Vol.2013-EMB-30 No.4
2013/9/17

スマートフォン・アプリ電力消費のモデルベース解析

中島 震[1,2,a)]

概要：電池容量が小さいスマートフォンでは，アプリケーションの電力消費が大きな問題となる．機能振る舞いが意図通りであっても，電池の消耗が予想以上に大きい場合は，アプリケーションの不具合である．このような不具合（電力バグ）はプログラム開発後に行われるプロファイラを用いた事後検査で調べることが多い．しかし，電力バグには，フレームワークが提供する電力管理 API の誤使用など，設計時の不具合が原因となる場合がある．開発の上流工程で検知し，除去すべきであろう．本稿では，アンドロイド・アプリケーションの電力消費を対象としたモデルベースの解析手法を提案する．

1. はじめに

スマートフォンは手のひらにのる小ささでありながら，多様なハードウェア部品からなる複雑なシステムである．アンドロイドは階層アーキテクチャ [1] になっておりコンポーネントが複雑に絡み合う．その結果，システムのさまざまな構成コンポーネントが電力を消費するにも関わらず，電力消費の関係を把握することが難しい．たとえば，機能振る舞いが意図通りのアプリケーションでも，予想外の電力消費を示すことがある．一種の非機能的な不具合であり，「電力バグ（ebugs）」[21] と呼ぶ．現状，このようなアプリケーションのデバッグには，プログラム実行時の消費電力を監視するエネルギー・プロファイラ ([22] など) が使われている．

スマートフォンは，電池の消耗を抑えることを目的として，パワーセーブを積極的に行う．ところが，パワーセーブ・モードに遷移しないように，スマートフォンを「起こした状態」に保ちたい場合がある．このような制御を実現することを目的に アンドロイド・フレームワークは，`WakeLock` などの電力管理を行うクラス群を提供する [1]．これらのクラスは便利な反面， たとえば，`WakeLock` の不適切な使い方によるバグ混入の可能性が残る．これは設計バグであり，開発の上流工程で検知し，除去すべき不具合である．エネルギー・プロファイラに頼らない設計検証の技術が必要である．

本稿はアンドロイド・アプリケーションの電力消費問題に対して，モデルベースの表現と解析の方法を提案する．電力消費を状態遷移システムの考え方で表現する電力消費オートマトン (PCA)[17] に基づき，既存ツール利用の方法を考察する．

1 国立情報学研究所, National Institute of Informatics
2 総合研究大学院大学, SOKENDAI
a) nkjm@nii.ac.jp

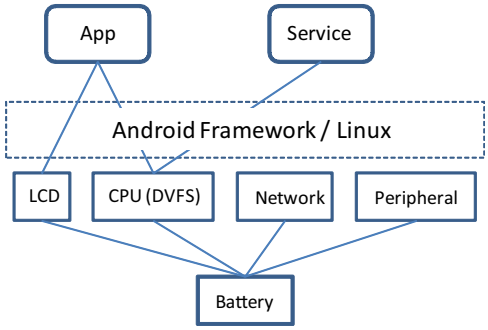

図 1 アンドロイド・スマートフォンの構成概要

2. アンドロイドの電力管理

2.1 階層アーキテクチャ

アンドロイド・スマートフォンの概要を図 1 に示す．電力消費に関わるコンポーネントを中心とした階層を表現した．

アンドロイド・フレームワークはリナックス・カーネルの上に構築されており，アプリやサービスといったアプリケーション・プログラムに基本機能を提供する．アンドロイド・アプリは GUI を持つアプリケーション・プロセスであり，一方，サービスは GUI を持たないデーモン・プロセスの一種である．いずれも，単一スレッドあるいはマルチスレッドで作動する．さらに，これらのアプリケーションは，ネットワークや周辺機器といったハードウェア・コンポーネントを利用する．その多くは電力の消費者である．唯一の例外は，電力を供給するドッキング・ステーションであろう．本稿では，電力消費者としてのハードウェア・コンポーネントを考える．

ⓒ 2013 Information Processing Society of Japan

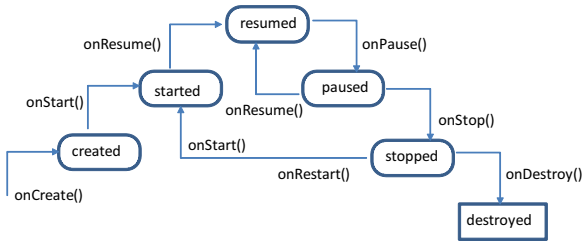

図 2 アクティビティのライフサイクル

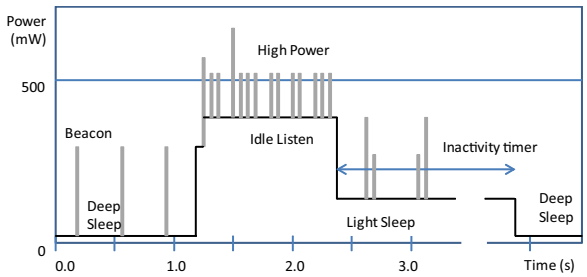

図 3 Wi-Fi 電力消費プロファイル (文献 [15] から引用)

アプリケーション・プログラム開発の立場からみると，アンドロイド・フレームワークが，リナックス・カーネルやハードウェア・コンポーネントを隠蔽しているとみなせる．これによって，プログラミング作業が容易になるという利点がある．逆に，このことは，電力消費に関わる振る舞いを厳格に把握することを難しくしている．

また，アンドロイドは，積極的なパワーセーブの方法を採用している．ある時間間隔，ユーザがスクリーンを操作しない時，電力管理サブシステムが作動して，システムを自動的にスリープさせる．これによって，電池の消耗を軽減する．

このような電力管理の方法を採用すると同時に，アンドロイド・フレームワークは起きている状態を持続させるウエイク・ロック（wake lock）による電力管理機構を提供している．たとえば，クラス `WakeLock` がある．アプリケーションはウェイク・ロックを使って CPU 資源を要求することができる．ウェイク・ロックが有効（アクティブ）な限り，CPU はスリープ状態にならない．

ウェイク・ロックは，特に指定しない限り，参照カウント方式で管理されている．獲得メソッドには解放メソッドが対応しなければならない．対応しない場合，呼び出し側プログラムが実行終了した後でも，ウェイク・ロックはアクティブであり続ける．電力管理機構があるにも関わらずシステムはスリープ状態にならず，電力が消費され続ける．このようにウェイク・ロックの使い方を誤ると，電力バグ（エネルギーバグ）[21] となる．メソッドの使い方の誤りであり，設計の不具合と関連することから，開発の上流工程で検知し，除去したい．

2.2 アンドロイド・アプリ

アンドロイド・アプリは，フレームワークが提供しているクラス `Activity` を継承して作成する．機能振る舞いを実現するコールバック・メソッドは図 2 が示すライフサイクルにしたがって起動される．アプリ実行開始時に呼ばれる `onCreate()` から始まり，ユーザ操作によっていくつかのメソッドが呼ばれ，終了時には `onDestroy()` が起動される．ライフサイクルに基づくことから，状態遷移システムの考え方を用いて，アプリ設計を進めることができる．

コールバック・メソッドはユーザ操作がきっかけとなって起動されることから，サブアクティビティへの切り替えなど，複数アプリの競合によって，あるアプリに着目した場合，コールバック・メソッドの起動系列が影響を受けることがある．さらに，多くのアプリはマルチスレッド機能を利用する．たとえば，WiFi などの通信を利用するアプリは，ユーザ操作を司るビュー・スレッドと通信処理スレッド[*1]を分ける必要があり，マルチスレッドになる．このようなアプリでは，状態遷移システム群が並行動作すると考えなければならない．いずれの場合も複数の状態遷移システムからなる全体を表現し，解析する方法が必要である．

2.3 Wi-Fi サブシステム

次に，Wi-Fi サブシステムを対象として，スマートフォンの電力消費を具体的に説明する．

図 3 は，パワーセーブ・モードで作動する Nexus One の WiFi クライアントを対象とした測定結果を示す．この電力消費プロファイルは，文献 [15] の結果をもとに模式的な図として表した．測定の詳細な条件については，上記の文献を参照されたし．

Wi-Fi クライアントはステーション（STA）とも呼ばれる．図 3 はパッシブスキャン・モード [2] で作動している場合になる．アクセスポイント（AP）は定期的にビーコン信号を送り，STA に対して，転送データの有無を通知する．図 3 では，最初，STA は Deep Sleep 状態にいる．ビーコン信号によってデータ転送が開始されることがわかると，High Power 状態に遷移し，認証ならびにアソシエーション後，データフレームを受け取る．図 3 は，データ転送の電力消費が大きいことを示す．

STA は，データ転送の切れ間では，Idle Listen 状態で，次のデータフレームの到着を待つ．AP からのフレームに「データ終了」フラッグが設定されている場合，STA は Light Sleep 状態に遷移し，データ転送が継続する可能性に備える．Deep Sleep 時よりも電力供給を大きくすることで，フレーム処理への切り替えを高速化することが可能になる．Light Sleep 状態から Deep Sleep 状態への遷移は，イナクティビティ・タイマーのタイムアウトによって起こる．

[*1] 非同期タスク・ローダを用いる．

ⓒ 2013 Information Processing Society of Japan

Wi-Fi サブシステムは，他のハードウェア・コンポーネントと同様に，電力管理を行うウェイク・ロックの機能を持つ．クラス `WifiManager` が全体を管理機能を司り，クラス `WifiLock` を提供している [1]．第 5.1 節で，ここで述べた電力消費プロファイルを説明する状態に，`WifiLock` の影響を考慮する場合を示す．

2.4 省電力化 CPU

スマートフォンは，省電力化 CPU を採用している．適切な制御を行うことで，電池の消耗を抑えることができる反面，アプリの電力消費量予測が難しくなる．ここでは，DVFS（Dynamic Voltage-Frequency Scaling）[10] の効果と影響について説明する．

アンドロイド・スマートフォンの CPU は，ARM コアを搭載し，動的な電圧周波数変化（DVFS）に基づく電力管理機能を利用している．CPU の半導体としての消費電力は，電圧の 2 乗と周波数に比例するので，低電圧・低周波数にすることで消費電力を抑えることができる．

アンドロイド・フレームワークは，リナックス提供のオンディマンド・ガバナーに，対話的なユーザ操作の特徴を加味したインタラクティブ・ガバナーを使う．オンディマンド・ガバナー [19] は，リナックスのプロセス・スケジューラと協働する．CPU 利用率に応じて，CPU への供給電圧と動作周波数を変更する．CPU 利用率が低い時は，低電圧ならびに低周波数の低速動作モードにし，利用率が高くなるにしたがって，高電圧高周波数の高速動作モードに移る．

低速動作モードでは，アプリの物理的な実行時間が長くなる．アンドロイド・アプリはユーザ操作の対話性が主な関心事であって，実時間特性は厳しくない．ソフトリアルタイム・システムといえる．したがって，アプリ実行の物理的な時間が長くなっても，電力消費を小さくすることを優先する．

ところが，アプリがウェイク・ロック等による電力バグを含む場合，ハードウェア・コンポーネントの電力消費は物理的な時間に依存することから，バグに起因する電力消費も大きくなる．電力消費プロファイルは，厳密には，DVFS ガバナーの影響，つまり，CPU 利用率に影響されることになる [*2]．

一方，CPU 利用率は，アプリやサービスの実行状況によって決まり，予め決めることができない．電力消費への DVFS ガバナーによる影響は実行時になってはじめてわかる．モデルベース解析の方法では，このような不確定さという特徴を考慮しなければならない．

2.5 電力消費モデル

システム全体の電力消費 P^{TOTAL} を，簡単に次のように考える．

$$P^{TOTAL} = P^{BASE} + P^{SOFT}$$

ここで，P^{BASE} はシステム基盤をなすハードウェアの電力消費であり，CPU やメモリといった常に作動する基本コンポーネントが影響する．DFVS を考慮すると，P^{BASE} も変動する．P^{SOFT} は本稿で問題としているアプリの振る舞いに起因する電力消費である．

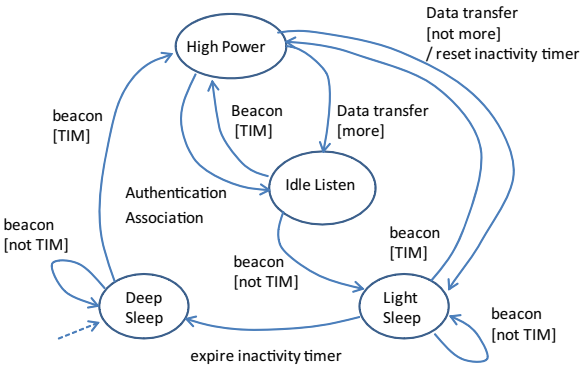

図 4 Wi-Fi STA の振る舞い

ここで，P^{SOFT} の特徴を考察する具体例として，Wi-Fi サブシステムについて考える．図 3 の電力消費推移を表すグラフを関数 $F(t)$ とする．時間 0 から時間 T までの電力消費 $P(T)$ は，$F(t)$ を加え合わせれば良いので，積分記号を使って整理できる．

$$P(T) = \int_0^T F(t)dt$$

図 3 の説明にもあったように，STA の状態はイベント駆動（ビーコンやフレーム送受信）ならびにタイム駆動（周期処理やタイムアウト）によって変化する．このような変化は状態遷移システムの考え方で表現できる．すなわち，パワー状態の考え方を導入し、それらの間の遷移として表せばよい．パワーセーブ・モードで作動する Wi-Fi の STA を状態遷移システムで表現したのが，図 4 である [17].

上記では， $P(T)$ を関数 $F(T)$ の積分として表した．各パワー状態での消費電力が，その状態に滞留する時間に比例すると仮定すると $P(T)$ は簡単になる．このような近似を行っても，図 3 の電力消費プロファイルを良く再現する．

この近似によると，時点 t_S から t_E までのパワー状態 PS での電力消費を $P(t_S, t_E)^{PS}$ とする時，C^{PS} を定数として，$P(t_S,\ t_E)^{PS}$=C^{PS}×$(t_E - t_S)$ が成り立つ．この時，$P(T)$ は，STA の状態遷移システムが移り動いた全てのパワー状態について，次のような総和になる．

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

ここで，$t_0 = 0$ と $t_N = T$ とした．時間依存性だけを考える場合，$P(t_S, t_E)^{PS}$ は $t = t_E - t_S$ として，$P(t)^{PS} = C^{PS} \times t$ となる．

[*2] 影響が定量的に有意であるかは実験によって確認すべきであろう．

ここで，たとえば，$P(t)^{DeepSleep}$ は，アナログ回路を駆動する電流やビーコン信号の情報を解析するのに必要な電力などの総和になる．なお，Wi-Fi サブシステムが使用可能になっている間，STA は，このようなパワー状態に何度も遷移する．

3. モデルベース表現

3.1 電力消費オートマトン

STA を状態遷移システムで表現すると，状態遷移の列は STA の動作を追跡して得られるパワー状態の遷移列になる．$P(T)$ は全てのパワー状態で消費した電力の総和であり，この遷移列から求めることができる．$P(T)$ を得る方法として，パワー状態を対象とする状態遷移システム，パワー消費オートマトン，を導入する．

パワー消費オートマトン（PCA）を文献 [5] に準じた方法で表すと，次のような 6 つの構成要素からなるオートマトンとして定義できる．

$$\langle\ Loc, Var, Lab, Edg, Act, Inv\ \rangle.$$

各構成要素を説明する．

(1) Loc はパワー状態を表すロケーションの有限集合．

(2) Var は実数変数の有限集合，変数のバリュエーション v は各変数 x（$x{\in}Var$）に実数値を対応させる関数（$v(x){\in}R$），V はバリュエーションの集合．

(3) Lab は同期ラベルの有限集合であって，スタッタ記号 $\tau{\in}Lab$ を含む．

(4) Edg は遷移の有限集合．遷移 e はタプル $\langle l, a, \mu, l'\rangle$ として表され，$l{\in}\ Loc$ と $l'{\in}Loc$ が各々，遷移元と遷移先ロケーションを表し，$a{\in}Lab$ は同期ラベルを，μ はガード付き代入（更新）によって定義されるアクション．

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

において，ガード ψ は変数の線型式，値の計算式 α_x も線型．

(5) Act はロケーション Loc からフロー・ダイナミックスを表すアクティビティの集合への関数．$Act(l)$ は，K を整数定数 $(K{\in}Z)$ として，$dP^l/dt = K$ の形をとる微分方程式であって，電力消費 $(P(t)^l)$ の場合には K が各パワー状態の定数 C^l であり，クロック変数の場合には 1.

(6) Inv はロケーション Loc から不変量 $Inv(l){\subseteq}V$ への関数．$Inv(l)$ は変数の集合 Var 上の線型式 ϕ.

2 つの PCA は共通するラベル $Lab_1{\cap}Lab_2$ によって同期する．すなわち，PCA_1 が同期ラベル $a{\in}(Lab_1{\cap}Lab_2)$ で離散遷移する時，PCA_2 も離散遷移する．

PCA は線型ハイブリッド・オートマトン (LHA) [6] のサブクラスになる．LHA は，ガード (ψ)，更新 (μ)，不変量 (ϕ) が線型式であって，フローダイナミックスが 1 階微分の線型制約 $(C_1 \le dx/dt \le C_2)$ であるような線型ハイブリッドシステムである．PCA のダイナミックスは $dP^l/dt = C^l$（電力消費）あるいは $dX/dt = 1$（クロック変数）という線型 1 階微分方程式であり，LHA で表すことができる．

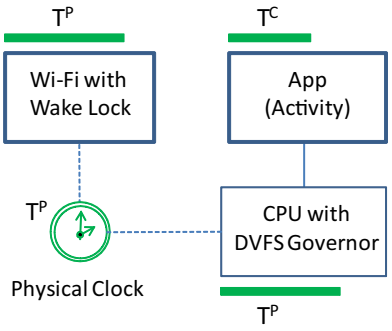

図 5 DVFS の影響

3.2 クロックの変動

DVFS ガバナーによる CPU 制御では，供給電圧と動作周波数は，CPU 利用率に影響される（第 2.4 節）．しかし，どのくらいのアプリケーションが実行しているかを予め知ることはできない．このような不確定さを表すには，PCA のダイナミックスを拡張する必要がある．

CPU が最大性能で作動している場合を基準とする．これは，DVFS ガバナー制御の観点では，CPU 利用率が 100%の場合に相当する．CPU 動作が遅くなると，アプリケーション実行時間は長くなる．アプリケーションからみると，遅いクロック変数によって，状態遷移が支配されることになる．一方，アプリケーションが使用しているハードウェア・コンポーネントの電力消費は物理時間に依存する．アプリケーションの見かけの実行時間が遅くなると，コンポーネントへの制御メソッド起動間隔が大きくなり，それだけ多くの電力を消費する．この状況を，図 5 に模式的に示した．アプリケーションの見かけの経過時間（T^C）は物理時間（T^P）よりも短い．

上記に説明したようなクロックの変化を PCA に採り入れる方法を考える．まず，時間 t によって変化する関数 $Q(t)$ を導入する．ある定数を α（$1 \le \alpha$）として，$Q \in [1,\ \alpha]$ とする．

今，仮に，アプリケーションの状態不変量（Inv）に $X \le N$ の制約があるとする．遅いクロック変数 $Q(t)$ によって作動していると，基準クロック T_S で作動する場合とは異なる制約，$X_S \le N{\times}Q$，を扱うことになる．すなわち，この状態に留まることが可能な最大時間が長くなる．アプリケーションを表した PCA では，クロック変数 X のダイナミックスが，$dX/dt\ =\ 1/Q(t)$ となることである．

逆に，アプリケーションの見かけの経過時間を基準とする（$dX/dt\ =\ 1$）と，使用しているコンポーネントの電力消費は大きくなる．

ⓒ 2013 Information Processing Society of Japan

$dP^l/dt = C^l \times Q(t)$

仮に，$Q(t)$ の時間による変化を無視して，最小値と最大値の区間に置き換えると，次のような関係式が得られることから，PCA は LHA と同等になる．

$C^l \leq dP^l/dt \leq C^l \times \alpha$

オートマトンの動作から考えると，最小値と最大値の間の任意の値を非決定的に選ぶことを表している．

もうひとつの方法として，$Q(t)$ が区間 $[1,\ \alpha]$ の値を確率的にとると考えても良いだろう．DVFS ガバナーによる CPU 特性の切り替え箇所を i でインデックスする時，切り替え箇所 i ごとに決まる確率変数 R_i の値に依存すると考える．

$dP^l/dt = C^l \times Q(R_i)$

ここで，R_i は，ある確率分布 $f(r)$ に対して，独立かつ一様分布 (i.i.d.) する確率変数である．この時，PCA は確率過程となる．

4. モデルベース解析

4.1 LHA の解析

本節では，線型ハイブリッド・オートマトン（LHA）に対するアルゴリズムベースの解析法を振り返る．解析法の基本的な特徴を説明するので，LHA ごとに対応するラベル付き遷移システム（LTS）が作り出す無限状態での到達性解析アルゴリズムを対象として考える．以下の説明は，文献 [5] による．

LHA は，時間オートマトン（TA），マルチレート時間システム（MTS），n レート時間システム（nRTS），ストップウォッチ・オートマトン（SWA) などを特殊ケースとして持つ．また，ロケーション不変量 (ϕ)，遷移ガード (ψ) が，変数 $x \in Var$ と整数定数 $k \in Z$ に対して，$x \leq k$ あるいは $k \leq x$ の形をしている時，LHA は シンプル という．

変数 x はすべてのロケーション l に対して $Act(l,x) = 1$，すべてのエッジ e に対して $\mu(e,x) \in \{0,x\}$ を満たす時，クロックと呼ぶ．スキュークロックはレート $k(k \in Z)$ で変化するクロックであり，$Act(l,x) = k$ と $\mu(e,x) \in \{0,x\}$ を満たす．k は 1 以外の整数値でも良い．

TA は変数がクロックであるようなシンプルな LHA である．MTS はクロック変数がスキュークロックであるような LHA である．nRTS は，あるスキュークロックが異なる n 種類のレートで変化する MTS である．SWA は，$Act(l,x) \in \{0,1\}$ かつ $\mu(e,x) \in \{0,x\}$ の TA である．

TA ならびにシンプルな MTS に対する到達性解析は決定可能であることが知られている．一方，シンプルという制限をつけたとしても，その他の場合，決定可能ではない．したがって，過大近似の方法で解析することになり，さまざまな近似アルゴリズムが考案されている．

4.2 PCA の解析

PCA に関する素朴な問題は，図 3 のような電力消費プロファイルを再現できるかである．このようなプロファイルは典型的な実行トレースから得られる．PCA の操作的な意味にしたがってテスト実行させれば，PCA の実行履歴から，電力消費プロファイルを得ることは可能と考えられる．

一方で，実行トレースに依存することを考えると，設計段階で行うモデルベースのデバッグ手法としては十分ではない．PCA を様々なテストシナリオのもとで解析すべきであり，記号モデル検査のような静的テストの方法が期待される．一方で，第 4.1 節でみたように，LHA を対象とする到達性検査の問題では，決定可能性の制限が強い．PCA が LHA のサブクラスであるとはしても，どのくらいの表現力を必要とするのかを詳しく調べなければならない．

今，M 個のパワー状態を持つ PCA を考える．各パワー状態で単位時間に消費する電力は異なるとする，$P(t)^j = C^j \times t$ $(j = 0 \ldots M-1)$．また，総電力消費量 (Pow) が，ある決められた値（Max）を超えないことを確認したいとする．検査する時相論理式は，$\Box(Pow \leq Max)$　のようになる．この性質を自動解析することを念頭において，PCA を，LHA のどのサブクラスとして表現できるか否かを考える．

M 個のパワー状態を持つ PCA は，M レート時間システムで表せる．$P(t)^j$ は時間 t の 1 次関数であるから，あるパワー状態 j での電力消費量は当該状態に留まる時間に比例する．スキュークロック Pow を導入し，パワー状態 j で，レート C^j の変化をさせれば良い．すなわち，スキュークロック Pow は M 通りのレートで進行し，各パワー状態での消費電力の総和を記録する．このようにすれば，PCA は M レート時間システムである．ただし，C^j は整数として近似する．

もうひとつの方法として，PCA を MTS と考える方法がある．状態 j での電力消費 $P(t)^j$ を固定レート C^j で進行するスキュークロック X^j で表す．一方，X^j は PCA が状態 j にいる時だけ時間を進めるようにしたい．つまり，それ以外の状態では，X^j を止めておきたい．したがって，PCA はマルチレートのストップウォッチ・オートマトンとなる．すなわち，PCA は MTS と SWA のいずれよりも表現力が大きいことがわかる．なお，総電力消費量は，$Pow = \sum_{j=0}^{M-1} X^j$ として計算すればよい．

第 4.1 節に述べたように，到達性検査の問題は，nRTS ならびにシンプル SWA に対して，決定不能である．したがって，たとえシンプルな PCA であっても，その到達性検査は決定不能である．したがって，自動検証を考える限り，何らかの過大近似の解析法を使わざるを得ない．

さらに，DVFS ガバナーの効果を考慮する場合，PCA のダイナミックスを拡張する必要があった（第 2.4 節）．非決定的な選択とする方法では，LHA と同等になる．ある

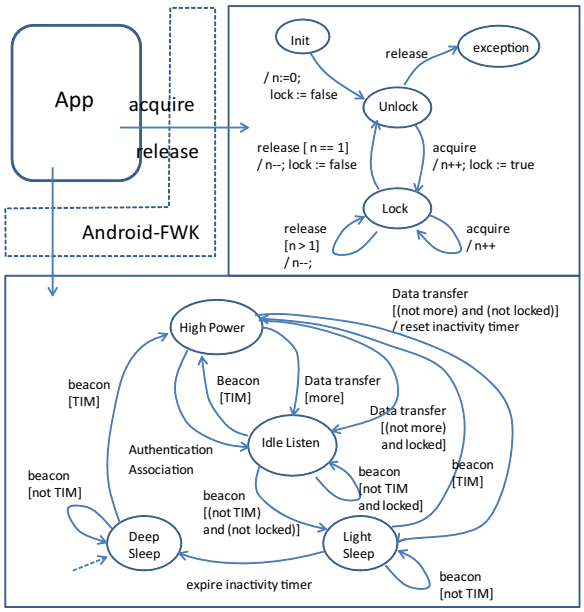

図 6 WiFi ロックの影響

いは，PCA を確率過程とする方法では，上記の解析手法は適用できない．統計モデル検査 [25] や統計的な実行時検証 [16] の方法を用いる．

5. 議論

5.1 アプリケーションの解析

最初の問題に戻り，アプリケーションの電力消費を具体例で考える．解析対象のイメージを図 6 に示した．ここで，解析対象アプリケーション `App` は，WiFi を用いた通信（たとえば，HTTP 通信）を行う．大量のデータ転送中，WiFi サブシステムをウェイク・ロックでアクティブに保つ．

図 6 右上が，`WiFiLock` クラスの状態遷移システムを表す．デフォールト設定のまま，レファレンス・カウント方式を用いるとする．アプリケーションが起動するロック獲得（`acquire()`），解放（`release()`）メソッドによって，内部状態が変化する．

図 6 の下部に，WiFi サブシステム（STA）の振る舞いを示した．基本的には，図 4 と同じであるが，`WiFiLock` の効果を加味してある．2 つのダイアグラムを比較すればわかるように，ロックされているか否かで，`no-more-flag` 時の High Power 状態からの遷移先が異なる．同様に，Idle Listen 状態で `no TIM` ビーコンを受け取った時の遷移先が異なることがわかる．

たとえば，`App` がロック解放メソッドを忘れていた場合，WiFi サブシステムは，より長い期間，電力消費が大きい状態に留まる．このような不具合の状況は，図 2 のライフサイクルで作動している `App` へのコールバック・メソッド系列の乱れから起こる可能性もある．最悪の場合，`App` がデストロイされた後でも，ウェイク・ロックがアクティブのまま残される．WiFi サブシステムが電力消費の直接原因であるが，もともとの原因（root causes）は `App` の設計ミスにある．

アンドロイド・アプリケーションの電力消費を解析するには，アプリケーションの実行に関わるリナックス拡張ドライバやアンドロイド・フレームワーク機能（アクティビティ管理，非同期タスクローダ，低メモリ・キラー等）を適切に抽象化して PCA として表現する必要がある．検査対象システム全体を，同期通信機構を持つ複数の PCA として表現することで，本稿が目的とするアプリケーション電力消費のモデルベース手法を実現できると期待できる．

5.2 利用ツールの考察

PCA の性質解析に用いるツールについて考察する．自動検査する場合，モデル検査ツールを使えば良い．

到達性が決定可能なクラスは TA であり，UPPAAL が代表例である．しかし，PCA は TA よりも表現力が大きい．文献 [17] ではストップウォッチ拡張 UPPAAL を使った実験を示している．SWA の到達性解析は，TA の記号モデル検査を巧妙に用いた効率の良い過大近似の方法 [8] による．一方，PCA は SWA よりも表現力が大きいので，HyTech[11] のような一般の LHA を取り扱える記号モデル検査ツールを用いる必要がある．

いずれであっても，近似せざるを得ないのであれば，時間有界な解析に限定されるが，サンプリング・ベースのモデル検査を用いる方法を考えても良い．電力消費の問題は時相論理式では，$\Box(Pow \leq Max)$ のように，安全性検査の形になる．しかし，並行システムのデッドロック解析等とは異なり，消費電力の計算開始時点から終了時点を決めて，その範囲での解析を行えば良い．無限の時間について積算すれば上限を明らかに超える．したがって，時間有界な解析で良い．

Realtime Maude[18] は，Maude [9] をハイブリッドシステム向けに拡張した形式仕様言語である．時間有界なサンプリング抽象の解析を基本とする．検査対象システムの構成要素に最大時間経過（maximum time elapsed）の情報を与えることで，サンプリング時点を明示する．サンプリング時点は mte の和集合になることから，システム全体としてはオーバーサンプリングになる．この方法で PCA を検査することができる [*3]．

モデル検査ツールによる自動検証と異なるアプローチとして，ハイブリッド・オートマトン（HA）を対象とする対話的な検証作業支援ツール STeP [14] や KeYmaera[23] がある．これらのツールの難しさは，対話的な支援ツールであることから，背景にあるロジカルシステムならびに証明技法の知識が必要なことにある．また，検証過程では，多くの場合，適切な不変量を発見することが重要であるが，

[*3] 実験内容は別稿で報告．

ⓒ 2013 Information Processing Society of Japan

人手まかせになっている．D.Ishii ら [12] は，Mathematica を用いることで，不変量発見を半自動化する検証支援の方法を提案した．

別の難しさとして，検証作業に先だって，完璧と考えられる記述を得ることが前提となる点がある．一般に，複雑な対象を適切に表現するには，繰り返し作業を必要とする．単純な記述から系統的に記述を追加する枠組みがあれば良い．このような検証作業を容易にする方法として，水平リファインメントを用いる方法がある．文献 [4] にある Event-B[3] を用いた方法論では，構築からの正しさ（Correct by Construction）の考え方を基にした方法を論じている．離散システムから出発し，連続系の性質をリファインメント技法で段階的に追加していく方法である．記号モデル検査の方法が，連続系を離散系に抽象化する考え方に基礎をおくのに対して，逆の方向にシステム機能を詳細化していく．

なお，モデル検査のような自動検証ツールを用いる場合，対象記述を検査しながら洗練していくという方法論が使える．自動化ツールの利点である軽量形式手法（Lightweight Formal Methods）による段階的な形式仕様作成法である．

最後に，確率的な取り扱いをするには，以上のツールは不十分である．一方で，検証処理の実行性能ならびにスケーラビリティという観点からは，記号モデル検査よりも，統計モデル検査の方法のほう良いことが知られている．統計モデル検査 [25] は時間有界な範囲での検査であるが，網羅的な探索に代えて，多数の検査列による結果から，統計的な処理によって性質を調べる方法である．検証対象のコンポーネントに確率的な要素を導入しなければならない．逆に，本質的に確率的な振る舞いを持つ対象の解析に適している．

5.3 関連研究

A. Pathak ら [21] は，スマートフォンでエネルギーバグを除去することの重要性を指摘し，このような不具合を *ebugs* と名付けた．また，電力消費のモデル化手法として状態遷移システムの考え方を導入した解析ツール Eprof を開発した [22]．Eprof はプログラム実行を監視する電力消費プロファイラであり，テスト実行の方法で電力バグを見つける．本稿がベースとした状態遷移システムの考え方による PCA は，A. Pathak らの研究 [20] に触発されたもので，文献 [17] で提案された．

MoVES [7] は組込みシステムを対象とするスケジュール可能性や電力消費の問題の表現ならびに解析法として，ストップウォッチ拡張された UPPAAL を用いている．しかし，電力消費モデルは，異なるパワー状態を区別することはなく，時間に比例する定数を用いて，$P(t) = C \times t$ のように計算する．ストップウォッチ・オートマトンで表現可能な性質に限定していることがわかる．

C. Thompson ら [24] は，モデル駆動エンジニアリングの方法で，電力消費の問題を取り扱っている．デザイン記述から電力消費を見積もるプログラムを自動生成し実行することで電力消費を予測する．D.-H. Kim ら [13] は電力消費の振る舞いを表現した UML シーケンス図を，シミュレーション実行することで電力消費を予測する．いずれも，実行による方法であり，本稿で考察した形式手法とは異なるアプローチである．

PCA のついての最初の報告 [17] では，PCA をストップウォッチ・オートマトン（SWA）で表し，ストップウォッチ拡張された UPPAAL で解析する方法を論じている．パワー状態にストップウォッチを割り当てるという方法であることから，各パワー状態での電力消費を独立に検査することになる．記号モデル検査のアルゴリズム上，複数クロックの総和は意味をなさないことから，全電力消費についての検査式 $A\Box((\sum_{PS} P^{PS}) \leq Max^{Total})$ は取り扱うことができない．$\bigwedge_{PS} A\Box((P^{PS} \leq Max^{PS})$ は検査可能であるが，これは，上記の検査式とは内容が異なる．

6. おわりに

スマートフォンの電力消費問題は新しい技術課題であり，さまざまな要素が絡み合うことから，問題の本質を把握することが難しい．実際，本稿の引用文献をみてもわかるように，幅広い技術に目を配る必要がある．現状では，ハードウェアや実行基盤からの課題抽出や解決アプローチの研究が多い．

本稿は，ソフトウェア工学の研究として問題を捉えなおした．モデルベースの方法を提案し，従来から知られている形式解析の技術の中に位置づけした．その中心は，電力消費オートマトンであるが，厳密な静的解析が困難であること，過大近似の方法で取り扱わざるを得ないことを論じた．第 5.1 節で説明したようなアプリケーションの具体例を，第 5.2 節に紹介したツールで実験し比較検討することも面白いだろう．

参考文献

[1] Android. `http://developer.android.com`.
[2] IEEE Standard 802.11, Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications , 1999.
[3] J.-R. Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University press. 2010.
[4] J.-R. Abrial, W. Su, and H. Zhu. Formalizing Hybrid Systems with Event-B. In *Proc. ABZ 2012*.
[5] 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.
[6] R. Alur, Formal Verification of Hybrid Systems, in *Proc. EMSOFT'11*, pp.273-278, 2011.
[7] A. Brekling, M.R. Hansen, and J. Madsen. MoVES – A

ⓒ 2013 Information Processing Society of Japan

Framework for Modeling and Verifying Embedded Systems, In *Proc. ICM2009*, pp.149-152, 2009.

[8] F. Cassez and K.G. Larsen. The Impressive Power of Stopwatches, In *Proc. CONCUR 2000*, pp.138-152, 2000.

[9] 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.

[10] J.L. Hennessy and D.A. Patterson. *Computer Architecture : A Quantitative Approach (5ed.)*, Morgan Kaufmann 2011.

[11] 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.

[12] D. Ishii, G. Melquiond, and S. Nakajima. Inductive Verification of Hybrid Automata with Strongest Postcondition Calculus, In *Proc. iFM 2013*, pp.139-153, 2013.

[13] D.-H. Kim, J.-P. Kim, and J.-E. Hong. A Power Consumption Analysis Techniques Using UML-Based Design Models in Embedded Software Development, In *Proc. SOFSEM 2011*, pp.320-331, 2011.

[14] Z. Manna, and H. Sipma. Deductive verification of hybrid systems using STeP. In *Proc. HSCC'98*. pp. 305-318, 1998.

[15] J. Manweiler and R.R. Choudhury. Avoiding the Rush Hours: WiFi Energy Management via Traffic Isolation, In *Proc. MobiSys'11*, 2011.

[16] S. Nakajima. Importance Sampling of Runtime Interference, In *Proc. APSEC 2012*, pp.693-696, 2012.

[17] S. Nakajima and Y. Ueda. Power Consumption Analysis of Smartphone Applications using UPPAAL, In *Proc. 1st CPSNA*, 2013.

[18] 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.

[19] V. Palipadi and A. Starikovskiy. The Ondemand Governor, In *Proc. Linux Symp. 2006*, 2006.

[20] A. Pathak, Y.C. Hu, M. Zhang, P. Bahl, and Y.-M. Wang. Fine-Grained Power Modeling for Smartphones Using System Call Tracing, In *Proc. EuroSys'11*, 2011.

[21] 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.

[22] 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.

[23] A. Platzer. *Logical Analysis of Hybrid Systems*. Springer, 2010.

[24] 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.

[25] P. Zuliani, C. Baier, and E.M. Clarke. Rare-Event Verification for Stochastic Hybrid Systems, In *Proc. HSCC 2012*, pp.217-225, 2012.

ⓒ 2013 Information Processing Society of Japan