to be modified a little for illustrative purposes.

The Wi-Fi client, or station (STA), is in the passive scan mode. The access point (AP) periodically sends beacon signals to notify the STA to start transferring data. Figure 2 shows that the STA is first in the Deep Sleep state, and

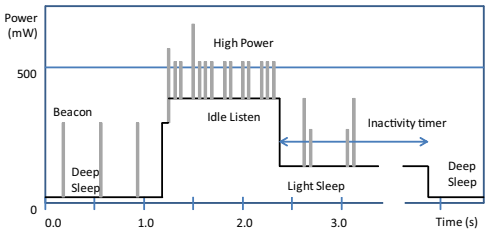

Fig. 2. Wi-Fi Energy Profile (Adapted from [11])

then goes to the High Power state to exchange various frames. The STA stays in the Idle Listen state to see if there are further frames to come. The STA moves down to the Light Sleep state when it recognizes a no-more-data flag in the data frame. The STA stays in this state for a while to be ready for further data transfer occurring soon. The STA returns to the Deep Sleep state by a time-out event of the inactivity timer.

The Wi-Fi subsystem also adapts the notion of wake locks for the power management. `WifiManager` provides a method to control over `WifiLock` [1]. It allows an application to keep the Wi-Fi radio awake. The STA, while a lock is active, stays in the Light Sleep, but does not go to the Deep Sleep state .

2.3 Power Consumption Model

If the graph in Figure 2 is represented as a function $F(t)$, the total power consumption ($P(T)$) from 0 to a time T is obtained by the accumulation, or integral of $F(t)$; $P(T) = \int_0^T F(t)dt$. As the above explanation with Figure 2 suggests, the states of the STA are changed in either event-trigger (signal or frame) or time-trigger (periodic or timeout) manners. Such changes can be represented by state-transition system consisting of power states. Figure 3 is an example state-transition system of Wi-Fi STA operating in PSM.

This paper focuses on the analysis of unexpected power consumption, but does not try to predict the power consumption behavior in a numerically precise manner. It is suffice to check whether the whole system stays at some particular power state in a unexpectedly long time. $F(t)$ can be approximated by a linear funtion of time. Let $P(t_S, t_E)^{PS}$ be the consumed power in the state PS from t_S to t_E that $P(t_S,\ t_E)^{PS}$=$C^{PS}\times(t_E - t_S)$ with a constant C^{PS}. $P(T)$ becomes a sum of the power consumptions in all the states that the STA transition system visits.

$$P(T) \ = \ \sum_{i=0}^{N-1} P(t_i,\ t_{i+1})^{PS}$$

where $t_0 = 0$ and $t_N = T$. $P(t_S, t_E)^{PS}$ can be written as $P(t)^{PS}$, equal to $C^{PS}\times t$, with $t = t_E - t_S$. For example, $P(t)^{DeepSleep}$ refers to the sum of the

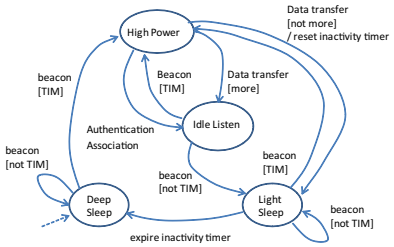

Fig. 3. Behavioral Model of STA

power consumed as the drain currents of the radio circuits and that needed to process the beacon signals (see Figure 2). A state, *DeepSleep* for example, is visited many times while the Wi-Fi subsystem is enabled.

3 Model-Based Analysis of Power Consumption

3.1 Power Consumption Automaton

We introduce a state transition system, power consumption automaton[1], to be a formal representation. Power consumption automaton (PCA) is a 6-tuple, whose definition follows the presentation in [4], $\langle\ Loc, Var, Lab, Edg, Act, Inv\ \rangle$. The components are explained below.

1. Loc is a finite set of locations to represent power states.
2. Var is a finite set of real-valued variables, A valuation v for the variables is a function to assign a real-value $v(x) \in R$ to each variable $x \in Var$. V represents the set of valuations (v).
3. Lab is a finite set of synchronization labels to contain the stutter label $\tau \in Lab$.
4. Edg is a finite set of transitions. Each transition e is a tuple $\langle l, a, \mu, l' \rangle$ where $l, l' \in Loc$ are a source and a target, $a \in Lab$ is a synchronization label, and μ is an action defined by a guarded set of assignments (updates)

   $$\psi \Rightarrow \{\ x := \alpha_x \mid x \in Var\ \}.$$

   where the guard ψ is a linear formula over the variables, and α_x is a linear term.
5. Act is a mapping from locations in Loc to a set of activities to represent the flow dynamics. $Act(l)$ is a differential equation of the form $dP/dt = K$ with $K \in Z$. K is C^l for the case of power consumption and 1 for clock such as the inactivity timer.
6. Inv is a mapping from locations in Loc to invariants $Inv(l) \subseteq V$. $Inv(l)$ is defined by a linear formula ϕ over Var.

[1] It was introduced first in an oral presentation [12].

Two PCAs synchronize on the common set of labels $Lab_1 \cap Lab_2$. Whenever PCA_1 makes a discrete transition with a synchronization label $a \in (Lab_1 \cap Lab_2)$, PCA_2 also performs a discrete transition. Therefore, concurrency is naturally represented; an example is found in Figure 4.

PCA is a strict subclass of linear hybrid automaton (LHA) [5]. LHA is a hybrid automaton whose guards (ψ), updates (μ), and invariants (ϕ) are only linear expressions, and the dynamics are specified using differential inequalities that are linear constraints over first-order derivatives ($C_1 \leq dx/dt \leq C_2$). The dynamics of PCA are differential equalities of the form, $dP/dt = C^l$ for the power consumption, and $dX/dt = 1$ for the clock variable X, which are linear equality constraints over first-order derivatives. PCA is a subclass of LHA.

3.2 Reachability Problem

The algorithmic analysis methods were studied for linear hybrid automaton (LHA), specifically the reachability analysis over infinite state spaces generated by the labeled transition system (LTS) associated with the LHA [4].

LHA has several special cases such as a timed automaton (TA), a multirate timed system (MTS), n-rate timed system (nRTS), and a stopwatch automaton (SWA). An LHA is *simple* if the location invariants (ϕ) and transition guards (ψ) are of the form $x \leq k$ or $k \leq x$ for $x \in Var$ and $k \in Z$. A variable x is a clock if $Act(l, x) = 1$ for each location and $\mu(e, x) \in \{0, x\}$ for each edge. A skewed clock is a clock to change with time at some rate $k \in Z$; $Act(l, x) = k$ and $\mu(e, x) \in \{0, x\}$.

A TA is a simple linear hybrid automaton with clocks. An MTS is a linear hybrid automaton whose clocks are skewed. An nRTS is an MTS whose skewed clocks proceed at n different rates. A SWA is a TA where $Act(l, x) \in \{0, 1\}$ and $\mu(e, x) \in \{0, x\}$. The reachability problem is known decidable for TA and *simple* MTS, but undecidable for other cases even if they are simple.

3.3 Analysis of Power Consumption Automaton

We assume here a PCA with N power states, each of which consumes different electric power as $P(t)^j = C^j \times t$ ($j = 0 \ldots N-1$). The property to check may take a form of $\Box(Pow \leq Max)$, which mentions that the total power (Pow) is always within a given Max. We consider how PCA is encoded in some subclass of LHA to see the decidability of the PCA reachability analysis.

First, a PCA is encoded as an N-rate timed system. A skewed clock Pow is introduced, which changes at the rate of C^j in the power state j, proceeding at N different rates. Pow accumulates all the consumed power. Alternatively, a PCA is represented as an MTS. We regard each $P(t)^j$ to be a skewed clock X^j to change at a fixed rate of C^j. Since X^j changes with time only when the PCA stays on that state j, X^j must be stopped in all the other states. PCA is now a multirate stopwatch automaton, where the total power consumption is calculated such that $Pow = \sum_{j=0}^{N-1} X^j$. PCA is more expressive than either MTS or SWA. Some over-approximation method is needed in the tool-assisted analysis.

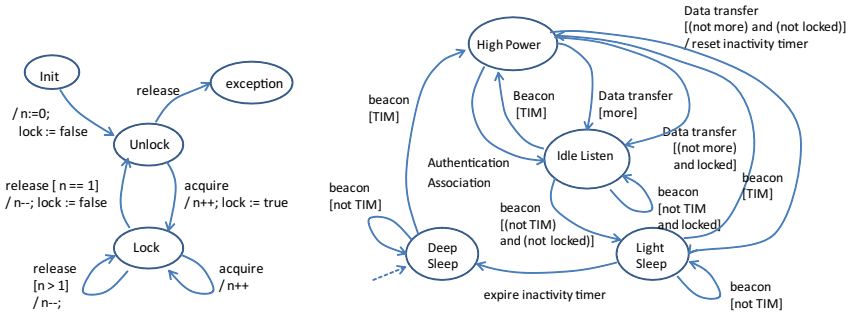

Fig. 4. Wi-Fi PSM with Wake Lock

4 Analysis with Automatic Tools

This section presents two methods of representing and analyzing the PCA with two automatic tools, UPPAAL [7][9] and Readltime Maude [2][13].

4.1 Wi-Fi Power Consumption Automaton

The Android framework provides `WifiLock` to allow an application to keep the Wi-Fi awake [1]. Figure 4 shows diagrammatic representations of the PCAs for a wake lock (the left) and a Wi-Fi STA (the right).

The reference-counted `WifiLock` in Figure 4 shows that the `acquire()` and `release()` must be balanced, which is similar to counting semaphore. The `acquire()` method locks the Wi-Fi on until the `release()` is called to turn it off. The locked Wi-Fi stays in the `Idle Listen` state even when the data transfer ends with the no-more-data flag on[2].

4.2 Analysis using UPPAALL

This subsection sketches how the PCA is encoded in the stopwatch-extension of UPPAAL [7][9].

Let X_{PS} be a clock variable introduced for a power state PS. We add invariants $dX_{PS}/dt = 0$ to all the states except in the state PS. X_{PS} must proceed in the state PS and thus the invariant there is $dX_{PS}/dt = 1$. Figure 5 shows four clock variables, for recording how long the Wi-Fi client stays in each state. The first-order derivative of `xIL` (written `xIL'`), for example, is set to `0` in all the states other than `Idle Listen`.

We then use a query expression of the form $\mathrm{A}\Box(X_{PS} \leq A_{PS})$ where A_{PS} is a constant to refer to the maximum power consumption allowed for this power state. For example, a counter-example is generated for the case of $\mathrm{A}\Box(X_{IL} \leq A_{IL})$ when WiFi lock bugs are hidden in the design.

[2] Comapre Figure 4 with Figure 3 to study the differences.

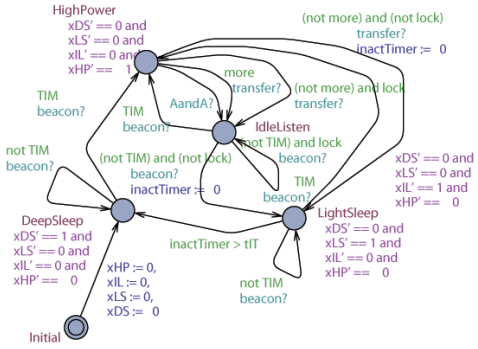

Fig. 5. Wi-Fi Client with Lock (Taken from [12])

4.3 Analysis Using Realtime Maude

PCA is encoded as an n-rate timed system, represented in Realtime Maude.

Realtime Maude Realtime Maude [2][13] is an extension of Maude [8] for the application of rewriting logic to the executable specifications of realtime and hybrid systems. Realtime Maude provides two kinds of rewrite rules, the instantaneous rules and tick rules. Let $T(A)$ be a term. A conditional instantaneous rule with a label r takes a form below.

$$r : T(A) \longrightarrow T(A') \ \textbf{if} \ C \ .$$

The term on the left-hand side ($T(A)$) is rewritten to the right-hand side one ($T(A')$), while the rule is fired only when the side condition C is *true*.

The tick rules are responsible for time progress. Let T be a set of terms, a snapshot of the whole system. A tick rule works on $\{\ T\ \}$ nondeterministically. Time-nondeterministic systems are analyzed under *sampling abstractions*, in which the maximum time elapsed (mte) plays an important role.

$$\mathrm{l} : \{\ T\ \} \longrightarrow \{\ \delta(T, \tau_l)\ \} \ \textbf{in time} \ \tau_l \ \textbf{if} \ \ \tau_l \leq mte(T)$$

The rule states that the amount of time τ_l is passed in rewriting T to $\delta(T, \tau_l)$. $mte(T)$ is the upper limit of the time progress, and thus it instructs the analysis method to fire transitions at *some* time so long as the side condition is satisfied. The transition is fired at least once in the time interval that $mte(T)$ specifies. Two functions, mte and δ, are defined for each term T.

Realtime Maude supports the timed model-checking of the property expressed in linear temporal logic (LTL) under the sampling abstraction.

Encoding Wi-Fi PCA in Realtime Maude The Wi-Fi STA in Figure 4 requires a set of rewriting rules, and the below shows some fragments of its behavior only. First, we introduce a $wifiSTA$ term to represent the PCA to have four components; *Loc* for power states, *Power* for accumulated power consumption, a Boolean flag *Bool* for the WifiLock status, and *Time* for the inactivity timer.

$$\textbf{op}\ wifiSTA\ : Loc\ \ Power\ \ Bool\ \ Time\ \longrightarrow System$$

Here are three instantaneous rules; transitions from the High Power state when $wifiSTA$ receives data with flags.

$$\begin{aligned}
&\text{h1: } wifiSTA(HighPower, P, B, X)\ Data(More)\\
&\quad \longrightarrow wifiSTA(IdleListen, P, B, X)\\
&\text{h2: } wifiSTA(HighPower, P, false, X)\ Data(NoMore)\\
&\quad \longrightarrow wifiSTA(LightSleep, P, false, X)\ Reset\\
&\text{h3: } wifiSTA(HighPower, P, true, X)\ Data(NoMore)\\
&\quad \longrightarrow wifiSTA(IdleListen, P, true, X)
\end{aligned}$$

δ calculates the consumed power at each state. The first equation below shows the case for the High Power state, and the rests describe the two cases for the Light Sleep state. They select appropriate new states depending on the status of the inactivity timer. C stands for the timeout constant.

$$\begin{aligned}
&\delta(wifiSTA(HighPower, P, B, X), \tau)\\
&\quad = wifiSTA(HighPower, P + C^{HP}\times\tau, B, X)\\
&\delta(wifiSTA(LightSleep, P, B, X), \tau)\\
&\quad = wifiSTA(LightSleep, P + C^{LS}\times\tau, B, X + \tau)\ \textbf{if}\ X < C - \tau\\
&\delta(wifiSTA(LightSleep, P, B, X), \tau)\\
&\quad = wifiSTA(DeepSleep, P + C^{LS}\times\tau, B, X + \tau)\ \textbf{if}\ X \geq C - \tau
\end{aligned}$$

The inactivity timer is time-dependent component in $wifiSTA$. *mte* is defined so that the timer is checked at least once at some critical intervals.

$$\begin{aligned}
&mte(wifiSTA(L, P, B, X)) = inf\ \textbf{if}\ not(L = LightSleep)\\
&mte(wifiSTA(L, P, B, X)) = C - X\ \textbf{if}\ (L = LightSleep)
\end{aligned}$$

4.4 Discussion

Section 3.3 presented two of PCA encoding methods with LHA subclasses; (a) a multirate stopwatch automaton, and (b) an n-rated timed system.

The approach with the stopwatch-extension of UPPAAL has a slight restriction of the case (a) above since the clocks are not skewed. Therefore, query checking is performed separately for each clock. A sum of clock variables does not make sense due to the symbolic model-checking algorithm. The encoding with Realtime Maude adapts the approach (b). We introduced a single skewed

clock variable *Pow* to accumulate the consumed power. *Pow* proceeds at a different rate in each power state. Therefore, the total amount of the consumed power can be analyzed, although the sampling abstraction and time-bounded search methods have to be employed.

This paper used the Wi-Fi subsystem as the example PCA, not an Android application, while the motivation of this work was removing ebugs from the applications at the design stage. Since an application uses a wide variety of components, all the power consumers in the system must be represented as *libary* PCAs. The Wi-Fi PCA is a first example of such library components. Furthermore, Android application is implemented as a subclass of `Activity`. Its behavior is expressed in terms of state-transition system of the life-cycle implemented by a set of callback methods. Such behavioral specification can be captured by PCA. Thus, the model-based analysis with PCA can be useful for the design-level checking of ebugs in Android applications.

Android smartphones are equipped with ARM core processors to have the dynamic voltage-frequency scaling (DVFS). The DVFS governor in Android changes the voltages and frequencies based on the CPU usage, and has impact on the CPU power consumption and the execution time of a program as well. The CPU usage is related to the number of active processes, which is not known beforehand. The power consumption affected by the DVFS may be considered probabilistic. Therefore, the behavioral analysis of the PCA may need the stochastic model-checking method [17].

5 Related Work

A. Pathak et al recognized the importance of eliminating energy bugs in smartphones, which they called *ebugs* [14]. They also proposed to use the state-transition systems for modeling of the power consumption and developed Eprof [15]. Eprof is an energy profiler to monitor the program execution at runtime to detect potential ebugs.

MoVES [6] uses the UPPAAL/SWA for modeling and analyzing embedded systems such as the schedulability or the power consumption. The power consumption model, however, is that $P(t) = C \times t$ without considering the differences in power states. C. Thompson et al [16] presented a model-driven approach to develop smartphone applications. The power consumption analysis was performed on the generated program codes. The quantitative results reproduced the implemented programs. They, however, did not discuss the formal analysis at the design level.

6 Conclusion

The power consumption model (Section 2.3) assumes that $P(t)$ is linear in time. It is one of the future work to ensure that the approximation can mostly reproduce the energy profile in Figure 2. It requires a measurement by using the energy profilers such as Eprof.

We compared two approaches to the representation and analysis of the power consumption automaton using UPPAAL and Realtime Maude. Further feasibility study is planned to use HyTech [10]. Since the power consumption problem has some aspects that are hard to predict, such as the effects by the DVFS governor, some stochastic analysis methods would be valuable. It is an important research topic from both theoretical and practical perspectives (Section 4.4).

References

1. Android. `http://developer.android.com`.
2. Realtime Maude. `http://heim.ifi.uio.no/peterol/RealTimeMaude/`.
3. IEEE Standard 802.11, Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications , 1999.
4. R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The Algorithmic Analysis of Hybrid Systems, *Theor, Comp. Sci.*, No.138, pp.3-24, 1995.
5. R. Alur, Formal Verification of Hybrid Systems, in *Proc. EMSOFT'11*, pp.273-278, 2011.
6. A. Brekling, M.R. Hansen, and J. Madsen. MoVES – A Framework for Modeling and Verifying Embedded Systems, In *Proc. ICM2009*, pp.149-152, 2009.
7. F. Cassez and K.G. Larsen. The Impressive Power of Stopwatches, In *Proc. CONCUR 2000*, pp.138-152, 2000.
8. M. Clavel, F. Duran, S. Eker, O. Lincoln, N. Marti-Oliet, J. Meseguer, and C. Talcott. *All About Maude - A High-Performance Logical Framework*, Springer 2007.
9. A. David, J. Illum, K.G. Larsen, and A. Skou. Model-based Framework for Schedulability Analysis Using UPPAAL 4.1, Aalborg University 2009.
10. T.A. Henzinger, P.-H. Ho, H. Wong-Toi. HyTech: A Model Checker for Hybrid Systems, *Software Tools for Technology Transfer*, 1:110-122, 1997.
11. J. Manweiler and R.R. Choudhury. Avoiding the Rush Hours: WiFi Energy Management via Traffic Isolation, In *Proc. MobiSys'11*, 2011.
12. S. Nakajima and Y. Ueda. Power Consumption Analysis of Smartphone Applications using UPPAAL, In *Proc. 1st CPSNA-WIP*, 2013.
13. P.C. Olveczky and J. Meseguer. Semantics and Pragmatics of Real-Time Maude, *Higher-Order and Symbolic Computation*, vol.20, no.1-2, pp.161-196, 2007.
14. A. Pathak, Y.C. Hu, and M. Zhang. Bootstrapping Energy Debugging on Smartphones: A First Look at Energy Bugs in Mobile Devices, In *Proc. Hotnets'11*, 2011.
15. A. Pathak, Y.C. Hu, and M. Zhang. Fine Grained Energy Accoutning on Smartphones with Eprof: Where is the energy spent inside my app?, In *Proc. EuroSys'12*, 2012.
16. C. Thompson, D. Schmidt, H. Turner, and J. White. Analyzing Mobile Application Software Power Consumption via Model-Driven Engineering, In *Proc. PECCS 2011*, pp.101-113, 2011.
17. P. Zuliani, C. Baier, and E.M. Clarke. Rare-Event Verification for Stochastic Hybrid Systems, In *Proc. HSCC 2012*, pp.217-225, 2012.