UNIVERSIDADE FEDERAL DE PELOTAS
Centro de Desenvolvimento Tecnológico
Programa de Pós-Graduação em Computação
Mestrado em Ciência da Computação

Dissertação

Uma abordagem baseada em medição interna para avaliar eficiência energética de aplicativos Android

Leonardo Matthis Fischer

Pelotas, 2015

Leonardo Matthis Fischer

Uma abordagem baseada em medição interna para avaliar eficiência energética de aplicativos Android

Dissertação apresentada ao Programa de Pós Graduação em Computação do Centro de Desenvolvimento Tecnológico da Universidade Federal de Pelotas, como requisito parcial à obtenção do título de Mestre em Ciência da Computação.

Orientador: Lisane Brisolara de Brisolara
Coorientador: Julio Carlos Balzano de Mattos

Pelotas, 2015

Universidade Federal de Pelotas / Sistema de Bibliotecas
Catalogação na Publicação

F531a Fischer, Leonardo Matthis

Uma abordagem baseada em medição interna para avaliar eficiência energética de aplicativos android / Leonardo Matthis Fischer ; Lisane Brisolara de Brisolara, orientadora ; Julio Carlos Balzano de Mattos, coorientador. — Pelotas, 2015.

99 f.

Dissertação (Mestrado) — Programa de Pós-Graduação em Computação, Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas, 2015.

1. Dispositivos móveis. 2. Consumo de energia. 3. Perfil energético. 4. Potência. 5. Android. I. Brisolara, Lisane Brisolara de, orient. II. Mattos, Julio Carlos Balzano de, coorient. III. Título.

CDD : 005

Elaborada por Aline Herbstrith Batista CRB: 10/1737

Leonardo Matthis Fischer

Uma abordagem baseada em medição interna para avaliar eficiência energética de aplicativos Android

Dissertação aprovada como requisito parcial para obtenção do grau de Mestre em Ciência da Computação, Programa de Pós Graduação em Computação do Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas.

Data da defesa: 20 de março de 2015

Banca examinadora:

Prof. Dr. Mateus Beck Rutzig
Doutor em Computação pela Universidade Federal do Rio Grande do Sul (UFRGS)

Prof. Dr. Denis Teixeira Franco
Doutor em Communications et Electronique pela Universidade Ecole Nationale Superieure des Telecommunications, ENST-Paris, França

Prof. Dr. Leomar Soares da Rosa Junior
Doutor em Microeletrônica pela Universidade Federal do Rio Grande do Sul (UFRGS)

Agradecimentos

Agradeço primeiramente a Deus, pelas bênçãos que derrama sobre minha vida.

A meus pais, Jorge e Beatriz por sempre motivarem meus estudos e capacitação.

A minha esposa Cíntia, pelo companheirismo e paciência comigo durante esta etapa da minha formação.

Ao meu irmão Gustavo, pela parceria, pelas conversas sérias e pelas bobagens.

A minha orientadora Lisane e ao meu coorientador Júlio pelo apoio e orientação ao longo de todo mestrado.

Aos colegas do IFSul Santana do Livramento.

E a todos os amigos que contribuíram de forma direta ou indireta para a conclusão dessa etapa, meu muito obrigado!

“Um pouco de ciência nos afasta de Deus.
Muito, nos aproxima."
Louis Pasteur

Resumo

FISCHER, Leonardo Uma abordagem baseada em medição interna para avaliar eficiência energética de aplicativos Android. 2015. 99f. Trabalho acadêmico (Mestrado) - Mestrado em Ciência da Computação. Universidade Federal de Pelotas, Pelotas.

O aumento da complexidade das aplicações em dispositivos móveis como smartphones e tablets faz com que esses necessitem de mais recursos computacionais resultando em um maior consumo de energia. A preocupação com consumo é agravada pela limitação na evolução das baterias. Para reduzir o consumo energético, é necessário identificar gargalos do sistema e para isso são requeridas ferramentas de avaliação do consumo energético. Grande parte das ferramentas atualmente disponíveis empregam modelos do hardware para estimar o consumo energético, seja em um dispositivo físico ou em simuladores. Para que forneçam resultados precisos, são requeridos modelos com elevado nível de detalhamento e específicos a uma plataforma de hardware. Por outro lado, técnicas de medição, embora mais precisas tem utilizado instrumentação externa, as quais não são acessíveis e nem práticas para desenvolvedores e usuários finais. Existe, portanto, a necessidade de desenvolver ferramentas de avaliação que sejam flexíveis, ou seja, não sejam dependentes de um hardware específico e que possam ser empregadas por desenvolvedores e usuários. Este trabalho propõe uma nova abordagem para estimar consumo, a qual baseia-se em técnicas de medição de potência/energia a partir de amostras de tensão e corrente extraídas do driver da bateria do próprio dispositivo. Essa abordagem é capaz de estimar o consumo energético de um aplicativo Android, ou ainda de partes de seu código. Os resultados obtidos por esse método indicam que a ferramenta mantém uma coerência dos valores medidos em relação a outras abordagens.

Palavras Chave: Dispositivos Móveis; Consumo de Energia; Perfil Energético; Potência; Android; Medição.

Abstract

FISCHER, Leonardo M. An approach based on internal measurement to evaluate energy efficiency of Android applications – 2015, 99p. Mestrado em Ciência da Computação. Universidade Federal de Pelotas, Pelotas.

The increase of application complexity on mobile devices like smartphones and tablets demands more computational resources consuming more energy. The concern with consumption is aggravated by slow and limited development of batteries. To reduce energy consumption is necessary to identify system bottlenecks and for that, tools are required to evaluate energy consumption. Most of the available tools employ hardware models to estimate energy consumption, whether in a physical device or simulator. For providing accurate results, models at high level of detail and specific to a hardware platform are required. On the other hand, measurement techniques, although more accurate have used external instrumentation, which are not accessible and not practical for developers and end users. Therefore, flexible tools should be developed, which should be independent of a specific hardware and useful for developers and end-users. This work proposes a new approach to estimate consumption, which is based on power measurement techniques using samples of voltage and current extracted from the battery of the device driver itself. This approach is able to measure the energy consumption of an Android application, or parts of its code. Results achieved by this method show that it maintains a consistency of measured values in relation to other approaches.

Keywords: Mobile Devices; Energy Consumption; Energy Profile; Power, Android; Measurement

Lista de Figuras

Lista de Equações

Lista de Tabelas

Lista de Abreviaturas e Siglas

AC	*Alternate Current*
API	*Application Programming Interface*
ART	Android RunTime
BMU	*Battery Monitoring Unit*
CMOS	*Complementary Metal-Oxide-Semiconductor*
CPU	*Central Processing Unit*
DAQ	*Data Acquisition*
DDMS	*Dalvik Debug Monitor System*
FSM	*Finite State Machine*
GPIO	*General Purpose Input/Output*
GPS	*Global Position System*
GUI	*Graphical User Interface*
HAL	*Hardware Abstraction Layer*
I/O	*Input/Output* (Entrada e Saída)
JIT	*Just In Time*
LCD	*Liquid Crystal Display*
PID	*Process IDentifier*
SBM	Mapeamento Baseado em Amostras
SEMA	*Self Energy Metering for Android*
SO	Sistema Operacional
SSL	*Secure Socket Layer*
TBM	*Tracing Based Method*
VM	*Virtual Machine*
VNC	*Virtual Network Connection*

Sumário

1 INTRODUÇÃO

Os *smartphones* e *tablets* atuais tem permitido que as pessoas carreguem consigo mais recursos computacionais que os *desktops* de alguns anos atrás. O mercado de dispositivos móveis vem crescendo, pois além dos recursos computacionais disponíveis, estes oferecem e agregam várias funcionalidades como câmera, relógio, agenda, bem como permitem uma alta conectividade. Segundo Cisco (CISCO, 2015) o crescimento do número de dispositivos móveis no mundo, em 2014, foi de 497 milhões, totalizando 7,4 bilhões de dispositivos móveis conectados. Este mercado vem se destacando, o que motivou o lançamento de diversas plataformas móveis, as quais procuram se diversificar com melhor usabilidade, mais recursos ou ainda recursos mais eficientes. Isso tem exigido dos projetistas um esforço cada vez maior no desenvolvimento de plataformas eficientes tanto em desempenho quanto em consumo energético e dissipação de potência.

O consumo de energia tem sido enfrentado como um fator limitante às funcionalidades que podem ser oferecidas através de plataformas móveis. Isso se dá principalmente pelo fato desses dispositivos serem alimentados através de baterias. Segundo Shearer (2008) e Newman (2013), a tecnologia utilizada no desenvolvimento de baterias teve um crescimento lento se comparado ao crescimento observado na quantidade de recursos agregados aos dispositivos móveis, bem como aceleração dos mesmos. A figura 1 ilustra as duas curvas de crescimento, onde pode-se observar esta lacuna. Além disso, existe a exigência de dispositivos mais leves e menores, reduzindo também o espaço disponível para baterias e aumentando a preocupação com consumo energético destes equipamentos.

O consumo energético é, portanto, um fator de fundamental importância no projeto de plataformas móveis. Prolongar a vida útil de baterias é um desafio onde os recursos computacionais aumentam a taxas maiores que a capacidade de armazenamento de energia, gerando uma lacuna cada vez maior entre eles.

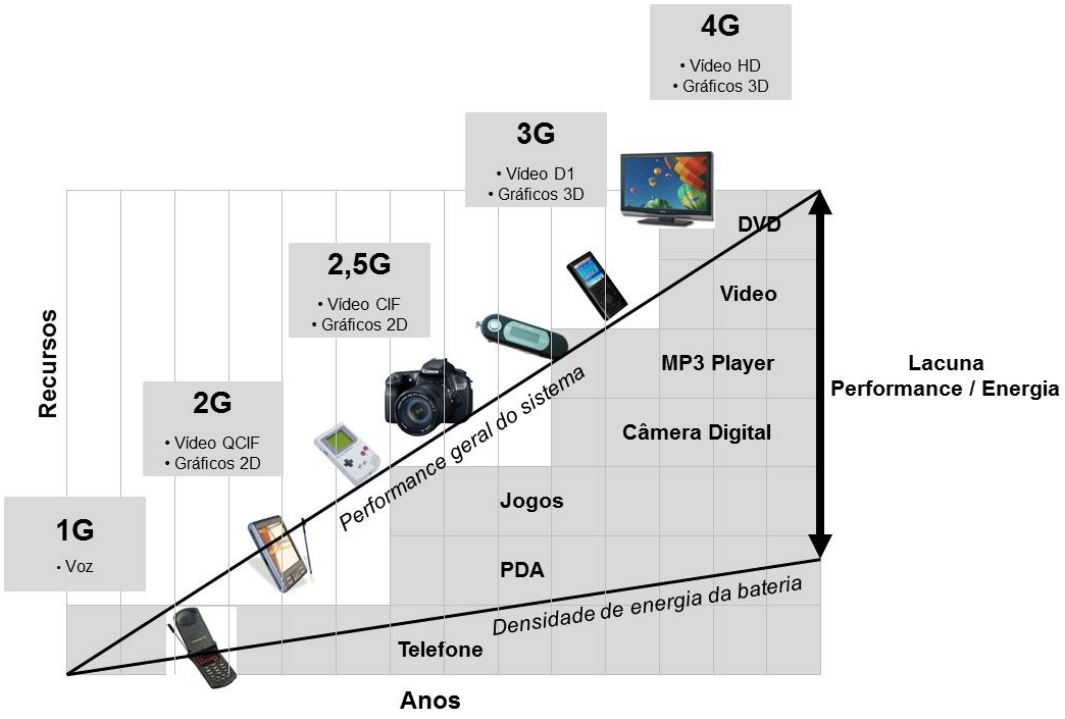

Figura 1: Lacuna entre Performance e Energia
Fonte: Adaptado de (SHEARER, 2008)

Neste cenário, a eficiência energética é um aspecto chave no projeto de dispositivos móveis, bem como no desenvolvimento de sistemas operacionais (SOs) e de aplicativos para os mesmos. Por exemplo, em Tonini et al. (2013), a preocupação com a eficiência energética de aplicativos Android é discutida e experimentos apontam que o uso de boas práticas de programação podem trazer melhorias.

A preocupação com o consumo motiva o estudo de métricas para avaliar desempenho, potência e consumo de energia, bem como o estudo das relações entre estas métricas. Alto desempenho com alto consumo de potência não significa necessariamente pequena eficiência energética, da mesma forma que baixo desempenho e baixo consumo de potência pode não significar que o sistema é eficiente energeticamente.

A obtenção de dados precisos sobre essas métricas é importante na busca do equilíbrio entre elas, além de permitir a identificação de pontos críticos para eficiência, onde otimizações devem ser adotadas (SNOWDON, PETTERS e HEISER, 2005). A obtenção destas métricas requer o desenvolvimento de ferramentas capazes de analisar e estimar o consumo de potência e energia destes dispositivos. Estas ferramentas podem ser usadas, por exemplo, para gerar um perfil do consumo

energético que possa ser utilizado como base para o estudo de otimizações no sistema. Inúmeros trabalhos têm sido propostos para atacar o problema em diferentes plataformas e com diferentes níveis de detalhe e precisão. Dentre eles destacam-se os trabalhos de Zhang et al. (2010), Dong e Zong (2011), Pathak, Hu e Zhang (2012), Yoon (2012) e Wilke, Gotz e Richly (2013) onde são propostas as abordagens PowerTutor, SeSaMe, Eprof, AppScope e JouleUnit, respectivamente.

As abordagens e ferramentas existentes podem ser classificadas em baseadas em modelos e baseadas em medição (TSAO e CHEN, 2012), (CHUNG, LIN e KING, 2011). Ferramentas baseadas em modelos dependem de um modelo bastante detalhado para gerar respostas precisas. Contudo, gerar modelos com elevado nível de detalhe pode ser uma tarefa custosa e demorada. Por outro lado, ferramentas baseadas em medição tendem a oferecer maior precisão, porém requerem a instrumentação do sistema com multímetros, osciloscópios, etc. Além disso, a sincronia entre as amostras de consumo coletadas e os eventos ocorridos no sistema é um desafio para quem propõe ferramentas desse tipo.

A plataforma Android é atualmente a líder em números de vendas de *smartphones* e *tablets, contando atualmente com 84,4% desse mercado (IDC, 2014)*. Atualmente, o Android é mantido por um grupo de empresa do ramo da tecnologia, chamado Open Handset Alliance, e tem como líder a Google. Por ser de código aberto, tem atraído atenção de desenvolvedores e pesquisadores, uma vez que permite customizações. A plataforma é composta por uma pilha de *software*, a qual contém, entre outras coisas, recursos oferecidos pelo driver da bateria que podem ser utilizados para análise da eficiência dos aplicativos executados sobre ela.

As ferramentas e abordagens existentes atualmente para avaliação de consumo energético possuem limitações, seja por utilizarem modelos específicos a uma plataforma de hardware, por necessitarem permissões especiais no SO, ou por exigirem uso restrito a laboratórios com equipamentos e ferramentas de instrumentação. Essas restrições tem dificultado a avaliação da eficiência energética por parte de desenvolvedores, os quais necessitam ferramentas capazes de identificar o impacto de suas otimizações no código, sem contudo estarem limitados a um modelo de dispositivo ou ambiente laboratorial.

Por outro lado, plataformas móveis atuais proveem recursos para gerência da bateria, fornecendo informações como a tensão que a alimenta o dispositivo e a corrente que circula por ele, motivando o emprego de dados extraídos da plataforma

na estimativa de consumo energético, sem a necessidade de utilizar modelos ou instrumentação. Esta estratégia pode ser utilizada em diferentes dispositivos e sem a necessidade de equipamentos especiais, facilitando seu uso por parte de desenvolvedores e contribuindo para o desenvolvimente de aplicativos eficientes energeticamente.

Este trabalho propõe o emprego de uma abordagem para medição de consumo energético, denominada SEMA (*Self Energy Metering for Android*), a qual utiliza recursos do próprio dispositivo, através do Sistema Operacional (SO) e de suas camadas de *software*. Além disso, esta abordagem foi implementada na forma de uma ferramenta que coleta amostras de consumo de potência e estima o consumo energético do dispositivo Android quando roda um dado aplicativo. Através de experimentos, o emprego da abordagem foi demonstrado e sua precisão é discutida, e comparada a outras técnicas de avaliação de consumo.

Este trabalho divide-se em seis capítulos. No segundo capítulo, é apresentada a fundamentação teórica do trabalho, abordando conceitos de potência e energia, revisa técnicas utilizadas para avaliação de consumo energético e conceitos relevantes sobre a plataforma Android. O terceiro capítulo discute os trabalhos relacionados que representam o estado-da-arte na área explorada. O quarto capítulo detalha a abordagem proposta neste trabalho para estimativa de consumo de aplicativos Android, explicando em que técnica esta se baseia e detalha também sua implementação e emprego. Resultados experimentais obtidos pela abordagem são apresentados e discutidos no quinto capítulo, bem como comparações com outras abordagens e uma avaliação de precisão. Por fim, o sexto capítulo conclui o trabalho e discute trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo tem por objetivo apresentar a base teórica deste trabalho. Primeiramente, são revisados conceitos fundamentais sobre potência e energia, e a forma como estas são calculadas em circuitos eletrônicos, bem como as técnicas utilizadas na avaliação de consumo energético em sistemas embarcados. Por fim, este capítulo inclui uma visão geral da plataforma Android e detalha alguns de seus componentes relevantes para a compreensão do trabalho.

2.1 Consumo Energético: Conceitos

A potência dissipada por um circuito eletrônico é chamada de potência elétrica, e é medida em Watts. Entende-se por potência elétrica, o produto da corrente elétrica que circula pelo circuito, pela tensão de alimentação a que este é submetida. A equação 1 representa essa relação, onde P representa a potência medida em Watts (W), V representa a tensão, medida em Volts (V) e I representa a corrente elétrica medida em Ampères (A). Já energia elétrica, é a capacidade de uma corrente elétrica realizar trabalho, e está relacionada ao conceito de potência através das equações 2 e 3.

$$\boldsymbol{P = V * I}$$

Equação 1: Potência Elétrica

$$\boldsymbol{Energia = Potência * Tempo} \text{ (}\textit{Joules}\text{)}$$

Equação 2: Definição da Energia com base na Potência

$$\boldsymbol{Energia\ Total = \int Potência\ Total\ dt}$$

Equação 3: Definição da Energia Total

A figura 2 ilustra a diferença entre estas duas grandezas. Enquanto a potência é representada pela amplitude da curva do gráfico, a energia é a área definida pela curva, ou seja, a integral da potência sobre o tempo, conforme definido na equação 3 (KEATING *et al.*, 2007)

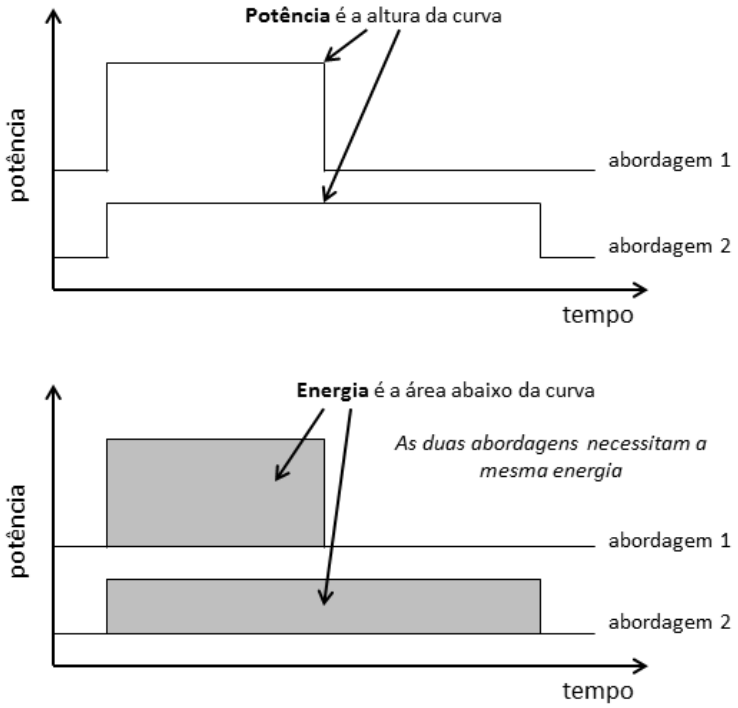

Figura 2: Potência vs. Energia

2.2 Estimativa de Consumo Energético

Existem duas características que são essencialmente levadas em consideração no projeto de ferramentas de avaliação de consumo energético: precisão e granularidade. Ferramentas de baixa precisão ainda que forneçam resultados que não refletem diretamente o valor real, podem ser utilizadas para verificar algum comportamento ou tendência. Ainda assim, é importante buscar aproximar o resultado da estimativa ao valor real, por isso, a busca pela precisão é constante, porém trabalhosa.

Granularidade diz respeito à quantidade de detalhes considerados na representação do sistema e na análise, ou seja, o nível de abstração adotado. Ferramentas flexíveis permitem que o usuário ajuste o nível de granularidade de acordo com sua necessidade. Aliar precisão e granularidade é um dos principais desafios enfrentados no desenvolvimento de ferramentas de análise de consumo (CHUNG, LIN e KING, 2011).

Também é desejado que as ferramentas não causem perdas de desempenho ou consumam muitos recursos do sistema, interferindo na análise. Portanto, a sobrecarga causada por uma ferramenta também é uma métrica essencialmente

importante. Ferramentas com elevado *overhead* afetam o desempenho do sistema de forma indesejada, alocando recursos para processos que deveriam apenas “observar” o comportamento do sistema sem afetá-lo. Contudo, obter resultados precisos e detalhados costuma causar maior *overhead*, uma vez que envolve análises mais complexas. Assim, é necessário encontrar um equilíbrio entre essas duas importantes características quando da escolha por uma dada ferramenta ou técnica.

O resultado de uma análise de consumo energético pode ser obtido imediatamente ou posteriormente ao término da execução do *software.* Em técnicas *online* a análise é realizada em tempo de execução, e o resultado final é obtido imediatamente ao final da execução. Já a análise *off-line* extrai dados em tempo de execução, porém a análise é feita somente após o final da execução, e o resultado apresentado após a análise. Para efetuar uma análise de consumo energético, diferentes técnicas têm sido propostas. Elas podem ser classificadas basicamente em: técnicas baseadas em modelos e técnicas baseadas em medição (TSAO e CHEN, 2012) (CHUNG, LIN e KING, 2011). As seções 2.2.1 e 2.2.2 apresentam essas técnicas.

2.2.1 Técnicas Baseadas em Modelos

As técnicas baseadas em modelos, como o nome sugere, utilizam como base para avaliação do consumo energético, um modelo de potência do sistema. Um modelo relaciona amostras de potência com blocos funcionais, instruções ou funções do sistema, e é através dele que as estimativas são calculadas.

Quanto ao nível de abstração, as técnicas baseadas em modelos podem ser classificadas em:

- **Técnicas de baixo nível**: levam em consideração modelos de potência em nível de arquiteturas ou instruções para gerar estimativas do consumo energético através de simulação ou emulação do *software*.
- **Técnicas de alto nível**: estimam o consumo energético do *software* através de blocos básicos ou funções. Nessa abordagem, o consumo de potência dos blocos básicos ou funções é determinado através de medição direta ou modelos de baixo nível, gerando uma tabela de custos. O *software* executa diretamente na plataforma alvo, e uma ferramenta contabiliza os acessos a esses blocos ou funções, gerando

um perfil (*profile*) energético, que combinado com os custos permite a estimativa do consumo. Através dessa técnica, um equilíbrio entre precisão e sobrecarga é estabelecido.

Para apresentar resultados precisos, os modelos devem possuir bons níveis de detalhe, porém agregar detalhe a um modelo é uma tarefa custosa. Além disso, quanto mais detalhado for o modelo, mais tempo este requer para ser processado, o que pode causar uma sobrecarga muito grande ao sistema.

A geração de um modelo pode ser feita de diferentes formas. A mais comum delas, é obter um modelo a partir de medição direta. Outra forma é gerar o modelo com base em uma técnica de nível mais baixo de abstração, ou ainda, através de simulação do sistema. Recentemente, técnicas utilizam recursos presentes no próprio *hardware* do sistema para efetuar auto geração do modelo, por exemplo, o *fuel gauge* da bateria.

Diferentes abordagens para estimativa baseada em modelo têm sido propostas. Dentre estas, destacam-se o uso de simuladores, ou ainda o emprego de técnicas baseadas na contagem de eventos ou ainda baseadas em estados.

Simuladores são ferramentas capazes de reproduzir o comportamento de um sistema, que é objeto de estudo. Existem simuladores em nível de ciclos de relógio, arquiteturas, instruções e até mesmo ferramentas que simulam o sistema completo, incluindo *hardware* e *software*. Por exemplo, um simulador a nível de instrução, permite analisar um relatório da execução das instruções, que combinado com uma caracterização de consumo destas, permite estimar o consumo do sistema.

Técnicas baseadas na contagem de eventos usam dados reais gerados por ferramentas de análise de desempenho como entrada para um modelo de consumo. Os contadores são configurados para medir os eventos que são significativos para o consumo de energia, e o modelo interpreta esses resultados para estimar a potência consumida pela CPU. Esses sistemas podem ser utilizados de forma eficiente em análises *online*.

Por fim, técnicas baseadas em estados instrumentam o sistema operacional, de forma que o *software* possa rastrear o estado da CPU e dos periféricos. A potência de cada um desses estados deve ser conhecida, e combinada com o tempo gasto em cada estado, determina-se a energia total consumida. Máquinas de Estados Finitos (FSM, do inglês, *Finite State Machine*) são comumente utilizadas neste tipo de

estratégia. Cada estado da FSM corresponde a um estado do sistema e o consumo de potência deste.

2.2.2 Técnicas Baseadas em Medição

Nesses modelos, o consumo de potência do sistema é avaliado através de ferramentas de instrumentação como multímetros ou osciloscópios. Enquanto um *software* de *profile* coleta informação no sistema alvo, outro sistema instrumentado adquire amostras do consumo de potência do sistema. Essas informações são associadas de forma a gerar o perfil de consumo energético do sistema.

Uma arquitetura clássica desse tipo de ferramenta é apresentada na figura 3. O sistema alvo é instrumentado por um registrador de atividades que coleta eventos importantes ocorridos no sistema, enquanto os componentes de *hardware* são conectados a uma placa de aquisição de dados no sistema monitor. Através da placa de aquisição de dados, são coletadas amostras de potência, as quais juntamente com o registro das atividades do sistema, alimentam o algoritmo de análise. O algoritmo de análise por sua vez, processa os dados recebidos e gera o perfil de consumo energético do sistema alvo.

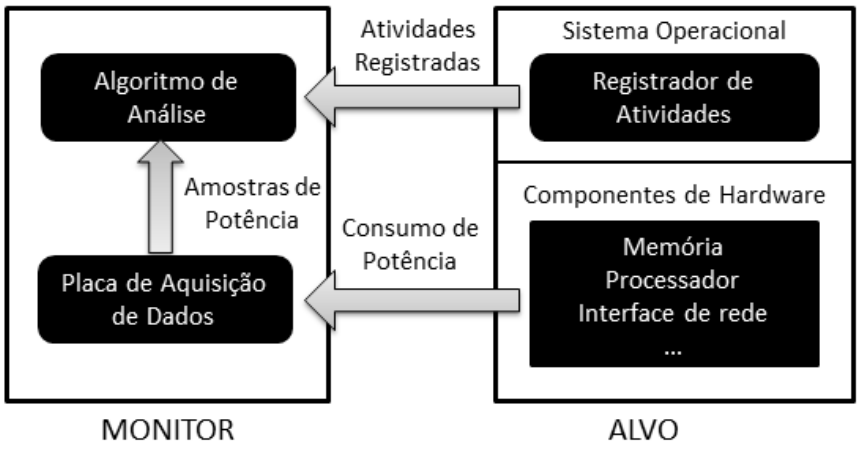

Figura 3: Arquitetura de uma ferramenta de medição
Fonte: Adaptado de (XIAN, CAI e LU, 2007)

Essa correlação entre as amostras de consumo e os eventos coletados pela ferramenta de *profile* é um problema recorrente em métodos baseados em medição. Duas formas principais são propostas para solucionar esse problema de mapeamento. A primeira delas é conhecida como Mapeamento Baseado em Amostras (SBM), no

qual amostras do consumo de potência do sistema são coletadas periodicamente. Quando uma amostra é coletada, um sinal é enviado ao sistema alvo, o qual grava as informações do processo que está executado no momento. Assim, é possível mapear amostras de potência aos processos executados pelo sistema (FLINN e SATYANARAYANAN, 1999).

No Mapeamento Baseado em *Tracing* (TBM, do inglês, *Tracing Based Method*), o escalonador do Sistema Operacional é instrumentado de forma a armazenar informações temporais do sistema e o identificador do processo em execução, conhecido como PID (*Process Identifier*) toda a vez que ocorre uma mudança de contexto. Assim, informações temporais do sistema são usadas para mapear as amostras de potência com um processo individual (XIAN, CAI e LU, 2007) (CHUNG, LIN e KING, 2011).

A técnica de SBM pode apresentar perdas quando, por exemplo, o período de amostragem é maior que o intervalo de troca de contexto dos processos. Como consequência, alguns processos podem ter apenas uma ou mesmo nenhuma amostra de potência coletada, acarretando em resultados imprecisos. Por outro lado, a TBM armazena o estado do sistema quando mudanças importantes ocorrem, assim, eventos importantes não são perdidos, contudo causa maior *overhead* devido as alterações no SO (CHUNG, LIN e KING, 2011).

A sincronização de relógio entre o monitor e o sistema alvo também é um problema a ser tratado pelas ferramentas baseadas em medição. Se essas máquinas forem heterogêneas e não estiverem devidamente sincronizadas, o *log* gerado pelo *profiling* vai ser incorretamente associado às amostras de potência. Existe ainda o atraso ao alinhar as atividades do sistema e as amostras de potência.

2.3 Plataforma Android

O Android é um sistema operacional móvel, mantido pela Open Handset Alliance, um grupo formado por várias empresas do ramo da tecnologia, tendo como líder a Google. Suas primeiras versões surgiram em 2008, enquanto sua versão mais popular, chamada GingerBread (2.3) surgiu em 2010 (MITROFF, 2014).

Trata-se de uma plataforma *open source* (ANDROID, 2014c), o que atrai os fabricantes de dispositivos, pois permite que cada um faça suas próprias personalizações sem, contudo, perder o suporte a um sistema operacional completo baseado no *kernel* Linux. Para os usuários, o que chama atenção é a interface com

boa usabilidade, atualizações frequentes e surgimento constante de novos aparelhos com *hardware* cada vez mais robusto utilizando a plataforma. Em função disso, o Android é o líder entre os sistemas operacionais utilizados em *smartphones* e *tablets*, contando com a significativa fatia de 84,4 % do mercado global (IDC, 2014).

A popularidade aliada à possibilidade de realizar alterações no sistema tornou o Android popular também em pesquisas que abordam plataformas móveis. Nesse trabalho, o foco é a avaliação de consumo energético em dispositivos móveis que utilizam essa plataforma. Desta forma, além de apresentar a pilha de *software* adotada por esta plataforma, bem como o ciclo de execução de aplicações Android, o capítulo revisa componentes do Android que gerenciam e que permitem coletar informações sobre a bateria do dispositivo.

2.3.1 Pilha de Software

O Android é formado por uma pilha de *software* (ANDROID, 2014n) composta por cinco camadas, conforme mostra a figura 4. Cada camada dessa pilha contém programas, bibliotecas, *drivers*, entre outros recursos, a fim de suportar as funções do sistema operacional.

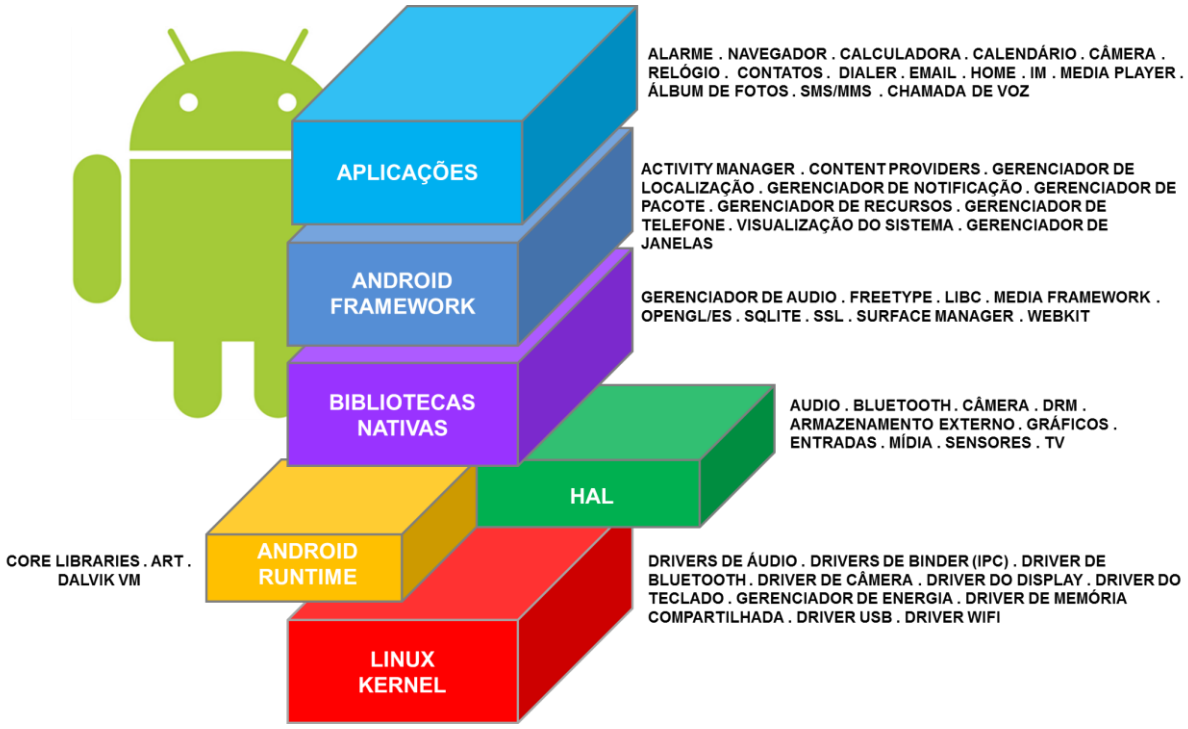

Figura 4: Pilha de *Software* do Android
Fonte: Adaptado de (ANDROID, 2014n)

A base da pilha é o *kernel* Linux, o qual provê vários serviços essenciais, como segurança, rede, gerenciamento de memória e processos, gerenciamento de energia (ANDROID, 2014d). Acima do *kernel*, encontra-se o Android Runtime, que provê componentes chave para execução e desenvolvimento de aplicativos na linguagem Java, dentre eles a máquina virtual (VM, do inglês, *Virtual Machine*) Java e as *Core Libraries*.

Duas VMs foram definidas para a plataforma Android: a Dalvik VM e a recentemente lançada ART (Android *RunTime*). A Dalvik é uma máquina virtual Java otimizada para uso em Android, e utiliza compilação *Just-In-Time,* ou seja, os *bytecodes* são traduzidos durante sua execução*.* A ART, por sua vez, utiliza compilação *Ahead-Of-Time,* ou seja, os aplicativos são pré-compilados durante a instalação. Através do emprego da ART e de outras otimizações, a Google defende que os aplicativos podem executar até duas vezes mais rápido (ANDROID, 2014j).

Para o desenvolvimento de aplicações as *Core Libraries* oferecem um conjunto de bibliotecas que permitem aos desenvolvedores criar aplicativos Android utilizando a linguagem Java padrão.

No mesmo nível do Android *Runtime*, está Camada de Abstração de *Hardware* (*Hardware Abstraction Layer* – HAL) (ANDROID, 2014i). Essa camada provê uma interface entre as APIs de *software* e os drivers do *hardware*.

As bibliotecas nativas estão localizadas acima do Android Runtime e da Camada de Abstração de *Hardware*. Alguns exemplos de bibliotecas, utilizadas por diversos componentes do sistema, são a libc, uma base de dados SQLite utilizada para armazenagem e compartilhamento de dados entre as aplicações, bibliotecas para executar e gravar áudio e vídeo, bibliotecas SSL para segurança de dados na internet, entre outras.

A camada de *Framework* contém as classes que são utilizadas para criar aplicativos Android. Essas classes são responsáveis por gerenciar o ciclo das aplicações, como é o caso da *Activity Manager*, bem como armazenar e recuperar dados, tornando-os acessíveis a todas aplicações, função dada ao *Contents Provider*. O *Framework* inclui ainda classes para gerenciar os serviços de localização (*Location Manager*), executar e gerenciar notificações e alertas (*Notification Manager*), entre outros.

No topo da pilha estão os aplicativos propriamente ditos. Nessa camada eles são executados fazendo uso dos recursos providos pelas camadas inferiores, de acordo com a especificação de cada aplicativo.

2.3.2 Bateria: Fuel Gauge e classe BatteryManager

A bateria dos dispositivos móveis baseados em Android é controlada por um circuito chamado *Fuel Gauge* (ANDROID, 2014l). Esse circuito pode fornecer informações de status da bateria, como porcentagem de carga, status carregando/descarregando, temperatura, entre outras.

Essas informações podem ser acessadas diretamente do *driver* do *Fuel Gauge*, através das variáveis contidas em */sys/class/power_supply/battery*, no caso de dispositivos alimentados por baterias, ou */sys/class/power_supply/ac* em dispositivos alimentados por fonte de alimentação. As informações disponíveis podem variar de acordo com o modelo do dispositivo. O arquivo que mantém um compilado de todas informações disponíveis bem como seus valores atualizados chama-se *uevent*, e está localizado no diretório citado anteriormente. A tabela 1 contém um exemplo das informações disponíveis para uma placa de desenvolvimento *Cubieboard A20 e para* um *smartphone* Motorola Moto G 2ª geração.

Tabela 1: Variáveis de Status da Fonte de Alimentação/Bateria obtidos com a placa Cubieboard vs smartphone Moto G

	Dispositivo	
	Cubieboard A20	**Motorola Moto G 2ª Geração**
Dados Disponíveis	POWER_SUPPLY_NAME	POWER_SUPPLY_NAME
	POWER_SUPPLY_MODEL_NAME	POWER_SUPPLY_CHARGING_ENABLED
	POWER_SUPPLY_STATUS	POWER_SUPPLY_STATUS
	POWER_SUPPLY_PRESENT	POWER_SUPPLY_CHARGE_TYPE
	POWER_SUPPLY_ONLINE	POWER_SUPPLY_HEALTH
	POWER_SUPPLY_HEALTH	POWER_SUPPLY_PRESENT
	POWER_SUPPLY_TECHNOLOGY	POWER_SUPPLY_ONLINE
	POWER_SUPPLY_VOLTAGE_MAX_DESIGN	POWER_SUPPLY_TECHNOLOGY
	POWER_SUPPLY_VOLTAGE_MIN_DESIGN	POWER_SUPPLY_VOLTAGE_MAX_DESIGN
	POWER_SUPPLY_VOLTAGE_NOW	POWER_SUPPLY_VOLTAGE_MIN_DESIGN
	POWER_SUPPLY_CURRENT_NOW	POWER_SUPPLY_VOLTAGE_NOW
	POWER_SUPPLY_ENERGY_FULL_DESIGN	POWER_SUPPLY_CAPACITY
	POWER_SUPPLY_CAPACITY	POWER_SUPPLY_CURRENT_NOW
	POWER_SUPPLY_TEMP	POWER_SUPPLY_INPUT_CURRENT_MAX
		POWER_SUPPLY_INPUT_CURRENT_TRIM
		POWER_SUPPLY_INPUT_CURRENT_SETTLED
		POWER_SUPPLY_VOLTAGE_MIN
		POWER_SUPPLY_INPUT_VOLTAGE_REGULATION
		POWER_SUPPLY_CHARGE_FULL_DESIGN
		POWER_SUPPLY_CHARGE_FULL
		POWER_SUPPLY_TEMP
		POWER_SUPPLY_TEMP_COOL
		POWER_SUPPLY_TEMP_WARM
		POWER_SUPPLY_SYSTEM_TEMP_LEVEL
		POWER_SUPPLY_NUM_SYSTEM_TEMP_LEVELS
		POWER_SUPPLY_CYCLE_COUNT
		POWER_SUPPLY_VOLTAGE_OCV
		POWER_SUPPLY_CHARGE_COUNTER

Outra forma de obter informações do estado da bateria de dispositivos Android é através da classe *BatteryManager* (*android.os.BatteryManager*) (ANDROID, 2014f). Essa classe oferece uma interface padrão para acesso a essas informações, enquanto no método apresentado anteriormente, o nome da variável pode sofrer alterações de acordo com o fabricante. A tabela 2 contém as variáveis que podem ser obtidas através da classe *BatteryManager*:

Tabela 2: Variáveis de status disponíveis através da classe BatteryManager

Variáveis de status disponíveis através da classe BatteryManager
BATTERY_HEALTH_COLD
BATTERY_HEALTH_DEAD
BATTERY_HEALTH_GOOD
BATTERY_HEALTH_OVERHEAT
BATTERY_HEALTH_OVER_VOLTAGE
BATTERY_HEALTH_UNKNOWN
BATTERY_HEALTH_UNSPECIFIED_FAILURE
BATTERY_PLUGGED_AC
BATTERY_PLUGGED_USB
BATTERY_PLUGGED_WIRELESS
BATTERY_PROPERTY_CAPACITY
BATTERY_PROPERTY_CHARGE_COUNTER
BATTERY_PROPERTY_CURRENT_AVERAGE
BATTERY_PROPERTY_CURRENT_NOW
BATTERY_PROPERTY_ENERGY_COUNTER
BATTERY_STATUS_CHARGING
BATTERY_STATUS_DISCHARGING
BATTERY_STATUS_FULL
BATTERY_STATUS_NOT_CHARGING
BATTERY_STATUS_UNKNOWN

Como alguns fabricantes não fornecem recursos para obter, por exemplo, o valor da corrente elétrica que o dispositivo está consumindo, cada uma dessas variáveis possui um valor padrão, que é o valor retornado caso essa informação não esteja disponível.

Embora os dois métodos forneçam informações que permitem monitorar o status da bateria, é bastante comum encontrar aplicativos que buscam esses dados diretamente nas variáveis do *driver* do *Fuel Gauge*, ao invés de utilizar a classe *BatteryManager*. Isso ocorre, pois a leitura desses dados através da classe emprega um *Broadcast Receiver*, e por isso, pode ser mais lenta do que a leitura direta da variável.

2.3.3 Ciclos de Aplicação

Os ciclos de vida da aplicação definem o comportamento da aplicação, ou seja, o que vai acontecer em cada estado em que a aplicação pode se encontrar. No Android, os componentes de uma aplicação são: Atividade (*Activity*), Serviço (*Services)*, Provedor de Conteúdo (*Content Provider*) e *Broadcast Receivers* (ANDROID, 2014e).

Uma *Activity* (ANDROID, 2014a) contém a interface para interação com o usuário, e novas atividades podem ser criadas de acordo com a ação tomada por ele. Já um *Service* (ANDROID, 2014m) executa em segundo plano, normalmente desempenhando tarefas de suporte a uma *Activity* que inicializa o dado serviço, mas sem interação direta com o usuário.

Através dos *Content Providers* (ANDROID, 2014h) é possível gerenciar o compartilhamento de dados entre os aplicativos. Já o *Broadcast Receiver* (ANDROID, 2014g) é um componente utilizado para receber mensagens oriundas do sistema ou de outras aplicações, por exemplo, um anúncio que a tela foi desligada ou que a bateria está com pouca carga. *Broadcast Receivers* não possuem uma interface com o usuário, mas podem adicionar notificações na barra de status.

Na abordagem deste trabalho, um serviço é utilizado para aquisição de informações da bateria. Desta forma, apenas *Activities* e *Services* são revisados, visto que é a partir da *Activity* do aplicativo que o *Service* é inicializado.

2.3.3.1 Activity

A *Activity* (ANDROID, 2014b) é o componente da aplicação que fornece uma interface através da qual o usuário interage com o sistema para realizar algo, como uma chamada telefônica, enviar um *e-mail* ou visualizar um mapa.

Uma *Activity* pode iniciar outra *Activity* para realizar diferentes ações. No projeto do aplicativo, uma dada *Activity* é definida como principal, a qual define a primeira tela apresentada ao usuário quando este inicia o aplicativo. A partir dela, novas *Activities* são criadas, e colocadas em uma pilha. A *Activity* do topo da pilha é a que se encontra ativa e visível, enquanto as demais estão paradas ou pausadas. A qualquer momento o usuário pode retornar a uma destas *Activities* pausadas, e neste caso, a *Activity* do topo é substituída.

Essencialmente, uma *Activity* pode assumir três estados distintos, conhecidos como: *Resumed*, *Paused* e *Stopped*. No estado *Resumed*, a atividade está ativa, ou

seja, em primeiro plano e tem o foco do usuário. No estado *Paused*, uma atividade está pausada quando existe outra atividade em primeiro plano, com foco do usuário, mas a atividade pausada ainda se encontra visível. Isso ocorre quando uma *Activity* é "transparente" ou não ocupa todo espaço da tela. Já uma atividade encontra-se no estado dito *Stopped*, quando estiver completamente obscurecida, por outra que está em primeiro plano, com foco do usuário e ocupando toda tela do dispositivo.

A figura 5 ilustra o ciclo de vida de uma *Activity*, onde seus estados podem ser observados, bem como as transições entre estes estados. Para realizar as mudanças de estado e determinar a ação da atividade ao entrar em cada um deles, existem métodos específicos que devem ser implementados, tais como *onCreate()* e *onPause()*.

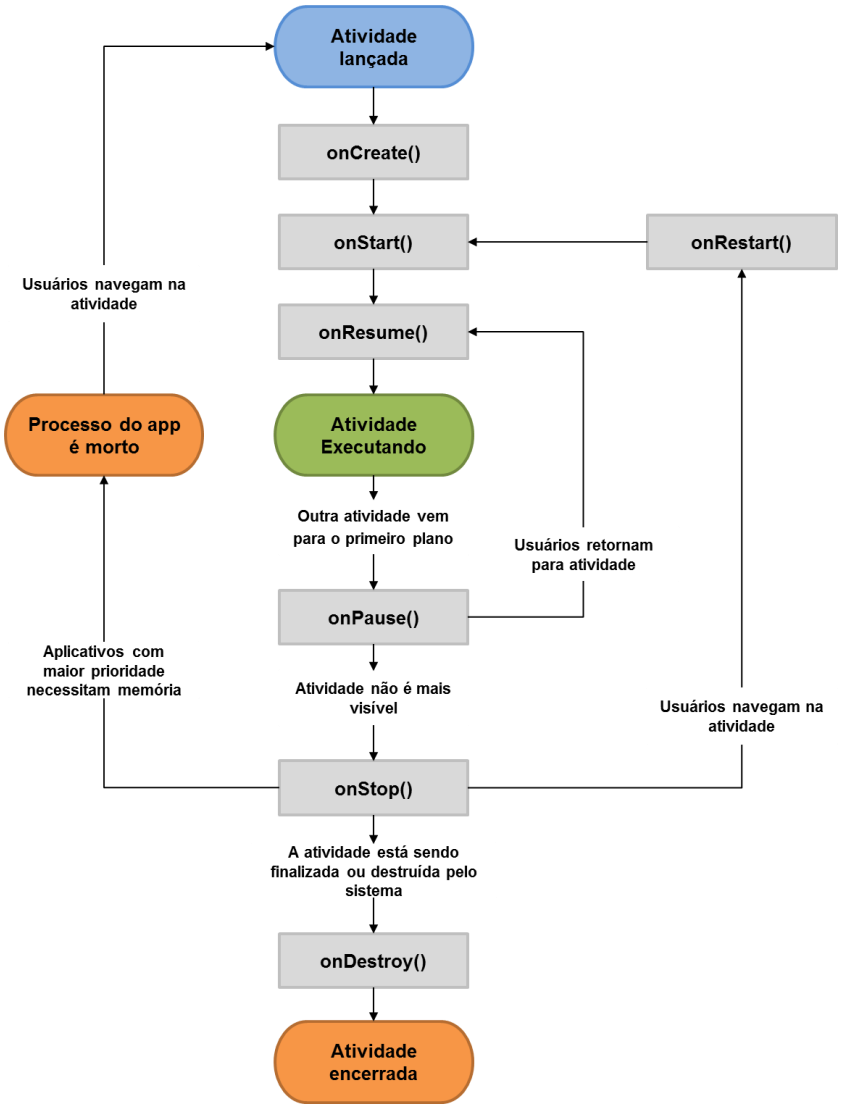

Figura 5: Ciclo de vida de uma *Activity*

Fonte: Adaptado de (ANDROID, 2014b)

2.3.3.2 Service

Um *Service* (serviço) (ANDROID, 2014m) é um componente de aplicação do Android capaz de realizar operações de longa duração em segundo plano sem prover uma interface de usuário. Uma aplicação pode inicializar um serviço, e este pode continuar executando mesmo que o usuário inicie outras aplicações em primeiro plano. Os serviços são utilizados, por exemplo, para executar uma música, fazer *download* de arquivos ou atualizações e realizar tarefas de I/O.

Os serviços podem ser de dois tipos, dependendo de como estes são inicializados. Um serviço é considerado do tipo *started* quando este é inicializado através de uma chamada ao método *startService()*. Uma vez iniciado, ele pode executar em segundo plano indefinidamente, mesmo que o componente que o iniciou seja destruído. Normalmente, esse tipo de serviço executa uma tarefa pontual e não fornece nenhum tipo de retorno. A finalização de um serviço deste tipo pode ser feita através do método *stopSelf()*, ou seja, um auto encerramento após a conclusão de alguma tarefa, ou o componente que o iniciou pode finalizá-lo através do método *stopService()*. Já um serviço em *bound* é inicializado através do método *bindService()*. Esse tipo de serviço fornece uma interface cliente-servidor para comunicação entre a aplicação e o serviço, e permanece ativo somente enquanto existir alguma aplicação conectada a ele. Quando não houverem mais conexões ativas, o serviço é finalizado.

O ciclo de vida de um serviço pode seguir dois caminhos, dependendo do tipo de serviço utilizado, conforme ilustrado na figura 6. Contudo, esses caminhos não estão completamente distintos, pois um serviço do tipo *started* pode ser transformado em um serviço do tipo *bound*, caso alguma aplicação se conecte a este. Por exemplo, um serviço pode ser iniciado, através do método *startService()*, para executar uma música, e depois, uma *A*ctivity pode se conectar a esse mesmo serviço para realizar alguma tarefa de controle ou ainda para obter dados sobre o som que está sendo executado, chamando para isso o método *bindService()*. Nesse caso, mesmo que os métodos *stopService()* e *stopSelf()* sejam chamados, o serviço continua em execução até que todos clientes se desconectem.

Um serviço normalmente compartilha o mesmo processo do aplicativo que o iniciou. Contudo, é possível fazer com que um serviço execute em um processo separado. Executar o serviço em seu próprio processo previne que a aplicação trave, case o serviço realize operações de longa duração (VOGEL, 2013).

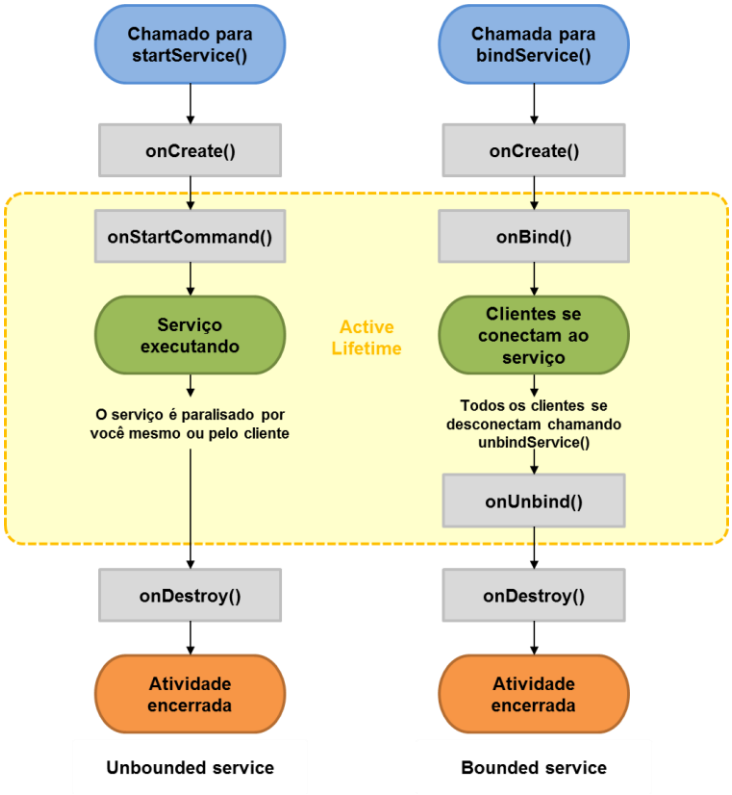

Figura 6: Variantes do Ciclo de vida de um serviço

Fonte: Adaptado de (ANDROID, 2014m)

3 TRABALHOS RELACIONADOS

Nosso grupo de pesquisa tem estudado o consumo de energia de dispositivos Android. Em Tonini et al. (2013), boas práticas de programação para Android são comparadas quanto a desempenho e consumo energético. Já em Vieira et al. (2012), a comparação é entre diferentes paradigmas de programação, utilizando para isso algoritmos de ordenação de dados. Nesses estudos, a ferramenta PowerTutor é empregada nas avaliações de consumo energético de aplicativos. No entanto, existem outras ferramentas disponiveis, baseadas em diferentes técnicas e que variam quanto a flexibilidade ou precisão.

Este capítulo apresenta a revisão do estado-da-arte em ferramentas e abordagens para avaliação do consumo energético para plataformas móveis no nível de *software*. Diversas ferramentas para avaliação de consumo energético têm sido propostas, tanto para dispositivos móveis, quanto focadas em outros domínios de aplicação. Primeiramente, são apresentados os trabalhos mais atuais e mais relevantes para este estudo, com foco em plataformas móveis. Posteriormente, esta revisão é estendida de modo a discutir também abordagens mais convencionais ou propostas para outros domínios de aplicação.

3.1 Avaliação de consumo em dispositivos móveis

3.1.1 Aneprof

A Aneprof (CHUNG, LIN e KING, 2011) é uma ferramenta para *profiling* energético de dispositivos Android (GOOGLE, 2014), a qual tem por objetivo a geração de perfis energéticos com granularidade em nível de função baseado em amostras de medição real. Para adquirir informações de status da execução do aplicativo analisado, o sistema operacional foi instrumentado por *profilers* atuando em diferentes camadas.

Para coletar amostras de potência, um dispositivo de coleta de dados foi conectado a um computador denominado monitor. É no monitor que os dados coletados pelo sistema são cruzados com as amostras de potência, gerando o perfil

energético de um aplicativo. No monitor também é executada uma interface de usuário que permite a configuração de parâmetros, como a frequência de coleta das amostras. Além disso, após análise dos *bytecodes* do aplicativo alvo, podem ser selecionados os métodos e funções a serem monitorados, o que confere boa granularidade a ferramenta.

O funcionamento da ferramenta ocorre de acordo com o controle de fluxo ilustrado pela figura 7. A configuração realizada pelo usuário através da GUI alimenta tanto o *Runtime Controller*, quanto o *NI Driver* (responsável pela coleta das amostras). A partir daí, os *profilers* coletam dados que alimentam um banco (*External Storage*). A ferramenta de análise *offline* (ilustrada como *Offline Analyzer*) estima o consumo com base nos dados providos pelo banco de dados e pelos *logs* de potência coletados pelo *NI Driver*.

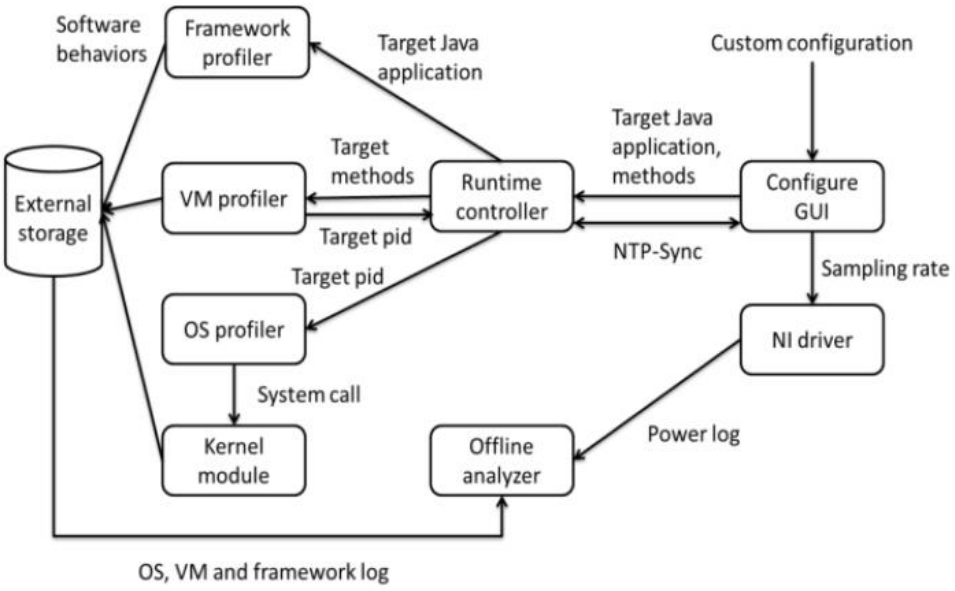

Figura 7: Controle de fluxo do Aneprof
Fonte: CHUNG, LIN e KING (2011)

Segundo os autores, entre os principais desafios enfrentados no desenvolvimento da ferramenta Aneprof estão relacionados à sincronização entre as amostras de dados, bem como o monitoramento de aplicações Java (ORACLE, 2012) no sistema Android, uma vez que o *software* é executado em diferentes camadas. A precisão do sistema, de acordo com os autores, é diretamente relacionada ao atraso entre a troca de mensagens entre o dispositivo alvo e o monitor, assim como a sincronização entre eles. Conforme apresentado em Chung, Lin e King (2011), o erro máximo é de 28% e o *overhead* da ferramenta é de 5%, um nível de precisão aceitável.

3.1.2 JouleUnit

O JouleUnit (WILKE, GOTZ e RICHLY, 2013) é um *framework* para *profiling* de energia. Trata-se de um *framework* genérico, o qual pode, segundo os autores, ser reutilizado em diferentes tipos de aplicações. Para isso, o *framework* é dividido em quatro componentes, os quais são: *WorkLoad Runner*, *Energy Profiler*, *Coordinator Layer* e *WorkBench*.

O *WorkLoad Runner*, ou Executor da Carga de Trabalho, é o componente responsável por fornecer e executar os dados de teste e a carga de trabalho no dispositivo a ser testado. Já o *Energy Profiler* fornece recursos para amostragem de consumo energético do dispositivo avaliado. Por sua vez, o *Coordinator Layer*, ou *a* Camada de Coordenação, é responsável por disparar a carga de trabalho e o *profiling*, além de sincronizar os resultados extraídos de cada um desses componentes. Primeiro as cargas de trabalho são carregadas no dispositivo e posteriormente, o *profiler* é iniciado, e então é iniciada a execução da carga de trabalho. O componente *Workbench* permite integrar o *framework* ao Eclipse IDE, suportando editores, assistentes e *views* para definir mais facilmente cargas de trabalho, disparar *profiling* e apresentar resultados. A figura 8 mostra como é essa interface e como os resultados são apresentados.

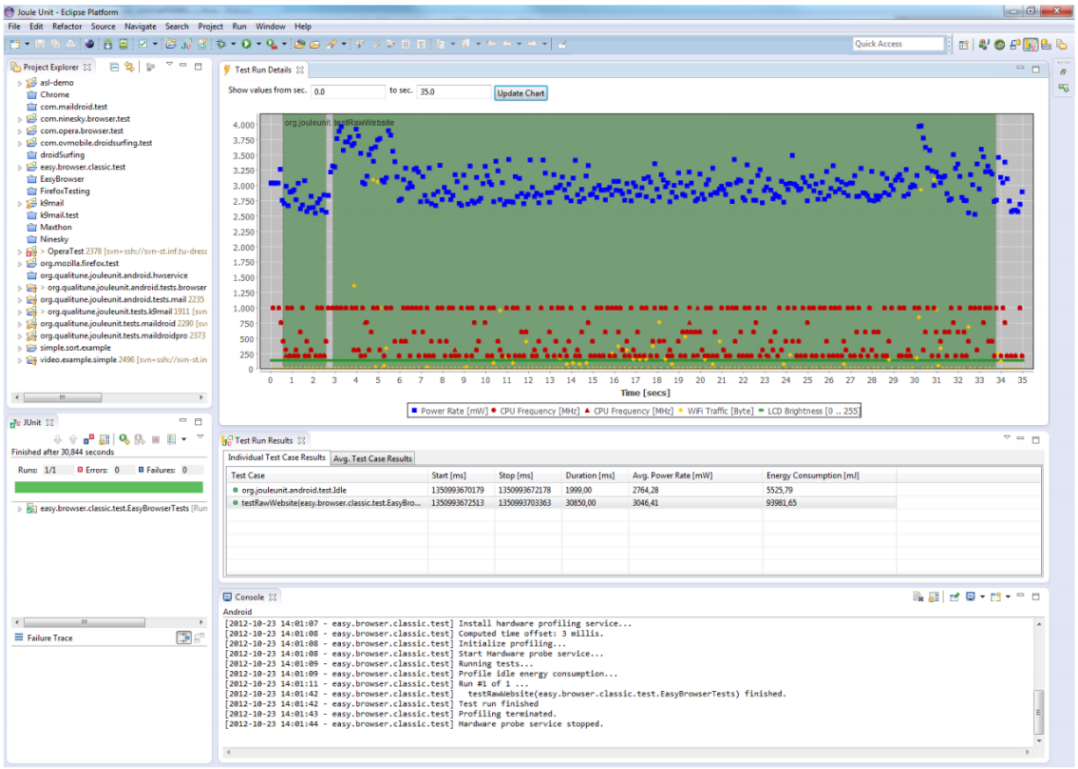

Figura 8: Janela do Workbench do JouleUnit integrado a IDE Eclipse

Fonte: WILKE, GOTZ e RICHLY (2013)

O núcleo do *Energy Profiler* é composto por três classes, conforme a figura 9. Estas classes são a base do *profiler* e podem ser estendidas para incorporar características mais de determinado dispositivo. A classe *JouleProfilers* é responsável pelo perfil do dispositivo, para isso, cria *EnergyProfilers* que são conjuntos de *PowerRate*. Cada instância da classe *PowerRate* representa a potência do dispositivo em um determinado ponto, ou seja uma amostra do consumo.

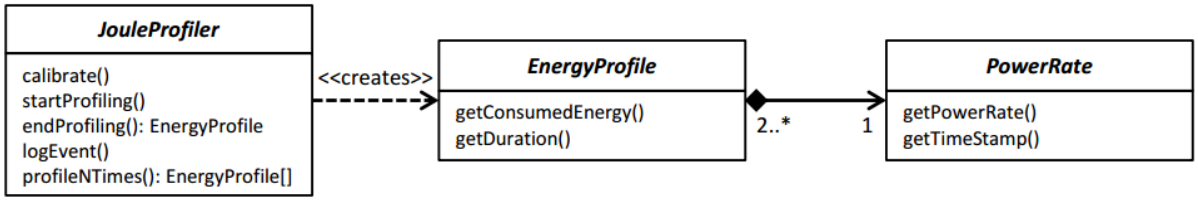

Figura 9: Classes do componente Energy Profiler

Fonte: WILKE, GOTZ e RICHLY (2013)

3.1.3 PowerBooter

PowerBooter é uma técnica para construção automática de modelos de potência a partir de sensores de tensão de baterias de dispositivos móveis modernos,

apresentada por Zhang et al. (2010). O método tem por objetivo obter a curva de descarga da bateria para cada componente individualmente. Para isso, todos componentes do sistema são colocados em modo de baixo consumo, exceto aquele a ser analisado. O processo se repete, colocando cada componente em um estado de potência num período de 15 minutos. Depois de determinar a curva de descarga da bateria para cada componente, o consumo médio de potência é calculado e é utilizado para alimentar o modelo de potência, conforme ilustrado na figura 10. Uma ferramenta para avaliação de aplicativos Android, chamada PowerTutor (ZHANG *et al.*, 2011), também foi proposta pelos mesmos autores, e está disponível para *download* na Google Play (GOOGLE, 2014). Esta ferramenta implementa a técnica PowerBooter e analisa o consumo de componentes de *hardware* como CPU, LCD, GPS, Wi-Fi, áudio, entre outros componentes. A interface da ferramenta é apresentada na figura 11, onde a imagem à esquerda representa a tela de abertura e a imagem à direita apresenta o resultado de uma análise de consumo considerando os componentes CPU, Wi-Fi, LCD e GPS.

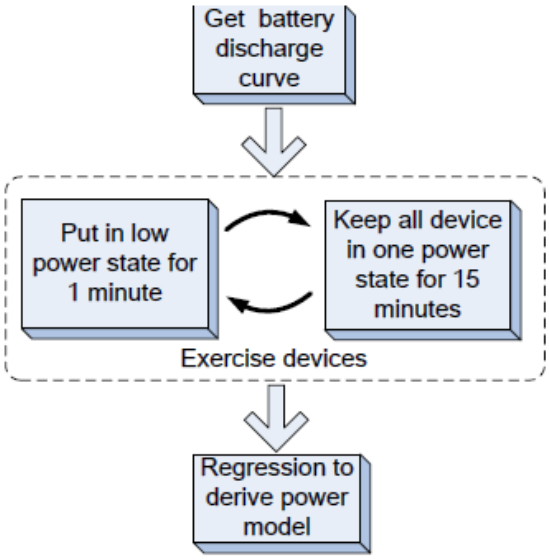

Figura 10: Construção do modelo de potência no PowerBooter
Fonte: ZHANG *et al.* (2010)

Figura 11: Interface da ferramenta PowerTutor
Fonte: ZHANG *et al.* (2011)

3.1.4 DevScope

A DevScope (JUNG *et al.*, 2012) é uma ferramenta desenvolvida em forma de *framework*, a qual utiliza os sensores do dispositivo para verificar o status da bateria e modelar o consumo de energia a nível de componentes de *hardware*. Ao componente que monitora a bateria, é dado o nome de BMU (do inglês, *Battery Monitoring Unit*). Na arquitetura apresentada na figura 12, a BMU encontra-se entre a bateria e os componentes de *hardware*, fornecendo informações a um módulo do DevScope chamado *BMU Event Monitor*. Esse módulo se comunica com um Controlador de Tempo (*Timing Controller*), e um Controlador de Componentes (*Component Controller*) para fornecer as informações necessárias para o Gerador de Modelos de Potência (*Power Model Generator*). Assim, como saída, o DevScope fornece um modelo de potência para cada componente de *hardware*.

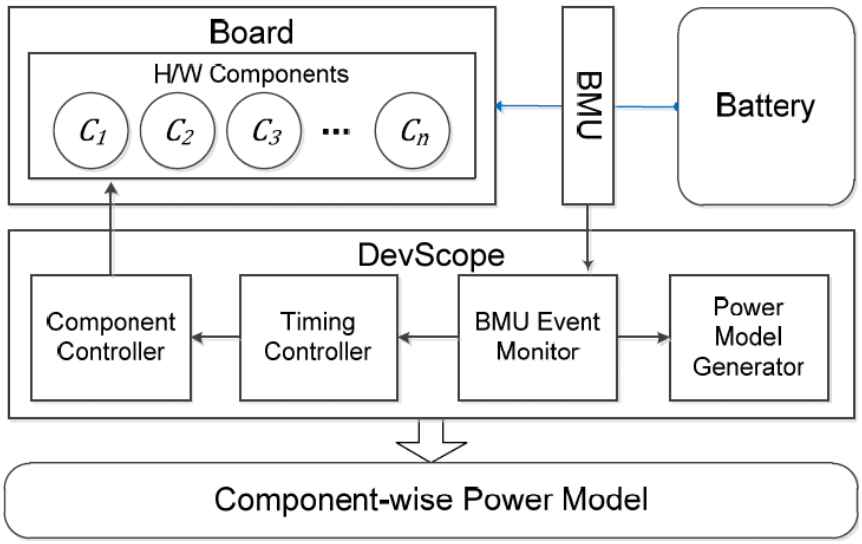

Figura 12: Visão geral da ferramenta DevScope

Fonte: JUNG *et al. (*2012)

Como a BMU não oferece o consumo energético de cada componente individualmente, esse valor precisa ser calculado. A equação 6 demonstra como o cálculo é efetuado. Nesta equação, P_{Medida} é a potência total consumida pelo sistema, e é medida através da BMU. Para P_{CPU} é feita uma estimativa através do modelo de potência da CPU computado pelo DevScope. Esse modelo é gerado quando todos componentes estão desligados, para diminuir a quantidade de erros.

$$\boldsymbol{P_{componente} = P_{Medida} - P_{CPU}}$$

Equação 4: Cálculo da potência dos componentes no DevScope

Os mesmos autores apresentam ainda, a ferramenta AppScope (YOON *et al.*, 2012), uma aplicação Java desenvolvida para *desktops*, a qual se comunica com o dispositivo alvo através de uma porta USB, e permite a avaliação de consumo energético em tempo real para dispositivos Android. A ferramenta AppScope instala o aplicativo DevScope no dispositivo alvo e adquire as informações de consumo energético através dele, apresentando os resultados para o usuário no formato de gráficos e tabelas na interface gráfica do aplicativo.

3.1.5 SeSaMe

Assim como o PowerBooter, a abordagem proposta por Dong e Zhong (2011) apresenta uma ferramenta chamada SeSaMe capaz de gerar o modelo de consumo

energético sem a necessidade de ferramentas auxiliares, utilizando para isso, os sensores presentes em plataformas móveis mais modernas.

Experimentos feitos pelos autores mostram similaridade entre a corrente medida através da Interface da Bateria (Bat I/F) e um equipamento para medição de potência chamado DAQ (*Data Acquisition*). A arquitetura do SeSaMe, ilustrada na figura 13, é composta por três componentes principais denominados *Data Collector, Model Constructor* e *Model Manager*. O *Data Collector*, ou Coletor, interage com o SO para adquirir informações de status do sistema como uso de CPU, ou de memória, assim como efetuar a comunicação com a interface de bateria. O *Model Constructor* ou Construtor é a unidade principal do SeSaMe, a qual recebe os dados do Coletor e constrói o modelo através dos módulos *Model molding* e *do Predictor transformation.* Por fim, o *Model* Manager é o responsável por organizar os diferentes modelos gerados pelo Construtor em uma tabela *Hash* de acordo com a configuração do sistema. Este componente é o responsável também por checar a precisão do sistema. Caso seja necessária alguma adaptação, o *Model Constructor* é acionado novamente.

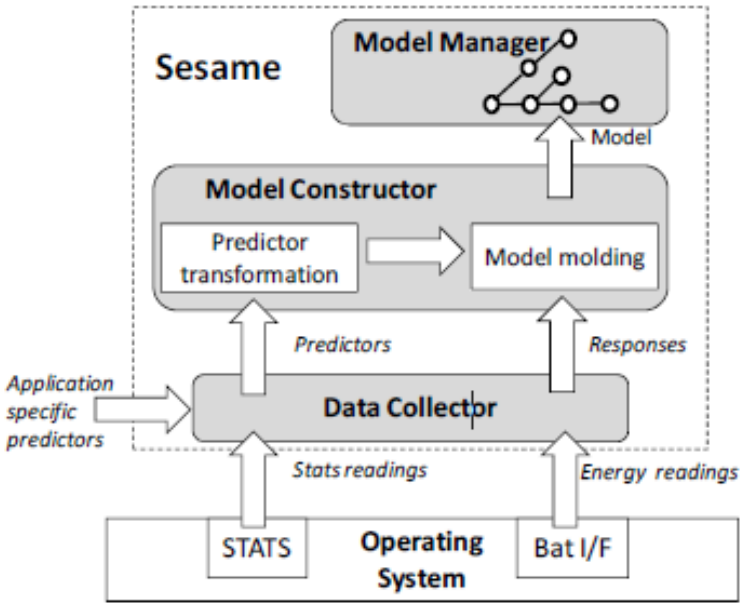

Figura 13: Arquitetura do SeSaMe
Fonte: DONG e ZHONG (2011)

As características principais do SeSaMe são: geração automática de modelos, baixo *overhead* e complexidade, alta precisão, e adaptação a mudanças de contextos. Para garantir baixo *overhead*, tarefas mais pesadas são executadas somente quando

o sistema está em repouso e conectado a uma fonte AC (*Alternate Current*). Enquanto o *software* é utilizado, são executadas somente a coleta de dados e cálculos simples. A precisão da ferramenta nos testes efetuados pelos autores utilizando as plataformas Lenovo Thinpad T61 (Linux), Dell Latitude D630 (Linux), Nokia N85/N96 (Symbian) e Google G1/Nexus One (Android), varia de 83 a 95%, comprovada através de medição direta.

3.1.6 Eprof

O trabalho apresentado por Pathak, Hu e Zhang (2012) propõe uma ferramenta capaz de capturar o comportamento dos componentes de um *smartphone* dirigido por chamadas de sistema, modelando o consumo de potência através de FSM. Os autores defendem que o consumo energético nos *smartphones* tem um comportamento assíncrono, ou seja, apresenta alta complexidade de modelagem, isso por que muitos componentes denominados “exóticos” estão presentes nestes dispositivos e influenciam diretamente no consumo energético. São exemplos desses componentes, o GPS, câmera de vídeo, acelerômetros, entre outros.

A arquitetura da ferramenta Eprof é composta por três blocos principais, que são: (1) *code instrumentation and logging*, (2) *energy accounting*, e (3) *data presentation*. Primeiramente, o sistema alvo é instrumentado para que efetue o rastreamento de chamadas de sistema e rotinas. A seguir, o *trace* gerado na primeira etapa é utilizado para dirigir a FSM e modelar o consumo energético. Por fim, o Eprof apresenta o perfil de consumo de energia do sistema. A arquitetura bem como o seu fluxo de trabalho está ilustrada na figura 14.

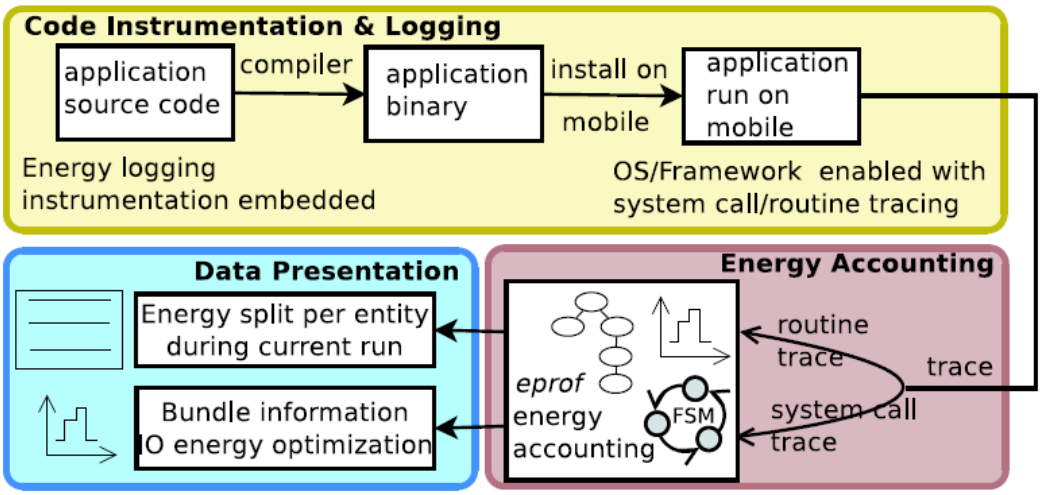

Figura 14: Arquitetura Eprof

Fonte: PATHAK, HU e ZHANG (2012)

3.1.7 NEAT

NEAT (*Novel Energy Analysis Toolkit*) é uma ferramenta para medição de consumo energético proposta por Browers, Zuniga e Langendoen (2014), e tem como base um pequeno hardware para medição de tensão e corrente do dispositivo, o qual é integrado a bateria do dispositivo avaliado. Além do hardware para coleta de amostras de tensão e corrente (Power Meter), a ferramenta utiliza um aplicativo que armazena os eventos ocorridos no sistema (Event Logger). A análise ocorre de forma off-line, e os resultados podem ser visualizados na ferramenta Traceview (ANDROID, 2014m) que é um visualizador gráfico para logs de execução de aplicativos Android. A figura 15 ilustra a abordagem e seus componentes principais.

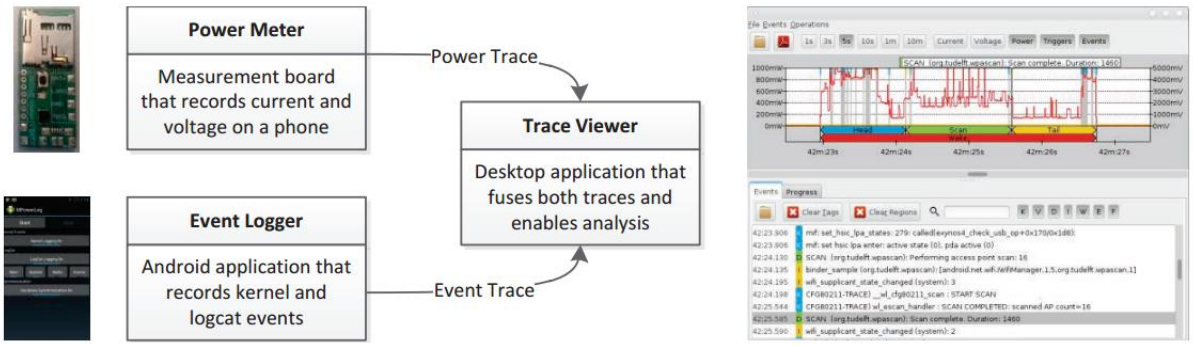

Figura 15: Componentes da ferramenta NEAT

Fonte: BROWERS, ZUNIGA e LANGENDOEN (2014)

Esta abordagem visa aliar a mobilidade de ferramentas baseadas em modelos, com a precisão de ferramentas de medição. A granularidade obtida é a nível de função. O hardware desenvolvido para medição possui recursos que permitem disparar a medição simultaneamente a coleta de eventos, facilitando a sincronia desses eventos. Para isso, a placa de medição deve ser conectada a algum componente do dispositivo que permite esse disparo.

3.1.8 AndroProf

Na proposta de Sartor, Beck e Corrêa (2014), o emulador QEMU foi modificado para permitir a geração de *traces* de aplicações Android. Esta extensão do QEMU foi nomeada AndroProf e suporta arquiteturas MIPS, ARM e parcialmente a arquitetura x86, e é capaz de gerar tanto o *trace* de aplicações Java quanto de aplicações nativas.

A figura 16 detalha a extensão provida pela AndroProf, identificando módulos adicionados ao QEMU, módulos modificados e que não sofreram alteração. A versão do QEMU fornecida com o SDK Android para emular disposivos foi modificada de forma a permitir a extração de um *trace* detalhado, bem como o mecanismo de conexão VNC foi alterado para permitir a gravação desse *trace*. Além disso, foram adicionadas interfaces gráficas para análise de dados e para caracterização das instruções utilizadas na arquitetura (através dessa interface pode ser definido o coeficiente de potência pra cada instrução, por exemplo). Todos os dados coletados pelo QEMU são armazenados em arquivos que podem ser analisados através dessas interfaces gráficas e então armazenados em uma base de dados.

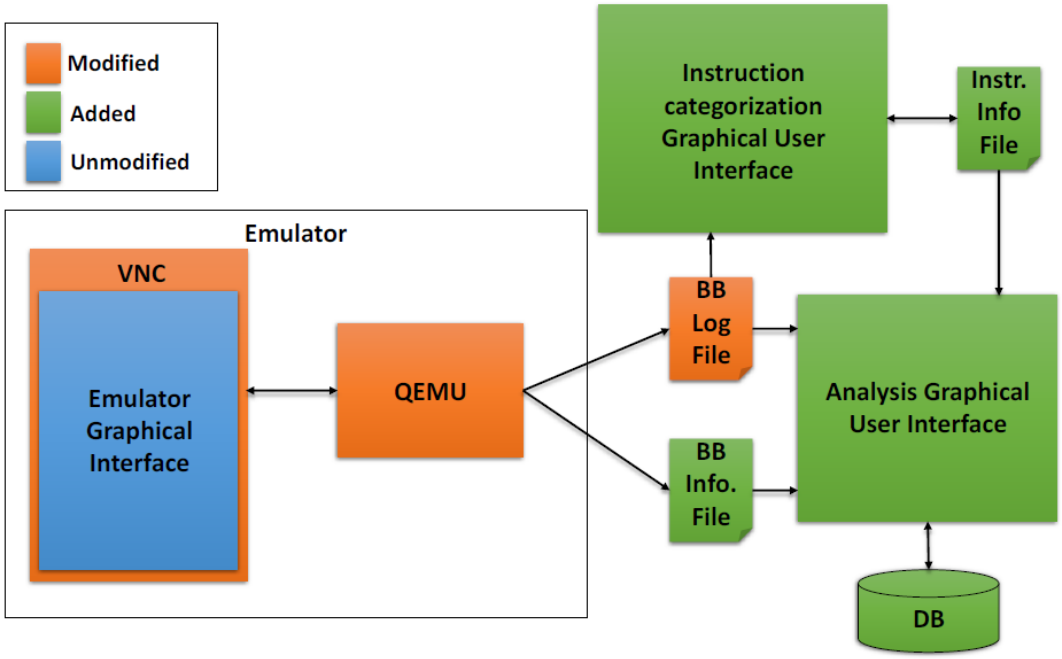

Figura 16: Arquitetura do Androprof
Fonte: SARTOR, BECK e CORRÊA (2013)

3.2 Avaliação de consumo: abordagens genéricas

São muitos os trabalhos que endereçam este problema de estimativa e análise de perfis energéticos, visto que é uma preocupação constante para engenheiros envolvidos no projeto de sistemas embarcados eficientes. Algumas abordagens consideradas estado-da-arte e que focam nas plataformas móveis foram discutidas nas seções anteriores. Esta seção complementa este estudo, revisando abordagens mais genéricas de análise de consumo energético, como Wattch (BROOKS, TIWARI e MARTONOSI, 2000) e PowerScope (FLINN e SATYANARAYANAN, 1999).

Wattch (BROOKS, TIWARI e MARTONOSI, 2000) é um *framework* para análise e otimização de consumo energético a nível de arquitetura. Neste trabalho, as principais unidades do sistema são mapeadas, e o modelo energético é calculado a partir da capacitância, que influencia diretamente o consumo de potência dinâmica em circuitos CMOS. A ferramenta é agregada ao simulador SimpleScalar (SIMPLESCALAR, 2011), o qual analisa quais unidades são acessadas por ciclo, permitindo o cálculo do consumo total de potência de uma aplicação. Segundo os autores, o erro máximo apresentado pela abordagem é de 10%.

Muitas das ferramentas propostas na literatura baseiam-se em medições diretas. Dentre estas, destaca-se o PowerScope (FLINN e SATYANARAYANAN, 1999), que é uma ferramenta de medição direta do consumo energético. O consumo

de energia é obtido através da coleta de amostras de potência com *hardware* externo, e mapeado com eventos do sistema a nível de estrutura de processos, através da técnica SBM. Este trabalho serviu como base para outras propostas mais recentes tal como o Aneprof (CHUNG, LIN e KING, 2011), que estende esta abordagem propondo melhorias quanto a sincronização entre as amostras, por exemplo.

Xian, Cai e Lu (2007) também propõe uma ferramenta de medição direta, com foco no *software* executado em computadores com múltiplos componentes de I/O. O trabalho se preocupa ainda com a questão de correlação de potência, propondo uma heurística, chamada M-Sync para tratar o problema do atraso entre as máquinas monitor e alvo.

Na ferramenta proposta por Snowdon, Petters e Heiser (2005), baseada também em medição direta de potência, a análise pode ser realizada tanto *online* quanto *offline*. Na análise *offline*, esta abordagem é bastante similar ao PowerScope (FLINN e SATYANARAYANAN, 1999), enquanto a análise *online* baseia-se na medição direta e na acumulação das amostras de potência coletadas, as quais são simultaneamente associadas a uma lista de controle dos processos executados.

3.3 Considerações Gerais

Esta seção discute e compara as abordagens apresentadas com base nos critérios discutidos no Capítulo 2. As abordagens foram classificadas quanto a técnica utilizada (baseada em modelo ou medição), a granularidade dos resultados, bem como a possibilidade de obter o consumo de energia dos componentes do sistema, como CPU, Display, Wi-Fi, etc. Por fim, as abordagens são classificadas quanto ao método de aquisição de dados empregado na estimativa de potência, quando aplicável. Quando informações da bateria do próprio dispositivo são empregadas, classifica-se como aquisição interna, já a externa, corresponde ao uso de instrumentação externa para medição da potência. O resumo dessa comparação pode ser vista na tabela 3.

A ferramenta Aneprof se destaca pela elevada granularidade, e por ser uma das poucas, senão a única ferramenta de medição direta para dispositivos Android. JouleUnit apresenta um conceito de medição interna, porém não apresenta boa granularidade. Outros esforços como PowerTutor, AppScope, SeSaMe, AndroProf e

Eprof utilizam modelos para estimativa de consumo de energia, com destaque para Eprof e AneProf, capazes de estimar o consumo no nível de função.

Tabela 3: Comparativo entre as abordagens

Abordagem	**Técnica Utilizada**	**Granularidade**	**Por Componente de *Hardware***	**Aquisição de dados**
PowerTutor	Modelo	Aplicação	Sim	Não aplicável
JouleUnit	Medição	Aplicação	Sim	Interna
Aneprof	Medição	Função	Não	Externa
AppScope	Modelo	Aplicação	Sim	Interna
SeSaMe	Modelo	Sistema	Não	Interna
Eprof	Modelo		Sim	Não aplicável
NEAT	Medição	Função	Sim	Externa
AndroProf	Modelo	Função	Não	Não aplicável

Com exceção de Aneprof, AndroProf e SeSaMe, as demais abordagens são capazes de estimar o consumo dos componentes de *hardware* do sistema individualmente. Quanto ao método utilizado para aquisição de dados, Aneprof realiza medição com uso de instrumentação externa. AppScope e SeSaMe utilizam dados extraídos da bateria do dispositivo para alimentar o modelo utilizado para geração das estimativas. No caso de PowerTutor e Eprof, são utilizados apenas os coeficientes de potência de cada componente, os quais estão embutidos no próprio modelo. Já o AndroProf utiliza coeficientes de potência associados a instruções e um trace detalhado da execução para realizar a estimativa.

DevScope e PowerBooter não estão listados na tabela, pois tratam de técnicas para auto geração de modelos dos dispositivos e não geração de estimativas. Trabalhos como Watch e PowerScope também não foram analisados na tabela 3, visto que estes são abordagens genéricas e portanto fora do escopo deste trabalho. Estes trabalhos reforçam que a avaliação de consumo energético é um problema recorrente também em outros domínios de aplicação.

4 ABORDAGEM PROPOSTA

Este trabalho propõe uma abordagem para avaliação de consumo energético de aplicativos Android, que emprega uma técnica de medição baseada em amostras obtidas internamente, sem necessidade de instrumentação externa. Para avaliar esta abordagem foi desenvolvida também uma ferramenta para estimativa de consumo energético em dispositivos móveis que utilizam a plataforma Android, a qual é capaz de estimar o consumo energético de aplicativos ou blocos de código Java-Android.

Neste capítulo, a abordagem é detalhada na Seção 4.1 e sua implementação é detalhada e discutida na Seção 4.2. Por fim, o capítulo apresenta ainda uma discussão sobre as principais características da ferramenta bem como comparações com as principais abordagens apresentadas no Capítulo 3.

4.1 Visão Geral

A abordagem proposta neste trabalho foi denominada SEMA (*Self Energy Metering for Android*) e baseia-se na medição interna do consumo para dispositivos Android. A técnica empregada nesta abordagem baseia-se na coleta amostras de tensão e corrente do dispositivo.

No entanto, diferentemente da maioria das abordagens similares encontradas, estas amostras são obtidas a partir da leitura direta dos dados fornecidos pelo componente *Fuel Gauge,* provido pela plataforma Android*,* e que podem ser obtidas através do *driver* da bateria. Desta forma, a abordagem emprega uma técnica de medição interna, na qual recursos do próprio dispositivo são utilizados, sem necessidade de instrumentação externa. A limitação dessa técnica é a baixa taxa de atualização dessa interface, que pode chegar até 10s, porém pode variar de dispositivo para dispositivo.

Além disso, essa técnica permite que a abordagem possa ser utilizada em diferentes dispositivos Android sem a necessidade de alterações, contando que o Fuel Gauge do dispositivo forneça informações da tensão que o alimenta e da corrente que circula por ele.

A escolha dessa técnica se dá por sua implementação ser possível sem a necessidade de realizar alterações no SO ou requerer permissões especiais do mesmo. Ainda que essas alterações sejam possíveis, tendo em vista que o Android é um sistema *open source*, elas requerem um conhecimento avançado. Além disso, as

ferramentas e recursos utilizados para esse tipo de alteração não são licenciados pelos fabricantes de dispositivos comerciais, assim, o uso dessas técnicas fica restrito ao uso em plataformas focadas em desenvolvimento ou em simuladores.

Visando prover uma avaliação com baixa sobrecarga, nossa abordagem emprega serviços do Android como meio para realizar as coletas de amostras em segundo plano, enquanto a atividade (ou *Activity*) em análise é exibida na tela e ao fim da execução, os dados coletados são analisados gerando um *log* que contém essa análise. Ainda assim, a sobrecarga causada pelo serviço depende da taxa de amostragem empregada, a qual pode ser definida na criação do serviço através de seu construtor.

Na SEMA, diferente de abordagens como Aneprof, onde a precisão é afetada pelo *overhead,* a precisão é limitada pela baixa taxa de atualização da interface da bateria, visto que forçar taxas de atualização muito maiores que a interface não resultam em melhor precisão (DONG e ZHONG, 2011), pois as leituras se repetem. Assim, a abordagem é mais indicada para avaliar aplicativos cujo tempo de execução excede a taxa de atualização da interface da bateria do dispositivo onde serão executados.

O foco da ferramenta são desenvolvedores de aplicativos, os quais necessitam ferramentas para avaliar a eficiência energética de seus códigos, permitindo a identificação de gargalos, bem como o impacto das otimizações realizadas. Para isso, é importante que a ferramenta seja portável a diferentes aparelhos, bem como não exija alterações no SO, características estas encontradas na SEMA.

A figura 17 ilustra a abordagem SEMA, a qual tem o Gerente de Execução como componente principal. Este componente é composto pelo Monitor de tempo de execução e pelo Coletor de Amostras. O Gerente de execução controla o monitor de tempo de execução, e é responsável por disparar e parar o coletor de amostras. O Coletor de Amostras através da Interface da bateria coleta amostras de tensão e de corrente. O SEMA leva em conta o aplicativo que está sendo avaliado, bem como faz uso de arquivos do SO que fornecem informações da bateria e status do processo.

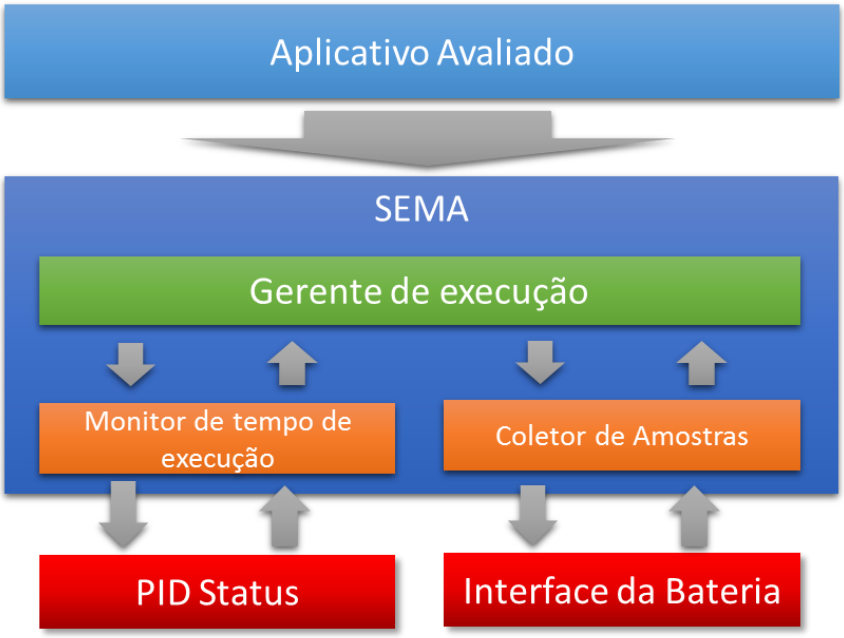

Figura 17: Visão geral da abordagem SEMA

4.2 Implementação

Nesta seção é apresentada uma visão geral da implementação da ferramenta de estimativa de consumo energético, bem como detalhamento sobre seu fluxograma de execução e diagrama de classes. A parte central da abordagem é um serviço (*service*) que coleta amostras de tensão e corrente do sistema, e as acumula enquanto o serviço estiver ativo. O serviço é disparado pelo aplicativo avaliado, imediatamente antes do bloco de código para o qual se deseja estimar o consumo de energia. Ao término da execução desse bloco, uma nova mensagem é enviada ao gerente de execução, sinalizando que a execução do serviço deve ser parada. A figura 18 contém um exemplo de uso dessa ferramenta.

```
Context context = getBaseContext();
EnergyMonitor energy = new EnergyMonitor(context);

energy.start();

bubbleSort(array);

energy.stop();
```

Figura 18: Exemplo de uso de ferramenta

O tempo de execução, calculado pelo Monitor de Tempo de Execução, é obtido através do arquivo que contém status do processo avaliado, localizado em */proc/PID/stat*, campos 14 (*utime*) e 15 (*stime*), ou seja tempo no modo usuário e no modo *kernel*. Esse arquivo contém o tempo de execução do aplicativo em *clock ticks*. Esse valor, dividido pela frequência de operação do processador, obtido em */sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq*, resulta no tempo em segundos, conforme equação 8. O tempo é lido assim que o serviço é iniciado, e uma nova leitura é feita ao fim de sua execução. A diferença entre o tempo final e o tempo inicial resulta no tempo de execução do aplicativo, conforme equação 7.

$$T_{execução} = T_{final} - T_{inicial}$$

Equação 5: Cálculo do tempo de execução

$$T(s) = \frac{Clock\ Ticks}{CPU\ Freq(Hz)}$$

Equação 6: Conversão Clock Ticks em segundos

O tempo de execução também pode ser calculado de forma mais simples, através da função *System.nanoTime()*. De forma análoga a anterior, são obtidas amostras no início e ao fim da execução, e a diferença entre elas corresponde ao tempo de execução.

O coletor de amostras, por sua vez, se comunica com a interface de bateria, e obtém amostras de corrente e tensão através do arquivo descrito no Capítulo 2, Seção 2.2.2. A taxa de amostras obtida com esse método é de aproximadamente 6 amostras/segundo. Esse dado foi obtido através dos experimentos relatados no Capítulo 5.

4.2.1 Fluxograma

O fluxograma de execução da SEMA está representado na figura 19. Ao ser iniciado, é registrado o tempo inicial, para futuro cálculo do tempo de execução. Após, são iniciadas as leituras de amostras, que são realizadas constantemente enquanto o serviço estiver ativo.

Figura 19: Fluxograma de execução da SEMA

Quando o serviço é finalizado pelo aplicativo avaliado, o tempo final é lido, e o tempo de execução é calculado. Então, a potência média, calculada com base nas amostras obtidas pelo Coletor de Amostras é multiplicada pelo tempo de execução, conforme equação 2, obtendo assim, a estimativa de consumo de energia.

Antes de finalizar, a ferramenta gera um *log*, que contém a energia consumida, o tempo de execução, a potência média e os valores das amostras coletadas, conforme exemplo da figura 20.

```
Tempo de Execucao (ms): 16891120883;
Potencia Media: 1.0836752941176437;
Potencia Acumulada: 184.22479999999942;
Energia Consumida:18.30449039
Numeros de Amostras: 170;
Amostras:
4.882,0.245;4.882,0.245;4.882,0.245;4.882,0.245;4.882,0.245;4.882
,0.245;4.882,0.245;4.882,0.245;4.882,0.245;4.882,0.245;4.882,0.24
5;4.882,0.245;4.882,0.245;4.882,0.245;4.882,0.245;4.882,0.245;4.8
82,0.245;4.882,0.245;4.882,0.245;4.882,0.245;4.882,0.245;4.882,0.
245;4.882,0.245;4.882,0.245;4.882,0.245;4.882,0.245;4.882,0.245;4
.882,0.245;4.882,0.245;4.882,0.245;4.882,0.245;4.882,0.245;4.882,
0.245;4.882,0.245;4.882,0.245;4.882,0.245;4.882,0.245;4.882,0.245
;4.882,0.245;4.882,0.245;4.882,0.245;4.882,0.245;4.882,0.245;4.88
2,0.245;4.882,0.245;4.882,0.245;4.882,0.245;4.882,0.245;4.882,0.2
45;4.882,0.245;4.882,0.245;4.882,0.245;4.882,0.245;4.882,0.245;4.
882,0.245;4.882,0.245;4.882,0.245;4.882,0.245;4.882,0.245;4.882,0
.245;4.882,0.245;4.882,0.245;4.882,0.245;4.882,0.245;4.882,0.245;
4.882,0.245;4.882,0.245;4.882,0.245;4.882,0.245;4.882,0.245;4.882
,0.245;4.882,0.245;4.882,0.245;4.882,0.245;4.882,0.245;4.882,0.24
5;4.882,0.245;4.882,0.245;4.882,0.245;4.882,0.245;4.882,0.245;4.8
82,0.245;4.882,0.245;4.882,0.245;4.882,0.245;4.882,0.245;4.882,0.
245;4.882,0.245;4.882,0.245;4.882,0.245;4.882,0.245;4.936,0.19;4
.882,0.245;4.882,0.245;4.882,0.245;4.882,0.245;4.936,0.19;4.936,0
.19;4.936,0.19;4.936,0.19;4.936,0.19;4.936,0.19;4.936,0.19;4.936,
0.19;4.936,0.19;4.936,0.19;4.936,0.19;4.936,0.19;4.936,0.19;4.936
,0.19;4.936,0.19;4.936,0.19;4.936,0.19;4.936,0.19;4.936,0.19;4.93
6,0.19;4.936,0.19;4.936,0.19;4.936,0.19;4.936,0.19;4.936,0.19;4.9
36,0.19;4.936,0.19;4.936,0.19;4.936,0.19;4.936,0.19;4.936,0.19;4.
936,0.19;4.936,0.19;4.936,0.19;4.936,0.19;4.936,0.19;4.936,0.19;4
.936,0.19;4.936,0.19;4.936,0.19;4.936,0.19;4.936,0.19;4.936,0.19;
4.936,0.19;4.936,0.19;4.936,0.19;4.936,0.19;4.936,0.19;4.936,0.19
;4.936,0.19;4.936,0.19;4.936,0.19;4.936,0.19;4.936,0.19;4.936,0.1
9;4.936,0.19;4.936,0.19;4.936,0.19;4.936,0.19;4.936,0.19;4.936,0.
19;4.936,0.19;4.936,0.19;4.936,0.19;4.936,0.19;4.936,0.19;4.936,0
.19;4.936,0.19;4.936,0.19;4.936,0.19;4.936,0.19;4.936,0.19;4.936,
0.19;4.936,0.19
```

Figura 20: Modelo de *log* da ferramenta SEMA

4.2.2 Estrutura da ferramenta

Para a implementação da ferramenta proposta, foram desenvolvidas quatro classes: *EnergyMonitor, BatteryService*, *GPIO* e *Functions*. Além destas classes, a implementação reusa recursos das classes Android *Intent*, *Timer*, *Context* e *Service*, conforme ilustrado no diagrama de classes da figura 21.

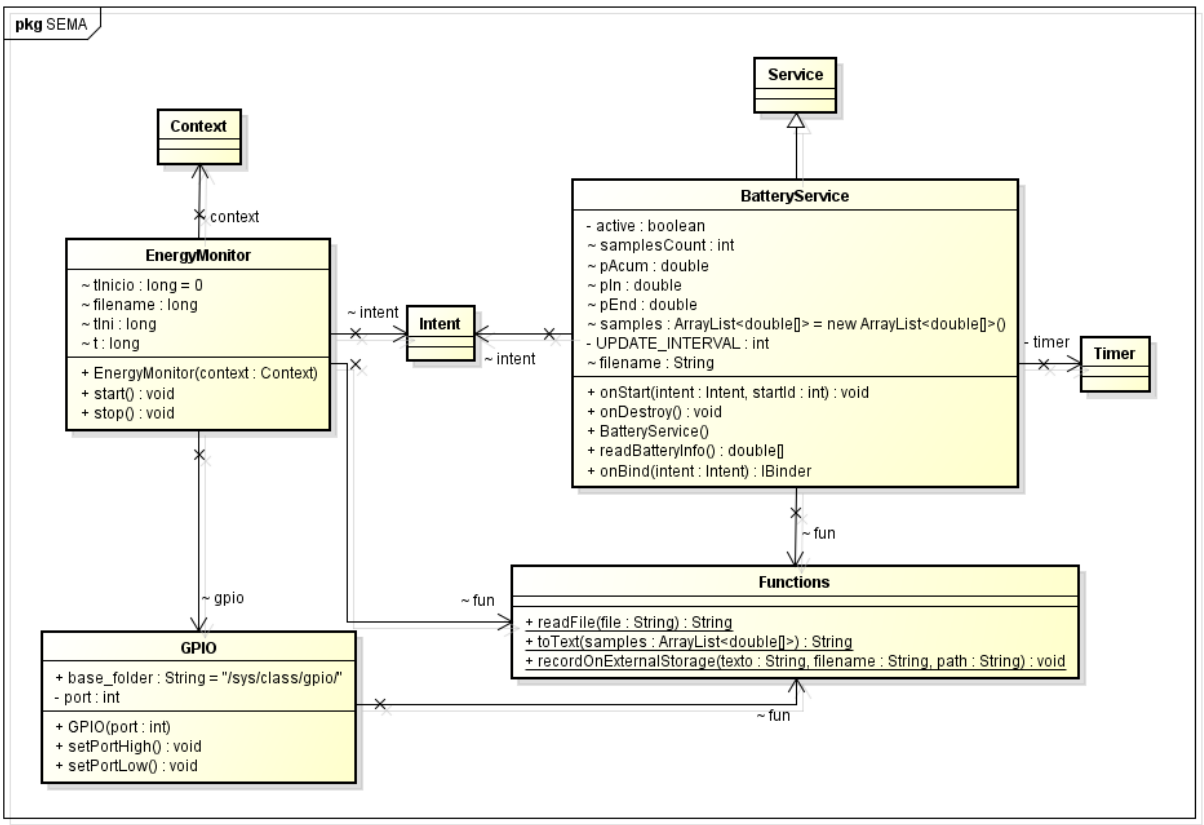

Figura 21: Diagrama de Classes

A classe principal é a *EnergyMonitor*, a qual corresponde ao Gerente de Execução e contém também as funções do Monitor de Tempo de Execução. Através da classe *Context*, obtém o contexto do aplicativo monitorado, o que permite saber, por exemplo, o PID do processo desse aplicativo. Através de uma *Intent*, se comunica com o *BatteryService*, que é o serviço de coleta de amostras. É através dessa *Intent* que o serviço recebe os comandos de *start* e *stop*, bem como nome do arquivo onde deverão ser salvas as amostras coletadas durante sua execução. Essa classe estende a classe *Service* do Android, a qual permite executar em segundo plano utilizando um processo diferente da aplicação que o iniciou. A classe *Timer* fornece um escalonador, que permite definir a frequência de leitura das amostras.

As classes GPIO e *Functions* são classes auxiliares, que contém recursos de suporte as classes principais. A classe GPIO foi utilizada nos testes de validação, para ativar o dispositivo de medição externa. Sua função é setar o valor de um determinado pino de GPIO da placa utilizada nos testes. O valor do pino colocado em “1” ativa o circuito de medição, ação realizada simultaneamente ao disparo do serviço de coleta de amostras. Quando o serviço de coleta de amostras é finalizado, o pino é setado

em “0” indicando que o circuito irá parar de medir a potência consumida pelo dispositivo.

A classe *Functions* por sua vez, traz algumas funções úteis as demais classes, tais como *recordOnExternalStorage*, *readFile* e *toText. A recordOnExternalStorage é* utilizada para gravar o *log* na memória externa do dispositivo, enquanto a função *readFile* é usada para ler os arquivos de status do processo e da bateria. toText é uma função utilizada para converter o *array* de amostras coletadas em texto para ser adicionado ao log. O *log* gerado utiliza como padrão de nome de arquivo o *timestamp* do sistema no momento em que a ferramenta é iniciada, assim, evitando duplicações no caso de múltiplas execuções em sequência.

4.2.3 Código do serviço

Os principais métodos do serviço de coleta de amostras, implementados na classe *BatteryService*, são *onStart(), onDestroy()* e *readBatteryInfo()*. Os dois primeiros estão diretamente relacionados ao ciclo de vida do serviço, enquanto o terceiro é o responsável por ler os dados da bateria. Não foi realizada a implementação do método *onCreate()*, pois o processamento necessário para as coletas deve ser realizado assim que o serviço inicia sua execução, e não quando é criado. Além disso, o método *onBind()*, não é utilizado, pois não é permitido que outras aplicações se conectem ao serviço.

No método *onStart()*, representado na figura 22, inicialmente são lidos os dados recebidos através do Gerente de Execução, através de uma *Intent*, afim de definir o nome que será utilizado no arquivo de *log*, e a frequência das leituras. A *flag Active* é setada como “*true*” indicando que o serviço está ativo, e então é realizada a leitura da primeira amostra de tensão e corrente. A amostra obtida é armazenada no vetor “*samples*”, e é acumulada na variável *pAcum*. A quantidade de amostras é incrementada a cada nova leitura na variável *samplesCount*. Um escalonador é ativado através da função *Timer*, de acordo com o período definido em UPDATE_INTERVAL. O método *run()* é executado de acordo com esse período, e as leituras são realizadas enquanto a variável *Active* mantém o valor “*true*”.

```
public void onStart (Intent intent, int startId) {

    this.intent = intent;
    filename = intent.getStringExtra("filename");
    UPDATE_INTERVAL = intent.getIntExtra("update_interval", 100);

    active = true;
    double batinfo[] = readBatteryInfo();
    double p = batinfo[0] * batinfo[1];
    samples.add(batinfo);
    pIn = p;
    pAcum = p;
    samplesCount = 1;

    timer.scheduleAtFixedRate(new TimerTask() {
        @Override
        public void run() {
            if (active) {
                double batinfo[] = readBatteryInfo();
                double p = batinfo[0] * batinfo[1];
                samples.add(batinfo);
                pAcum = pAcum + p;
                samplesCount++;
            }
        }
    }, 0, UPDATE_INTERVAL);
}
```

Figura 22: Método *onStart()* da classe BatteryService

O método *onDestroy()*, apresentado na figura 23, é chamado quando o serviço irá ser parado (vide Ciclo de Vida do Serviço, Figura 6). Nesta situação, a variável *Active* é setada como "*false*", indicando o fim das leituras de amostras. A seguir, uma última amostra é lida, e é calculada a potência média, com base na potência acumulada e no número de amostras lidas. Antes de chamar a função *stopSelf()*, que encerra o serviço, a potência média, a potência acumulada, o número de amostras e as amostras lidas são salvas no arquivo de *log*.

```
public void onDestroy() {
    active = false;
    double batinfo[] = readBatteryInfo();
    double p = batinfo[0] * batinfo[1];
    samples.add(batinfo);
    pAcum = pAcum + p;
    samplesCount++;
    double pMed = pAcum/samplesCount;

    String textToRecord =
              "Potencia Media: " + pMed
            + ";\nPotencia Acumulada: " + pAcum
            + ";\nNumeros de Amostras: " + samplesCount
            + ";\nAmostras: \n" + Functions.toText (samples);

    Functions.recordOnExternalStorage(textToRecord, filename+".txt",getExternalFilesDir(null).getPath());
    stopSelf();
}
```

Figura 23: Método *onDestroy* da classe *BatteryService*

É através do método *readBatteryInfo()*, que são realizadas as leituras de amostras de tensão e corrente da bateria, que serão multiplicadas para obter as amostras de potência. A função auxiliar *readFile*, da classe *Functions* é utilizada para ler os arquivos conforme o caminho armazenado nas variáveis volt_file e current_file. Os valores são lidos em µV (micro Volts) e µA (micro Ampéres), e são convertidos para valores de tensão em Volts e corrente em Ampéres diretamente. A seguir são armazenados em um vetor de duas posições, que será utilizado como variável de retorno da função. A figura 24 apresenta o código do método *readBatteryInfo()*.

```
public double[] readBatteryInfo(){
    String volt = null;
    String current = null;

    //Cubieboard 2.0
    String volt_file = "/sys/class/power_supply/ac/voltage_now";
    String current_file = "/sys/class/power_supply/ac/current_now";
    volt = Functions.readFile(volt_file);
    current = Functions.readFile(current_file);

    double v = (Double.parseDouble(volt))/1000000;
    double a = (Double.parseDouble(current))/1000000;

    double rets[] = new double[2];
    rets[0] = v;
    rets[1] = a;

    return rets;
}
```

Figura 24: Método *readBatteryInfo()* da classe BatteryService

4.3 Considerações Gerais

Nesta seção, faz-se um comparativo entre a abordagem proposta e os principais trabalhos relacionados. Este comparativo é sumarizado na tabela 4. Quanto a técnica utilizada, a SEMA se assemelha as abordagens Aneprof, NEAT e o *framework* JouleUnit, visto que estas baseiam-se também em medições. A diferença com relação ao Aneprof é quanto ao método de aquisição de dados. Enquanto Aneprof utiliza medição externa, nossa abordagem utiliza dados extraídos do próprio dispositivo. O NEAT permite a captura de dados de consumo internamente, mas para isso um módulo deve ser incorporado a bateria. Já JouleUnit não apresenta elevada granularidade, porém é capaz de estimar o consumo dos principais componentes de *hardware* do sistema, o que não é possível com o SEMA.

Tabela 4: Comparativo entre SEMA e demais abordagens

Abordagem	**Técnica Utilizada**	**Granularidade**	**Por Componente de *Hardware***	**Aquisição de dados**
PowerTutor	Modelo	Aplicação	Sim	Não aplicável
JouleUnit	Medição	Aplicação	Sim	Interna
Aneprof	Medição	Método	Não	Externa
AppScope	Modelo	Aplicação	Sim	Interna
SeSaMe	Modelo	Sistema	Não	Interna
Eprof	Modelo	Método	Sim	Não aplicável
NEAT	Medição	Método	Sim	Externa
AndroProf	Modelo	Método	Não	Não aplicável
SEMA	**Medição**	**Método**	**Não**	**Interna**

PowerTutor, Eprof diferenciam-se da abordagem proposta por serem baseadas em modelos e não utilizarem dados da bateria para alimentar seus modelos. As abordagens baseadas em modelos, com exceção do SeSaMe e AndroProf, são capazes de estimar o consumo por componente do sistema, característica não atendida na abordagem apresentada. No entanto, a abordagem SEMA é independente de um modelo específico de dispositivo e pode ser facilmente portada para diferentes dispositivos.

Quanto a granularidade, SEMA possui granulariedade no nível de método ou função, similar ao AneProf, EProf, NEAT e AndroProf. As demais abordagens/ferramentas trabalham com granularidade maior o que não é muito interessante para desenvolvedores.

Outra característica relevante, é que a SEMA não necessita alterações no SO, diferente do AppScope, onde há necessidade de adicionar módulos para instrumentar o *kernel* do sistema operacional.

5 EXPERIMENTOS E RESULTADOS

Esse capítulo tem por objetivo apresentar e detalhar os experimentos realizados para avaliar a abordagem proposta para a estimativa do consumo energético de aplicativos Android. Primeiramente, a metodologia empregada é detalhada na Seção 5.1. Na Seção 5.2, experimentos com aplicativos simples de ordenamento de dados e com um MP3 Player, implementado através de bibliotecas nativas do Android, são apresentados, bem como os resultados de nossa abordagem são comparados a resultados obtidos com outra ferramenta de avaliação de aplicativos. Na Seção 5.3 a precisão dos resultados obtidos pelos experimentos é discutida. Por fim, os resultados obtidos são analisados e discutidos na Seção 5.4.

5.1 Metodologia

Em todos os experimentos, os testes foram repetidos 30 vezes, e os resultados apresentados neste capítulo, consistem na média dessas execuções. No entanto, no Apêndice B, os dados completos dos experimentos podem ser encontrados, incluindo todos os valores obtidos e não apenas valores médios. Além das médias calculadas, também é calculado e analisado o desvio padrão de cada teste. A análise estatística dos resultados foi realizada através do método estatístico *t student*, com 95% de confiança, utilizando a ferramenta GraphPad QuickCalcs (GRAPHPAD, 2015). Os aplicativos escolhidos realizam ordenamento de dados usando diferentes algoritmos de ordenação, já utilizados em Vieira et al. (2012), além de um MP3 Player, desenvolvido através da classe *MediaPlayer* do Android.

Para avaliar a abordagem proposta foi utilizada uma placa de desenvolvimento, Cubieboard A20 (ARM A7 Dual Core, RAM 1GB DDR3, GPU ARM Mali400MP2, NAND 4GB), com Android 4.2.2 (CUBIEBOARD, 2015). Os resultados de potência média, energia e tempo de execução foram extraídos do *log* gerado ao final de cada execução. A frequência de coleta de amostras utiliza nos testes foi de 100ms. O tempo de execução foi obtido através do tempo de processo fornecido pelo Sistema Operacional, conforme descrito na Seção 4.2 do Capítulo 4.

Para fins de comparação, os mesmos aplicativos foram também avaliados com a ferramenta PowerTutor, que fornece uma estimativa de consumo energético.

Um circuito utilizando uma placa Arduino Uno foi empregado na medição da corrente e da tensão, para avaliação de precisão da medição, conforme descrito na Seção 5.3. Foi utilizado também um osciloscópio TBS1062, fabricado pela TekTronik, para verificar a precisão da medição realizada pelo circuito. Devido uma limitação no tamanho do arquivo de *log* gerado por este equipamento, não foi possível utilizá-lo nos testes comparativos. Assim, para validar as medidas realizadas através do Arduino, foi implementado um *loop* vazio conforme sugerido pelo Google (ANDROID, 2014l), e a queda de tensão sobre o resistor de *shunt* foi medido em amostras de 1s durante a execução desse *loop*.

5.2 Resultados Obtidos

Para demostrar a abordagem proposta (SEMA), foram desenvolvidos e avaliados, inicialmente, aplicativos que executam a ordenação de *arrays* através de conhecidos algoritmos de ordenação. Num segundo momento, foi avaliada a execução de um MP3 Player, pois trata-se de uma aplicação bastante utilizada em *smartphones* e *tablets*. As execuções desses aplicativos foram avaliadas e os resultados de tempo de execução, de consumo de potência e de energia foram obtidos através do emprego da abordagem SEMA.

Os algoritmos de ordenação utilizados nos experimentos foram: BubleSort, CountingSort, HeapSort, InsertionSort Iterativo e Recursivo e MergeSort. Esses algoritmos possuem diferentes complexidades, o que confere maior riqueza a análise de eficiência dos mesmos, visto que os tempos de execução dos algoritmos apresentam grande variação frente ao mesmo conjunto de dados. Já para o experimento com o MP3 Player, é utilizado um mesmo decodificar, variando a carga aplicada sobre ele com amostras de áudio de diferentes durações.

5.2.1 Algoritmos de Ordenamento

Os tempos de execução dos algoritmos, medido em segundos (s), foram extraídos do *log* gerado pela abordagem proposta e podem ser visualizados na figura 25. Observa-se que o tempo de execução do algoritmo BubbleSort é cerca de 28 vezes maior que o do algoritmo CoutingSort, representa a implementação mais rápida de ordenação avaliada em nossos experimentos.

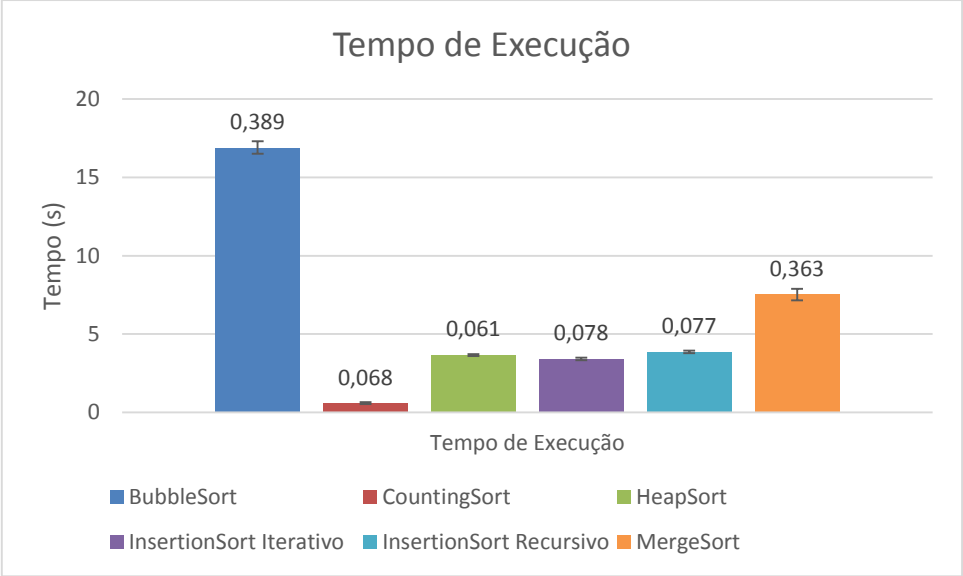

Figura 25: Tempo de execução dos algoritmos obtidos com SEMA

O gráfico ilustrado na figura 26, apresenta os resultados da medição de potência média consumida pelos aplicativos ao longo de sua execução, bem como os desvios padrões calculados são apresentados numericamente no topo das barras, e graficamente logo através da linha de erros abaixo. A análise estatística destes resultados indicou que para o algoritmo BubbleSort, há uma diferença estatisticamente significativa na potência média consumida quando comparada ao consumo dos demais algoritmos, variando de 15% a 33%. Para os demais algoritmos, não há diferença significativa nos consumos médios obtidos pela SEMA.

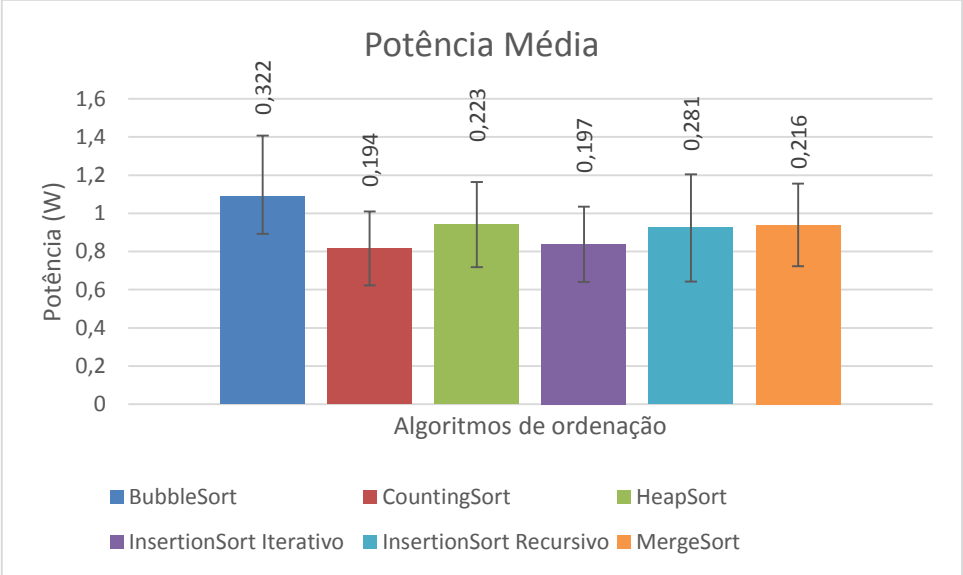

Figura 26: Potência medida pela SEMA

A figura 27 apresenta a média dos resultados obtidos para consumo energético para os diferentes aplicativos de ordenamento. A diferença entre os algoritmos acompanha as diferenças entre os tempos de execução. Contudo, a diferença entre os algoritmos BubbleSort e CountingSort aumenta para cerca de 38 vezes, isso porque a potência média consumida por esse algoritmo é maior que os demais algoritmos.

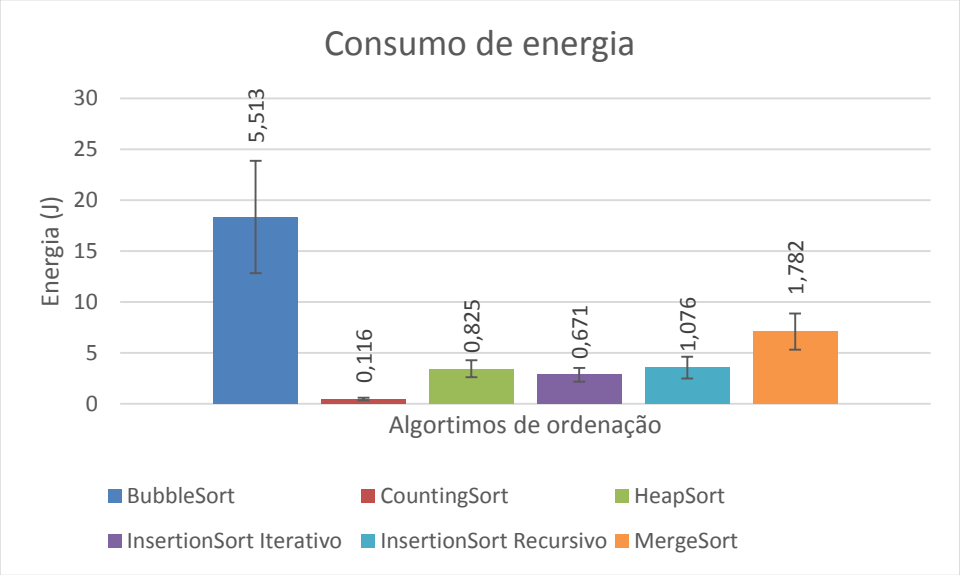

Figura 27: Consumo de energia medido através da SEMA

Apesar dos resultados obtidos nos experimentos com os aplicativos de ordenamento apresentarem boa possibilidade de análise e comparação com outros métodos, percebeu-se a necessidade de avaliar a SEMA também para aplicativos com tempos de execução maiores que aqueles obtidos nesse experimento. Para isso, foi desenvolvido um aplicativo MP3 *Player*, e foi realizada a variação da carga aplicada sobre ele para obter assim diferentes tempos de execução do aplicativo. Os resultados para esse experimento são apresentados na Seção 5.2.2.

5.2.2 MP3 Player

Como o consumo energético está muito associado ao tempo de execução dos aplicativos, experimentos também foram realizados para avaliar a ferramenta SEMA enquanto esta analisa um aplicativo com tempo de execução variavel em função da carga, visando principalmente discutir se esta variação traria impacto na precisão dos resultados.

Para isso, foi desenvolvido um aplicativo MP3 *Player*, implementado com uso da classe *MediaPlayer* do Android e que representa uma aplicação bastante usada em smartphones e tablets. Arquivos de áudio com durações variando entre 30 segundos, 1, 2 e 3 minutos foram executados em primeiro plano, e o consumo energético para cada um deles foi medido com a SEMA.

O objetivo desse experimento foi verificar se o comportamento da ferramenta se mantém quando utilizada em experimentos com tempos de execução maiores que aqueles obtidos nos testes com os algoritmos de ordenação.

Os tempos de execução nesse experimento são muito próximos a duração do áudio utilizado. O tempo médio de execução medido pela SEMA para os áudios de 30s, 1min, 2min e 3min estão representados na tabela 5.

Tabela 5: Tempo médio de execução

	Tempo Real de Execução (s) medido pela SEMA	**Desvio Padrão**
30 segundos	30,03	0,27
1 minuto	60,14	0,27
2 minutos	120,47	0,21
3 minutos	180,57	1,06

A figura 28 apresenta os valores de potência média medidos com a SEMA. Observa-se aqui uma diminuição da potência média conforme o tempo de execução aumenta. Esse comportamento é normal, pois a potência é maior no carregamento do aplicativo (PATHAK et al., 2012), o que pode ser observado na figura 29 que mostra o consumo de potência medido pela SEMA ao longo da execução do aplicativo. Conforme aumenta o tempo de execução, o impacto desse pico é menor e, portanto, reduz a potência média. A variação da potência média encontrada para os diferentes arquivos de áudio é de 13% a 38%. O único caso onde a diferença não é considerada estatisticamente significativa é entre a amostra de áudio de 30 segundos e 1 minuto.

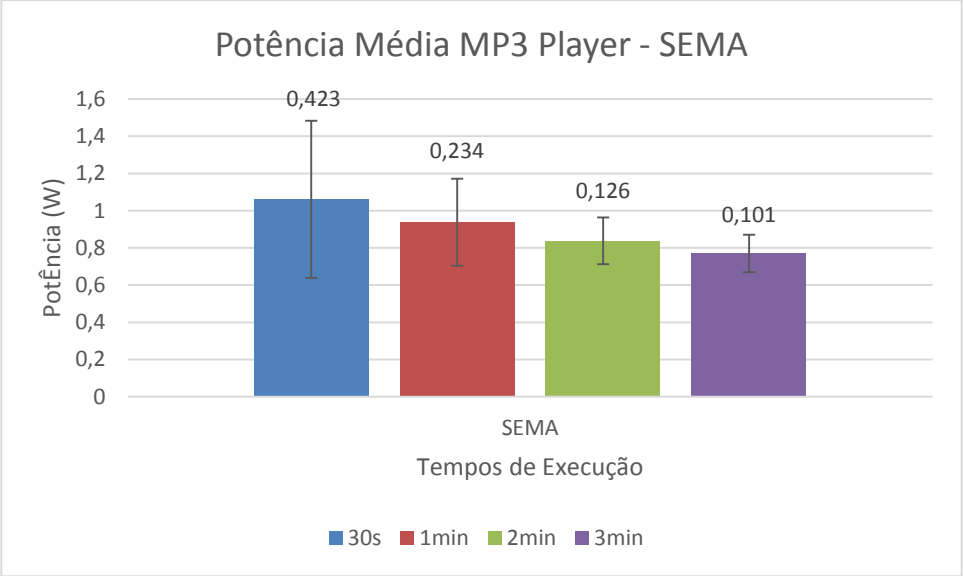

Figura 28: Potência Média para o MP3 Player medida com a SEMA

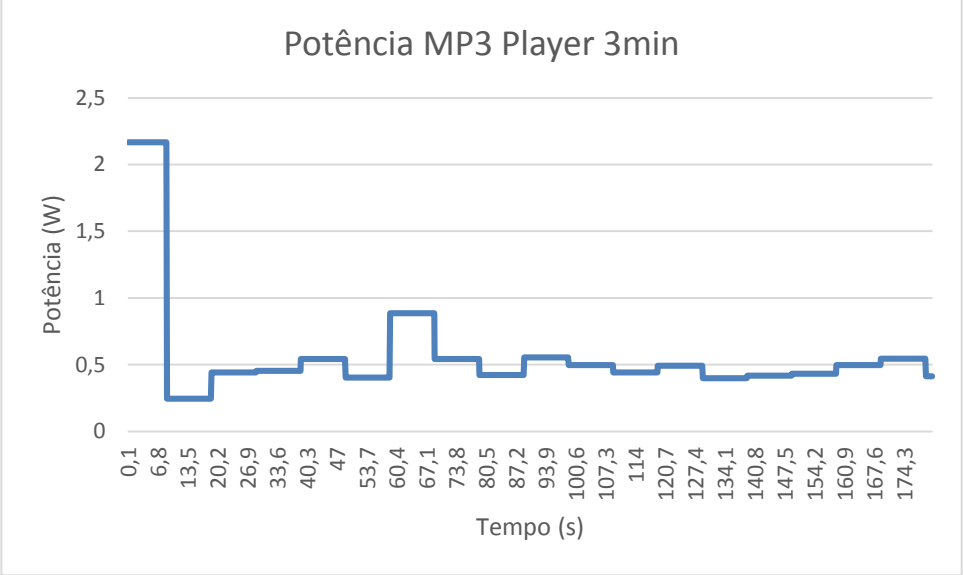

Figura 29: Potência medida pela SEMA para o MP3 Player

Para o consumo de energia, os resultados são apresentados na figura 30. Percebe-se que o aumento de consumo, nesse experimento, está relacionado ao aumento do tempo de execução, ainda que a potência média diminua conforme esse tempo aumenta. As diferenças entre as médias de consumo energético mais significativas entre as diferentes amostras de áudio, são de 44% comparando a amostra de 30s com a de 1min representando a menor diferença, e de 77% entre 30 segundos e 3 minutos, representando a maior diferença.

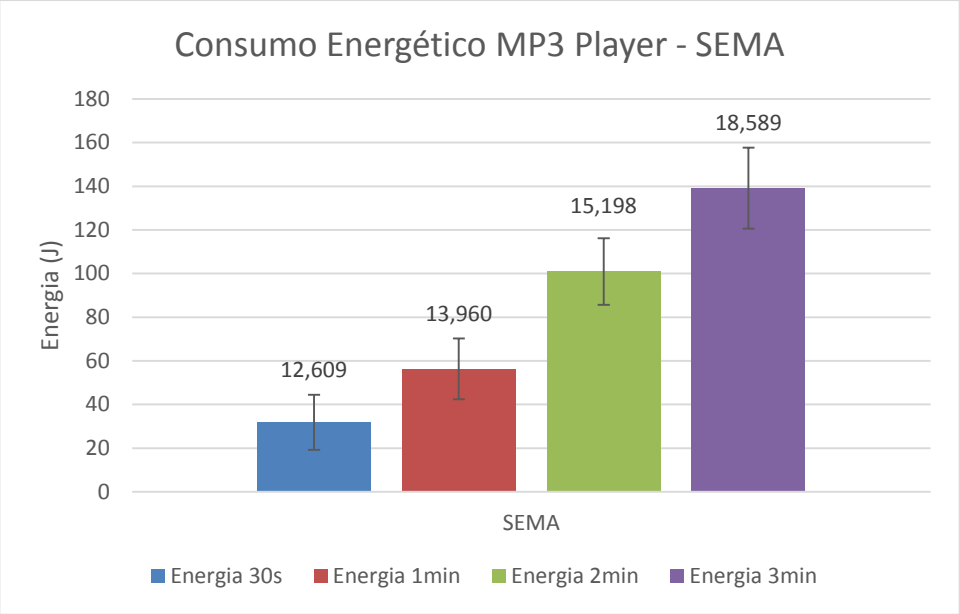

Figura 30: Consumo de energia do MP3 Player medido através da SEMA

5.2.3 Comparativo com PowerTutor

Nesta seção, os resultados obtidos por nossa abordagem são comparados com os resultados fornecidos pela ferramenta PowerTutor. Desta forma, a SEMA é comparada a uma outra abordagem de estimativa de consumo para aplicativos Android, que emprega modelos de mais alto nível de abstração e possui granularidade similar.

O gráfico comparativo ilustrado na figura 31 apresenta os resultados obtidos com SEMA e PowerTutor, onde é possível observar a diferença entre os valores obtidos por estas para o experimento com os algoritmos de ordenamento. Inicialmente, observa-se que a ferramenta PowerTutor apresenta, na maioria dos casos, valores de energia menores que aqueles medidos através da SEMA.

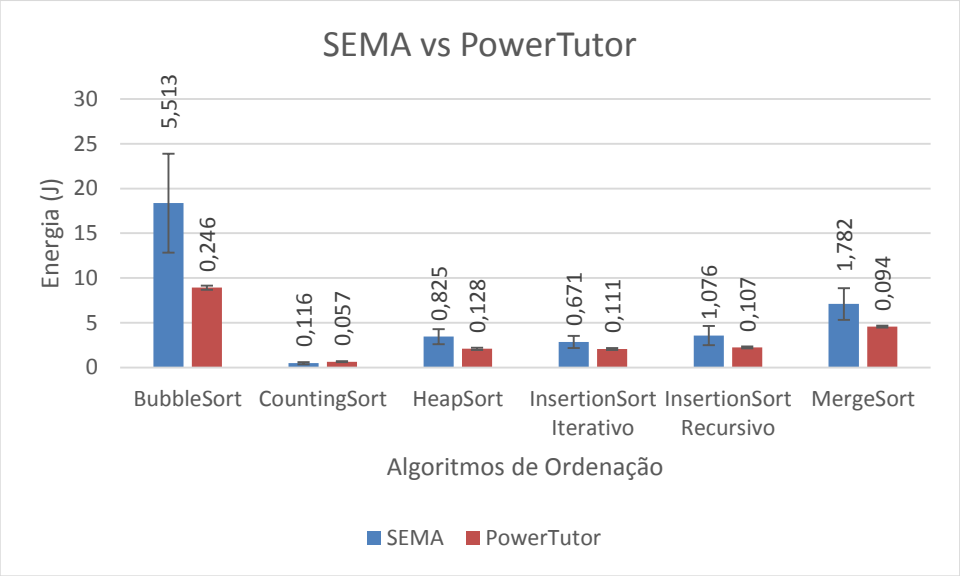

Figura 31: Gráfico comparativo entre SEMA e PowerTutor para os algoritmos de ordenamento

A diferença média entre as medidas dos dois métodos é de 26,14%. A maior diferença entre eles é verificada no algoritmo BubbleSort, onde chega a 51,42%. O caso do CountingSort é o único onde o valor medido pela SEMA é menor que a estimativa do PowerTutor, e a diferença é de 34,4%.

Quando a comparação entre as duas ferramentas/abordagens é realizada em experimentos com maior tempo de execução, percebe-se que a diferença entre elas é ainda maior. A diferença entre elas é de 93% para o áudio de 3 minutos, e de 94% para os demais, conforme ilustrado na figura 32. No entanto, como a precisão da ferramenta PowerTutor é desconhecida, na próxima seção, é empregado um método de medição menos abstrato para avaliar e discutir a precisão da abordagem proposta.

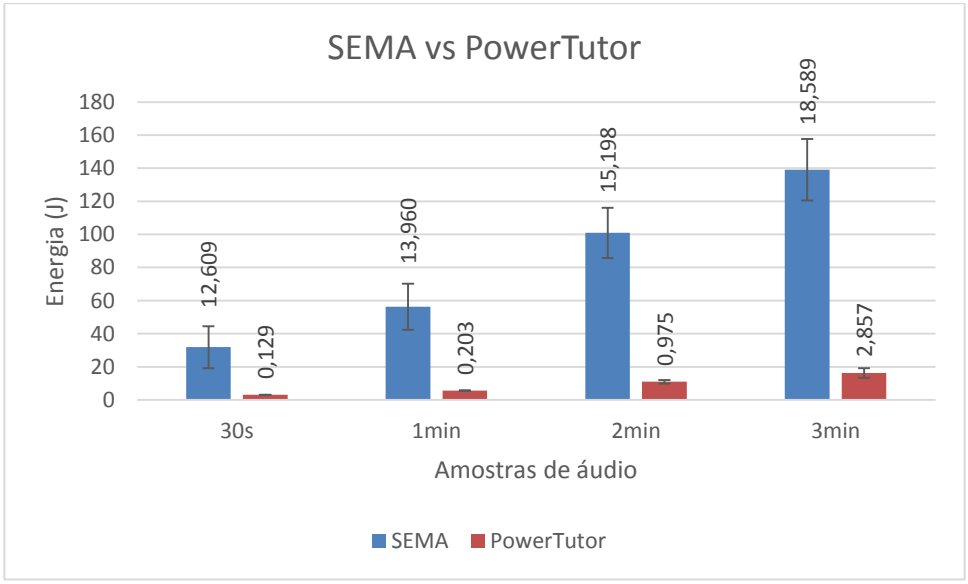

Figura 32: Comparativo entre SEMA e PowerTutor para o MP3 Player

5.3 Avaliação de precisão

Para avaliar a precisão da SEMA a comparação foi estendida também para um método de medição de menor nível de abstração. Para isso, foi desenvolvido um circuito baseado em uma placa Arduino, o qual é capaz de ler amostras da tensão que alimenta um dispositivo, bem como a corrente que circula através dele através de um resistor *shunt*, permitindo assim calcular a potência média consumida pelo circuito.

5.3.1 Método de avaliação

Para avaliação de precisão foi desenvolvido um circuito, ilustrado na figura 33, o qual utiliza um resistor de *shunt* em série com o circuito a ser medido. O resistor utilizado possui valor 0,68 Ohm com precisão de +/- 1%. A queda de tensão sobre esse resistor é proporcional a corrente que circula pelo circuito de acordo com a Lei de Ohm. Assim, a corrente pode ser calculada através da Lei de Ohm, que relacionada essas grandezes matematicamente através da equação 7.

$$\boldsymbol{R = \frac{V}{I}}$$

Equação 7: Primeira Lei de Ohm

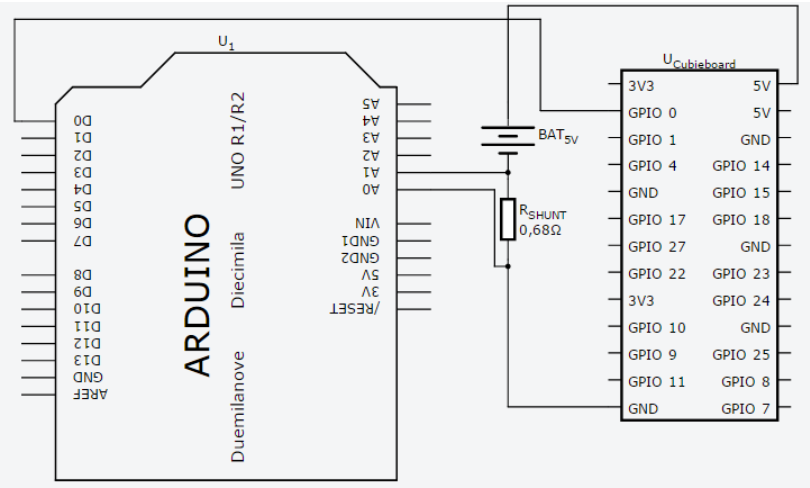

Figura 33: Circuito de medição com Arduino

A placa Arduino é conectada através das entradas analógicas A0 e A1 ao circuito a ser medido. A entrada A0 é conectada entre o resistor e a fonte de alimentação, representando a queda de tensão sobre ele, valor que vai ser utilizado para cálculo da corrente. A entrada A1 é conectada entre a o circuito alvo e a fonte, representando a queda de tensão sobre o circuito. A entrada digital D0 é conectada a

um pino de GPIO da placa Cubieboard que foi utilizada nos testes. Essa ligação permite que a Cubieboard dispare a medição, enviando um sinal através da GPIO para a entrada do Arduino. Enquanto a medição estiver ativa, o Arduino envia os valores medidos via serial a um computador onde os dados são coletados.

A precisão desta avaliação com o Arduino foi verificada, comparando a tensão sobre o resistor medida por este método, a tensão obtida através de um osciloscópio. Os resultados obtidos através desses experimentos são apresentados na tabela 6. A diferença entre a média dos resultados do Arduino e do osciloscópio é de 2,46% na média, e não é considerada estatisticamente significativa pela análise do método *t student*. Assim, pode-se afirmar que a medição realizada através do Arduino é próxima o bastante aos valores de tensão sobre o resistor e sobre o circuito, e consequentemente, a potência medida por este método, apresenta boa precisão, podendo ser utilizada para avaliar a precisão de outros métodos ou ferramentas de estimativa.

Tabela 6: Comparativo entre medições do Arduino e Osciloscópio

	Tensão sobre resistor shunt	**Desvio Padrão**
Circuito de Medição com Arduino	0,023422689	0,0012791
Osciloscópio	0,024014059	0,0018101

5.3.2 Análise da precisão da SEMA

Verificada a precisão desta medição realizada com o Arduino, este circuito foi utilizado para medir a potência da Cubieboard durante a execução dos algoritmos de ordenação implementados em Java-Android, permitindo assim, a avaliação e análise da precisão dos resultados obtidos por nossa abordagem.

A figura 34 apresenta os resultados para potência obtidos pela SEMA e pelo Arduino para os aplicativos de ordenação. A comparação entre os métodos de medição mostra diferenças que variam de 28% no algoritmo BubbleSort, a 51% no algoritmo CountingSort. A diferença média entre os dois métodos é em torno de 40% nos experimentos realizados com aplicativos de ordenação.

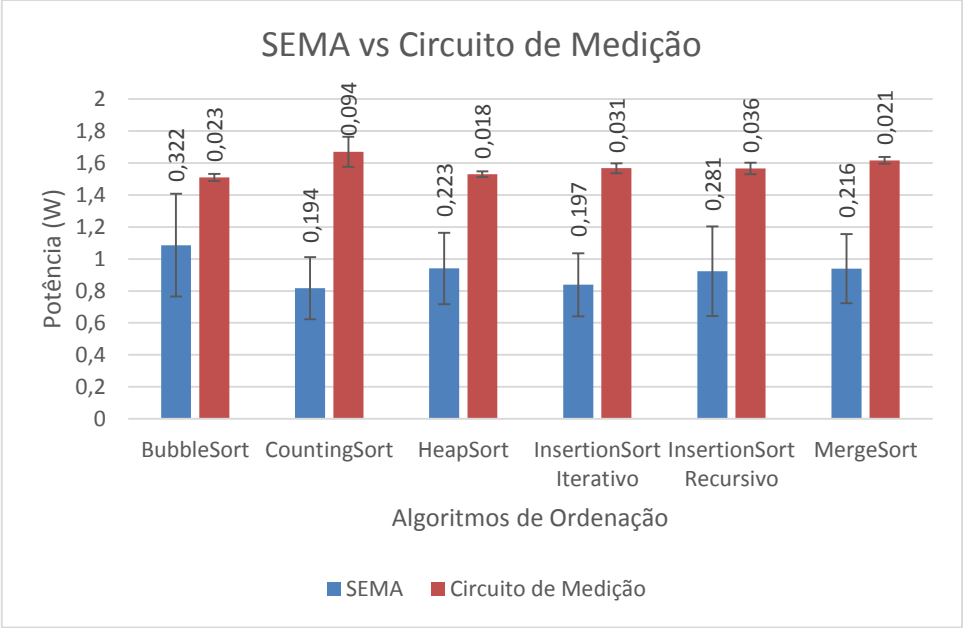

Figura 34: Potência média - SEMA vs Circuito de Medição com Arduino

Quando calculada a energia média consumida, observa-se que os valores obtidos pelo Arduino são sempre superiores ao obtidos por nossa abordagem, como pode ser visto na figura 35. As diferenças mais significativas entre eles estão nos algoritmos que apresentam maior e menor consumo energético que são BubbleSort e Couting Sorting, e a diferença é a mesma encontrada na avaliação da potência, ou seja, 28% e 51% respectivamente.

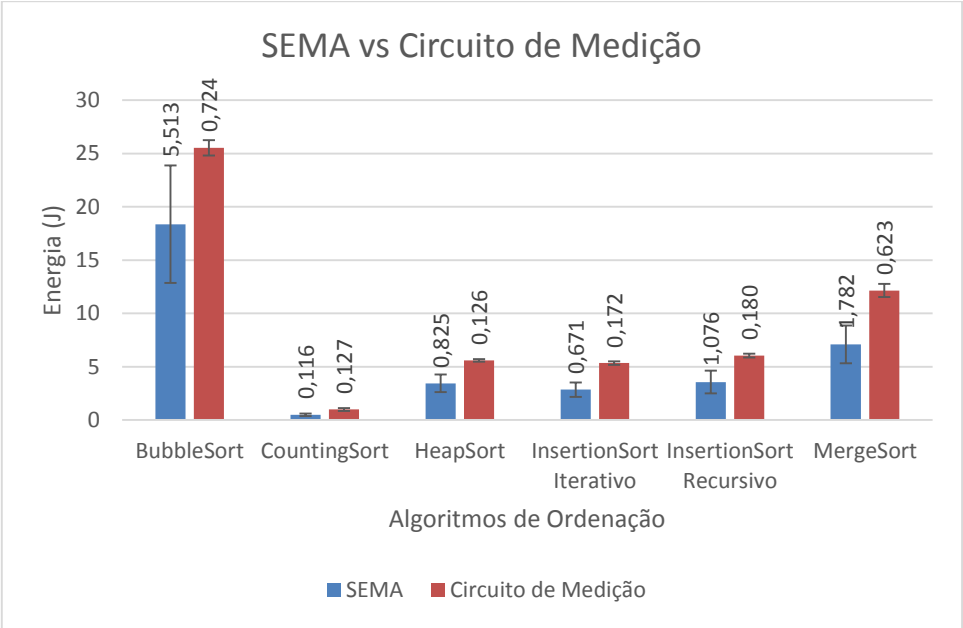

Figura 35: Energia Consumida - SEMA vs Circuito de Medição com Arduino

Para os experimentos com o MP3 Player, a potência média estimada pela SEMA também foi comparada a potência média obtida pelo circuito de medição com Arduino. Nesses experimentos, a potência média obtida pelos dois métodos difere entre 38% para os áudios de 30s e 1 minuto, e 41% para os áudios de 2 e 3 minutos, o que pode ser observado no gráfico da figura 36.

.

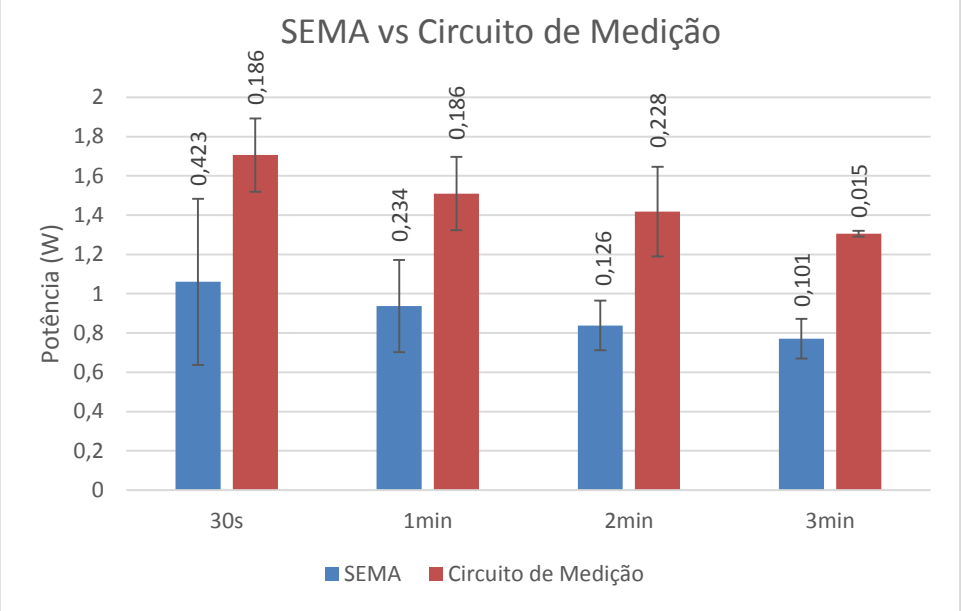

Figura 36: Potência Média - SEMA vs Circuito de Medição com Arduino

Quando avaliado o consumo médio de energia para o MP3 Player (apresentado na figura 37), são encontradas as mesmas diferenças na comparação da potência média, ou seja, de 38% a 41%. A diferença média nos consumos energéticos é de 39,5% nos experimentos realizados com este aplicativo, e é próxima da diferença média encontrada nos experimentos com os algoritmos de ordenamento que é de 41%. Percebe-se então, que a SEMA apresenta uma precisão média de 40,25% nos experimentos realizados.

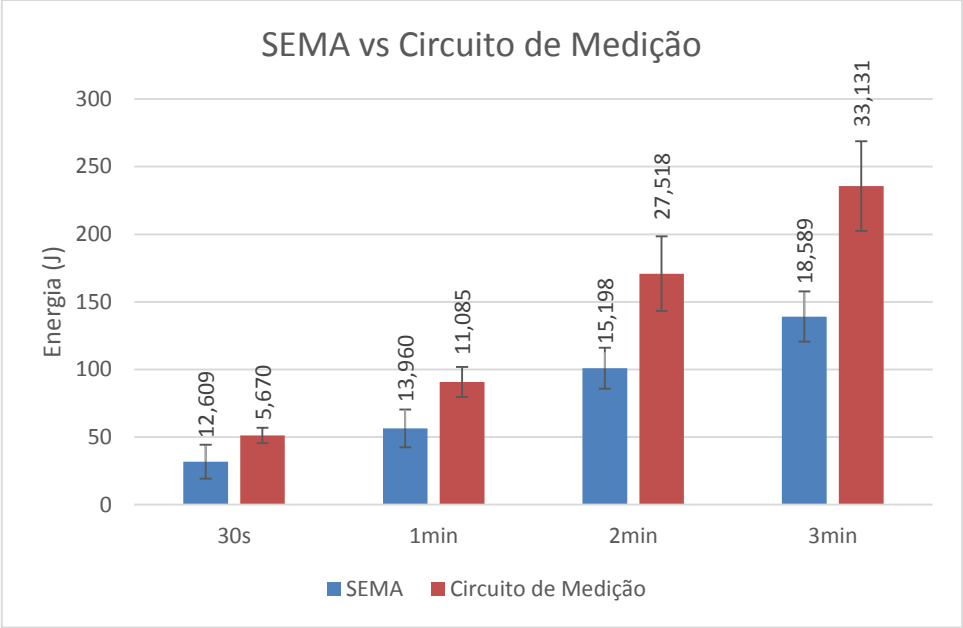

Figura 37: Comparativo entre SEMA e Circuito de Medição com Arduino

5.4 Comparativo entre as abordagens

Os resultados obtidos nos experimentos com a SEMA, PowerTutor e Circuito de Medição também foram comparados entre si, e essa comparação é apresentada graficamente nas figuras 38 e 39.

Observa-se que tanto para os algoritmos de ordenação quanto para o MP3 Player, existe uma correlação entre os resultados, onde o Circuito de Medição que representa os valores reais está sempre acima da SEMA e do PowerTutor. Já o PowerTutor apresentou sempre valores inferiores, com diferença de aproximadamente 62% para os algoritmos de ordenação, e até 94% no MP3 Player. A exceção é o caso do CountingSort, nos algoritmos de ordenação, onde a SEMA apresentou o menor resultado.

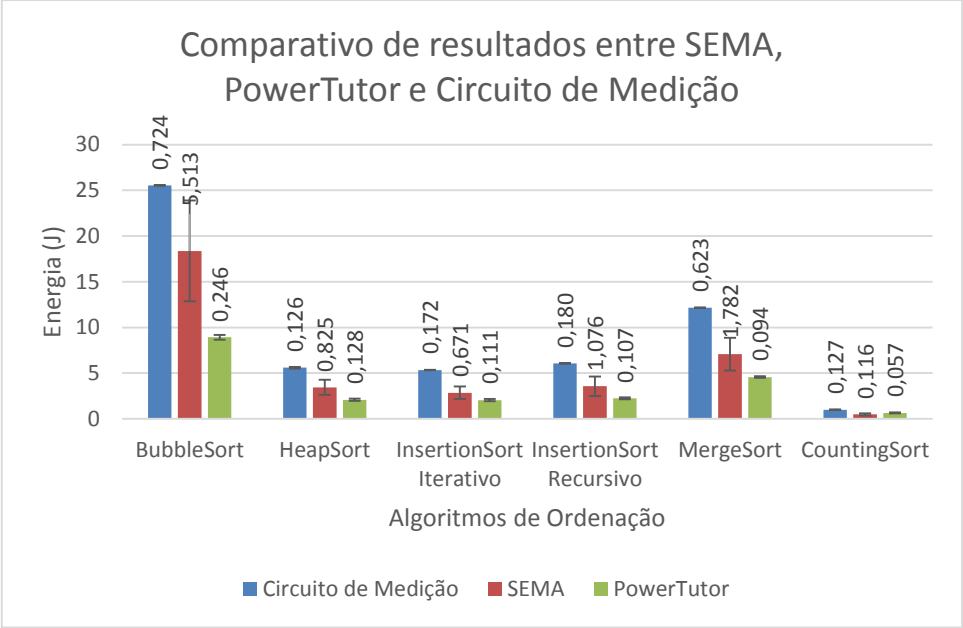

Figura 38: Comparativo dos resultados das abordagens para os algoritmos de ordenação

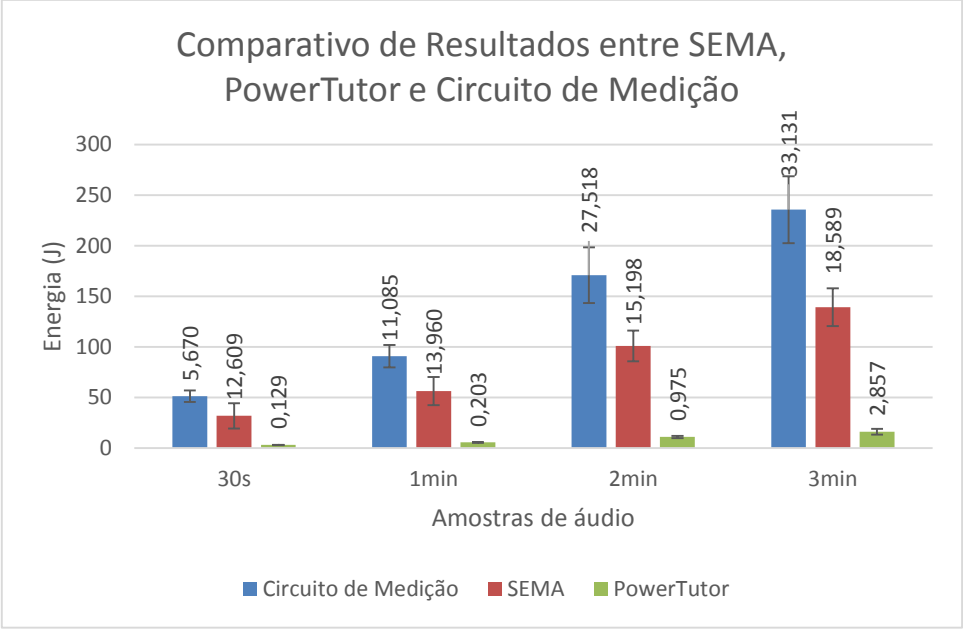

Figura 39: Comparativo dos resultados das abordagens para o MP3 Player

5.5 Análise dos Resultados

Nos experimentos conduzidos, observa-se grande diferença entre os valores de potência média e consumo energético medido através dos diferentes métodos avaliados. Na comparação entre a SEMA e a medição com Arduino, percebe-se que a diferença entre os valores encontrados mantém uma média em torno de 40%, tanto entre os algoritmos de ordenação quanto no MP3 Player.

A ferramenta PowerTutor baseia-se em modelos e por isso apresentou resultados pouco precisos nos experimentos, não se mostrando um bom comparativo. Comparando nossa abordagem com a ferramenta Aneprof que utiliza também uma técnica de medição, observa-se apenas uma diferença de 12%, considerando a precisão de 28% indicada em (CHUNG, LIN e KING, 2011). Ainda que essa diferença seja significativa, a SEMA não depende de ferrramentas de instrumentação externa como no Aneprof, e ainda é capaz de fornecer resultados online com o mesmo nível de granularidade.

A causa dessa diferença entre SEMA e o Circuito de Medição pode estar relacionada ao circuito de *Fuel Gauge* da bateria, uma vez que a SEMA utiliza os dados extraídos desse recurso para estimar o consumo de todo o *hardware* do sistema, e que é alimentado pela bateria. Trabalhos futuros poderão aplicar as métricas utilizadas nessa abordagem na comparação entre diferentes dispositivos a fim de verificar essa diferença.

Nos experimentos com os algoritmos de ordenação, o caso do algoritmo CountingSort é o único onde as diferenças apresentadas entre os métodos de medição (SEMA, PowerTutor e Arduino) não seguem o padrão dos demais algoritmos. Essa variação também está relacionada ao tempo de execução. Como o tempo de execução do CountingSort é o menor dentre os algoritmos avaliados, nossa abordagem adquire poucas amostras de corrente e tensão, e com isso, sua estimativa fica mais distante do valor medido pela placa Arduino.

Também nos experimentos com os algoritmos de ordenação ocorreu aproximação entre os valores estimados pela SEMA e pelo valor medido pelo circuito de medição com Arduino, haja vista a diferença entre esses métodos ser de 28% no algoritmo BubleSort, e representa a menor diferença encontrada em todos experimentos.

Nos algoritmos de ordenação levantou-se a hipótese de uma aproximação da estimativa da SEMA ao valor real conforme maior o tempo de execução, porém isso não foi verificado nos testes com o MP3 Player. Além disso, nos experimentos com algoritmos de ordenação, no BubbleSort, por exemplo, um conjunto pequeno de instruções é executado pela CPU repetidas vezes, já que o procedimento é iterativo e simples. Consequentemente, a potência consumida pela CPU neste caso é mais constante e pode ser melhor capturada pelas amostragens realizadas através do driver da bateria, se comparada ao MP3 Player, que é um algoritmo mais complexo.

Assim, embora haja também repetição de instruções no MP3 Player, há maior variação devido a complexidade da decodificação e devido a variação na carga que também impactará nos chaveamentos dos circuitos e consequentemente no consumo.

Observa-se também uma grande diferença quando os resultados obtidos pelo SEMA e pela medição com o circuito de medição com Arduino são comparados ao consumo energético adquirido através do PowerTutor. Os valores estimados pelo PowerTutor estão sempre abaixo dos outros dois métodos, ou seja, subestimados, e essa diferença ocorre, pois o PowerTutor utiliza modelos para estimar o consumo energético. Além disso, a versão da ferramenta disponível possui modelo apenas dos *smartphones* HTC G1 e G2, e Nexus One. Assim, ao ser utilizada em outros dispositivos, o consumo de alguns componentes não é avaliado pois estes não são compatíveis com o modelo original. Assim, como já era esperado, o consumo estimado pelo PowerTutor é menor do que os valores obtidos pelos outros dois métodos, indicando sua baixa precisão em todos os experimentos realizados.

As limitações dos modelos empregados pelo PowerTutor serviram de fator motivador para o desenvolvimento da abordagem apresentada nesse trabalho, visto haver dúvida com relação a precisão dessa ferramenta e sua flexibilidade. No entanto, vale ressaltar que ainda que os testes comparativos tenham mostrado a imprecisão do PowerTutor, uma mesma tendência é observada nos valores obtidos pela ferramenta, quando comparada aos valores medidos pelos outros métodos. Desta forma, dependendo da análise que se deseja realizar ferramentas menos precisas podem ser empregadas. Nossa abordagem quando comparada ao PowerTutor, obteve valores mais próximos dos valores medidos pelo circuito de medição com Arduino e provê facilidades de uso muito similares as do PowerTutor, apresentando porém maior flexibilidade, uma vez que não depende de modelos específicos a uma plataforma de hardware.

Além disso, experimentos com o osciloscópio demonstraram que a medição com o Arduino é bastante próxima do valor real consumido pelo sistema, podendo ser empregada para avaliar a precisão dos métodos de estimativa.

6 CONCLUSÕES E TRABALHOS FUTUROS

A lacuna existente entre a capacidade das baterias e a grande quantidade de recursos agregados aos dispositivos móveis trouxe a necessidade de avaliar e gerenciar os recursos energéticos da melhor maneira possível. Para isso, ferramentas têm sido desenvolvidas para auxiliar os projetistas na identificação dos componentes mais críticos quanto ao consumo de energia, fornecendo perfis de consumo energético que podem ser utilizados como ponto de partida para otimizações tanto no SO quanto nos aplicativos.

A estimativa de energia e análise de perfis energéticos tem sido uma preocupação dos engenheiros e pesquisadores na área de dispositivos móveis, devido a estes serem alimentados por baterias. Este problema pode ser atacado em diferentes níveis de abstração (desde os níveis mais baixos de *hardware* até o nível de *software*).

Este trabalho revisa as principais técnicas empregadas na avaliação de consumo energético, incluindo técnicas baseadas em modelos e baseadas em medição. Além disso, este trabalho apresenta abordagens e ferramentas propostas para avaliar consumo no nível de *software*. Dentre os trabalhos relacionados, foram destacados aqueles com foco principal na avaliação de consumo de energia em dispositivos móveis.

Com base neste estudo, este trabalho propõe uma abordagem para avaliar consumo energético de aplicativos móveis baseada em amostras obtidas através do driver do *Fuel Gauge* da bateria do dispositivo. Essa abordagem obtém granularidade em nível de função/método e tem por objetivo auxiliar principalmente desenvolvedores. Além disso, a abordagem proposta não necessita de alterações no SO do dispositivo para ser utilizada, bem como pode ser utilizada em diferentes dispositivos. Para demonstrar a abordagem, foi implementada uma ferramenta que realiza a estimativa de consumo para aplicativos Android, a qual foi denominada SEMA.

A precisão da SEMA está limitada pela taxa de atualização do driver da bateria do dispositivo. Os experimentos realizados apontam que os resultados obtidos através dessa abordagem diferem em média 40,25% da medição realizada com o Arduino, que é a mais precisa e usada como base. Ainda que esta diferença seja

consideravelmente alta, percebe-se que ela se mantém na maioria dos casos, indicando que nossa abordagem pode ser adotada como uma estimativa para orientar desenvolvedores em otimizações no código.

O ponto fraco da abordagem proposta é a baixa quantidade de amostras coletadas quando o tempo de execução do aplicativo é muito curto, o que foi verificado na execução do algoritmo CountingSort. Contudo, para aplicativos com maior tempo de execução, nossa abordagem se aproxima mais do valor real medido. Nos testes realizados algoritmos com tempo de execução a partir de 4 segundos passaram a apresentar precisão próxima a média de 40,25%.

Como trabalhos futuros, novos cenários de testes podem ser definidos com intuito de avaliar o consumo energético de outros algoritmos ou funções dentro de aplicativos. Ainda, a ferramenta pode ser aplicada na detecção de aplicativos que apresentam elevado consumo energético, drenando a bateria do dispositivo. É também prevista a avaliação da abordagem proposta em outros modelos de dispositivos, verificando a precisão de outros modelos de *Fuel Gauge* utilizados por diferentes fabricantes.

Além disso, a implementação da abordagem será revisada, visando adicionar um fator de correção às medições, a fim de aproximá-las ao valor real e permitindo assim reduzir o erro da estimativa. Ainda, a ferramenta será adaptada para uso em processadores com frequência dinâmica, de modo a evitar erro no cálculo do tempo de execução nesses dispositivos.

REFERÊNCIAS

ANDROID. Activities. **Android Developers**, 2014a. Disponivel em: <http://developer.android.com/guide/components/activities.html>. Acesso em: 20 nov. 2014.

ANDROID. Activity. **Android Developers**, 2014b. Disponivel em: <http://developer.android.com/reference/android/app/Activity.html>. Acesso em: 20 nov. 2014.

ANDROID. Android Developers. **Android Developers**, 2014c. Disponivel em: <https://source.android.com/>. Acesso em: 05 dez. 2014.

ANDROID. Android Interfaces. **Android Developers**, 2014d. Disponivel em: <https://source.android.com/devices/index.html>. Acesso em: 07 dez. 2014.

ANDROID. Application Fundamentals. **Android Developers**, 2014e. Disponivel em: <https://developer.android.com/guide/components/fundamentals.html>. Acesso em: 06 dez. 2014.

ANDROID. BatteryManager. **Android Developers**, 2014f. Disponivel em: <http://developer.android.com/reference/android/os/BatteryManager.html>. Acesso em: 14 dez. 2014.

ANDROID. BroadcastReceiver. **Android Developers**, 2014g. Disponivel em: <http://developer.android.com/reference/android/content/BroadcastReceiver.html>. Acesso em: 05 dez. 2014.

ANDROID. Content Providers. **Android Developers**, 2014h. Disponivel em: <http://developer.android.com/guide/topics/providers/content-providers.html>. Acesso em: 05 dez. 2014.

ANDROID. HAL Interfaces. **Android Developers**, 2014i. Disponivel em: <https://source.android.com/devices/sensors/hal-interface.html>. Acesso em: 10 dez. 2014.

ANDROID. Introducing ART. **Android Developers**, 2014j. Disponivel em: <https://source.android.com/devices/tech/dalvik/art.html>. Acesso em: 20 dez. 2014.

ANDROID. Perfomance Tips. **Android Developers**, 2014k. Disponivel em: <http://developer.android.com/training/articles/perf-tips.html>. Acesso em: 21 nov. 2014.

ANDROID. Power Profiles for Android. **Android Developers**, 2014l. Disponivel em: <https://source.android.com/devices/tech/power.html>. Acesso em: 19 dez. 2014.

ANDROID. Profiling with Traceview and dmtracedump. **Android Developers**, 2014m. Disponivel em: <http://developer.android.com/tools/debugging/debugging-tracing.html>. Acesso em: 16 abr. 2015.

ANDROID. Services. **Android Developers**, 2014m. Disponivel em: <http://developer.android.com/guide/components/services.html>. Acesso em: 21 dez. 2014.

ANDROID. The Android Source Code. **Android Developers**, 2014n. Disponivel em: <https://source.android.com/source/index.html>. Acesso em: 16 dez. 2014.

BROOKS, D.; TIWARI, V.; MARTONOSI, M. **Wattch:** a framework for architectural-level power analysis and optimizations. Anais do 27th International Symposium on Computer Architecture. Vancouver, BC, Canada: IEEE. 2000. p. 83-94.

BROWERS, N.; ZUNIGA, M.; LANGENDOEN, K. **NEAT:** A Novel Energy Analysis Toolkit. Anais da 12th ACM Conference on Embedded Network Sensor Systems (Sensys). Memphis, TN, USA: ACM. 2014. p. 16-30.

CHUNG, Y.-F.; LIN, C.-Y.; KING, C.-T. **ANEPROF:** Energy Profiling for Android Java Virtual Machine and Applications. Anais da 17th International Conference on Parallel and Distributed Systems (ICPADS). Tainan: IEEE. 2011. p. 372-379.

CISCO. Cisco Visual Networking Index: Global Mobile, 03 fev. 2015. Disponivel em: <http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white_paper_c11-520862.pdf>. Acesso em: 09 mar. 2015.

CUBIEBOARD. CubieBoard | A series of open source hardware. **Cubieboard**, 20 Janeiro 2015. Disponivel em: <http://cubieboard.org/>. Acesso em: 12 jan. 2015.

DONG, M.; ZHONG, L. **Self-constructive high-rate system energy modeling for battery-powered mobile systems**. Anais da 9th International Conference on Mobile Systems, Applications, and Services (MobiSys). Bethesda, MD, USA: ACM. 2011. p. 335-348.

FLINN, J.; SATYANARAYANAN, M. **PowerScope:** A Tool for Profiling the Energy Usage of Mobile Applications. Anais do 2º IEEE Workshop on Mobile Computer Systems and Applications (WMCSA). Washington, DC, USA: IEEE. 1999. p. 2.

GOOGLE. Android, 2014. Disponivel em: <http://www.android.com/>. Acesso em: 14 nov. 2014.

GOOGLE. Google Play, 2014. Disponivel em: <https://play.google.com/store>. Acesso em: 22 nov. 2014.

GRAPHPAD. GraphPad QuickCalcs: t test calculator. **GraphPad QuickCalcs**, 20 Janeiro 2015. Disponivel em: <http://www.graphpad.com/quickcalcs/ttest1.cfm>. Acesso em: 07 fev. 2015.

IDC. Smartphone OS Market Share, Q3 2014. **IDC:** The premier global market intelligence firm, 2014. Disponivel em: <http://www.idc.com/prodserv/smartphone-os-market-share.jsp>. Acesso em: 08 dez. 2014.

JUNG, W. et al. **DevScope:** A Nonintrusive and Online Power Analysis Tool for Smartphone Hardware Components. Anais do 8º International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS). New York, NY, USA: ACM. 2012. p. 353-362.

KEATING, M. et al. **Low Power Methology Manual for System on Chip**. 1ª. ed. Palo Alto, CA, USA: Springer, v. I, 2007.

MITROFF, S. The Android era: From G1 to Lollipop. **CNET**, 8 Maio 2014. Disponivel em: <http://www.cnet.com/news/history-of-android/>. Acesso em: 28 nov. 2014.

NEWMAN, J. Peak Battery: Why Smartphone Battery Life Still Stinks, and Will for Years. **TIME**, p. 1, 2013. Disponivel em: <http://techland.time.com/2013/04/01/peak-battery-why-smartphone-battery-life-still-stinks-and-will-for-years/>. Acesso em: 7 fev. 2015.

ORACLE. **Java.com**, 2014. Disponivel em: <http://www.java.com/pt_BR/>. Acesso em: 29 nov. 2014.

PATHAK, A.; HU, Y. C.; ZHANG, M. **Where is the energy spent inside my app?:** fine grained energy accounting on smartphones with Eprof. Anais da 7th ACM European Conference on Computer Systems (EuroSys). Bern, Suíça: ACM. 2012. p. 29-42.

SARTOR, L. A.; BECK, A. C. S.; CORRÊA, U. B. **AndroProf:** A Profiling Tool for the Android Platform. III Brazilian Symposium on Computing Systems Engineering (SBESC). Niterói: IEEE. 2013. p. 23-28.

SHEARER, F. **Power Management in Mobile Devices**. 1ª. ed. Burlington, MA, USA: Newnes - Elsevier, v. I, 2007. 336 p.

SIMPLESCALAR. SimpleScalar LLC. **SimpleScalar LLC to serve and project**, 2011. Disponivel em: <http://http://www.simplescalar.com/>. Acesso em: 07 Fevereiro 2012.

SNOWDON, D. C.; PETTERS, S. M.; HEISER, G. **Power Measurement as the Basis for Power Management**. Anais do 2005 WS Operat. System Platforms for Embedded Real-Time applications. Melbourne, Australia: NICTA. 2005. p. 10.

TONINI, A. et al. **Analysis and evaluation of the Android best practices impact on the efficiency of mobile applications**. Anais do Brazilian Symposium on Computing III Systems Engineering (SBESC). Niterói, RJ: IEEE. 2013. p. 2.

TSAO, S.-L.; CHEN, J. J. SEProf: A high-level software energy profiling tool for an embedded processor enabling power management functions. **The Journal of Systems and Software** , v. 85, n. Agosto, 2012, p. 1575-1769, 2012.

VIEIRA, A. et al. **Performance and Energy Consumption Analysis of Embedded Applications based on Android Platform**. Anais do II Brazilian Symposium on Computing III Systems Engineering (SBESC). Natal, RN: IEEE. 2012. p. 59-64.

VOGEL, L. Android Services - Tutorial. **Vogella - Eclipse, Android and Java training and support**, 29 dez. 2013. Disponivel em: <http://www.vogella.com/tutorials/AndroidServices/article.html>.

WILKE, C.; GOTZ, S.; RICHLY, S. **JouleUnit - A Generic Framework for Software Energy Profiling and Testing**. Anais do Workshop on Green In/By Software Engineering (GIBSE). Fukuoka, Japão: ACM. 2013. p. 9-13.

XIAN, C.; CAI, L.; LU, Y.-H. Power Measurement of Software Programs on Computers With Multiple I/O Components. **IEEE T. Instrumentation and Measurement**, v. 56, p. 2079-2086, out. 2007.

YOON, C. et al. **AppScope:** Application Energy Metering Framework for Android Smartphones using Kernel Activity Monitoring. Anais do USENIX Annual Technical Conference (2012). Boston, MA, USA: Usenix Associtation. 2012. p. 387-400.

ZHANG, L. et al. **Accurate online power estimation and automatic battery behavior based power model generation for smartphones**. Anais da International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS). Scottsdale, AZ: IEEE/ACM/IFIP. 2010. p. 105-114.

ZHANG, L. et al. PowerTutor, 2011. Disponivel em: <http://powertutor.org>. Acesso em: 17 Janeiro 2012.

APÊNDICES

APÊNDICE A – Resultados Preliminares

Este apêndice tem por objetivo apresentar os resultados obtidos nos primeiros trabalhos realizados pelo grupo de pesquisa na avaliação do desempenho e consumo energético em dispositivos móveis.

A.1 Introdução

Esses experimentos avaliam o impacto da aplicação de boas práticas de programação quanto ao consumo energético em diferentes dispositivos. As boas práticas de programação propostas por Google em (ANDROID, 2014k) foram aplicadas na implementação de um aplicativo de agenda. Os resultados apresentados comparam o uso ou não da boa prática.

Conforme (TONINI *et al.*, 2013), dentre as sete boas práticas, duas delas se destacam com significante impacto sobre o desempenho e a eficiência energética, que são: O uso da sintaxe apropriada do laço for, e evitar o uso de funções de GET e SET.

Exemplos de aplicação da boa prática que sugere evitar o uso de funções de GET e SET podem ser vistos nas figuras 40 e 41. Essa prática é recomendada, visto que a invocação do método é uma ação onerosa, e quando evitada pode tornar o acesso ao atributo até sete vezes mais rápida em dispositivos com compilação JIT (Just-In-Time) (ANDROID, 2014k)

```
public void insertContact(String name, int telephone){
    Contact contact = new Contact ();
    contact.setName(name);
    contact.setTelephone(telephone);
    contacts.add(contact);
}
```

Figura 40: Método para inserção de contatos - Código original

```
public void insertContact(String name, int telephone){
    Contact contact = new Contact ();
    contact.name=name;
    contact.telephone=telephone;
    contacts.add(contact);
}
```

Figura 41: Método de inserção após aplicação da boa prática

A sintaxe atual do Java permite iterar um *array* de três formas diferentes, conforme ilustrado na figura 42. A função zero() representa a forma tradicional de implementação do laço, onde o tamanho do *array* é lido a cada repetição. A função one() apresenta uma implementação onde o tamanho do *array* é obtido antes da iteração do *array*, e salva em um arquivo, evitando assim o acesso a função que retorna o tamanho do *array* a cada interação. Na função two() está representada a sintaxe Foreach, utilizada para iterar coleções que implementam a interface Iterable e também *arrays* normais. Como essa sintaxe apresenta uma facilidade, visto que diminui a complexidade, evitando a manipulação de index do *array*, seu uso é recomendado, exceto em iterações críticas de *arrays* do tipo ArrayList.

```
static class Foo {
    int mSplat;
}

Foo[] mArray = ...

public void zero() {
    int sum = 0;
    for (int i = 0; i < mArray.length; ++i) {
        sum += mArray[i].mSplat;
    }
}

public void one() {
    int sum = 0;
    Foo[] localArray = mArray;
    int len = localArray.length;

    for (int i = 0; i < len; ++i) {
        sum += localArray[i].mSplat;
    }
}

public void two() {
    int sum = 0;
    for (Foo a : mArray) {
        sum += a.mSplat;
    }
}
```

Figura 42: Diferentes sintaxes do laço for

A.2 Metodologia

Para realizar o estudo de caso, foi desenvolvido um aplicativo Agenda, onde são inseridos quatrocentos contatos. Após, esses contatos são pesquisados na Agenda. Para realizar o rastreamento e obter o tempo de execução como resultado de desempenho, são utilizados os métodos *Debug.startMethodTracing()* e

Debug.stopMethodTracing() da biblioteca *android.os.debug* e pelo DDMS (*Dalvik Debug Monitor System*) (GOOGLE, 2014). Para a visualização do arquivo de rastreamento é utilizado a ferramenta Traceview (GOOGLE, 2014). Entre os dados obtidos do rastreamento estão o Incl CPU Time, Excl CPU Time, número de Chamadas do método, métodos pais e filhos. Dentre essas informações, em nosso trabalho é utilizado o Incl CPU Time, que indica o tempo de CPU e é representado em *ms*. O Incl CPU Time também é importante por ser o tempo de execução do método somando com o tempo de execução dos métodos filhos, diferente do Excl CPU Time que não considera a chamada a métodos filhos em seu resultado.

A análise de consumo energético é realizada com o aplicativo PowerTutor, que faz uma análise de consumo dos aplicativos que estão sendo executados no dispositivo e os resultados são calculados em Joule. Para realizar os experimentos foram utilizados dois dispositivos (*smartphone* modelo GT-5512b com Android 2.3.3, *tablet* Asus Transformer com Android 4.0.3 e um tablet Samsung Galaxy Note 8.0, modelo GT-N5110). O aplicativo Agenda foi executado trinta vezes em cada dispositivo com o código original, após foram aplicadas as boas práticas e realizadas mais trinta vezes. Por fim é calculada a média das trinta execuções e estas são comparadas usando o teste estatístico “*t de student*”.

A.3 Resultados

Essas boas práticas foram avaliadas quanto ao seu real ganho em desempenho e consumo energético. Para o caso do uso de funções de GET e SET, os resultados apresentados nas figuras 43 e 44 indicam que a aplicação da boa prática tem resultados positivos tanto em desempenho quanto no consumo energético. É observada uma redução de 24% a 30% em desempenho, e 24% a 27% em consumo energético, de acordo com o dispositivo. Todas as médias foram comparadas, e o método *t student* indicou que elas são estatisticamente diferenças, ou seja, o uso da boa prática provê ganho na eficiência para os três dispositivos utilizados.

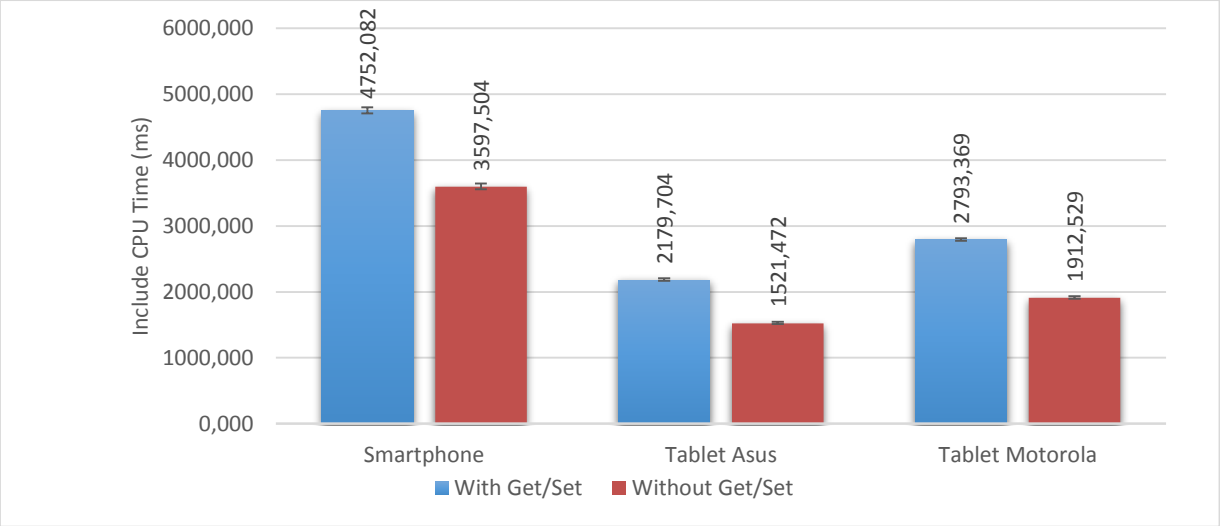

Figura 43: Comparativo de desempenho - Funções de GET/SET

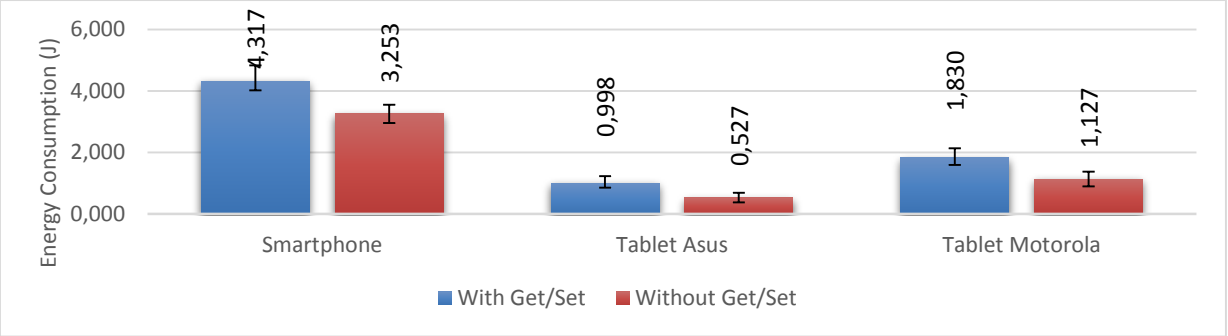

Figura 44: Comparativo de consumo energético - Funções de GET/SET

Para o caso da sintaxe do laço for, os resultados apresentados nas figuras 45 e 46, indicam que a sintaxe onde o tamanho do *array* é armazenado em uma variável local antes do início da execução do laço apresenta melhores resultados de desempenho e consumo energético, enquanto o For Each possui os piores resultados. A diferença apresentada é de uma melhoria de 33% a 38% em desempenho, e de 36% a 52% em energia quando comparamos o For com tamanho ao Foreach. A diferença em relação aos resultados esperados se dá pela utilização do ArrayList, conforme ressaltado por Google.

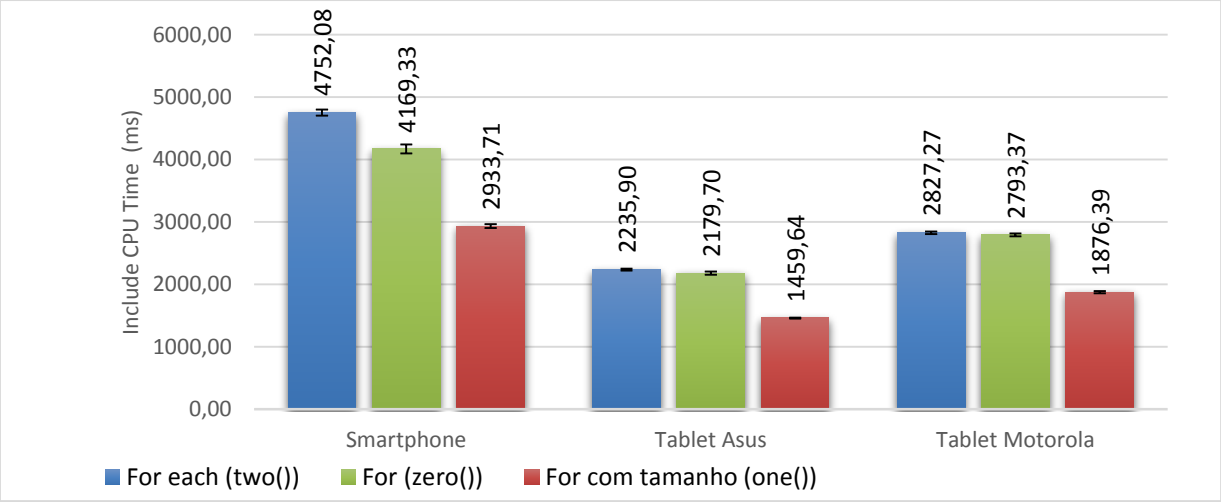

Figura 45: Comparativo de desempenho - Sintaxe do laço for

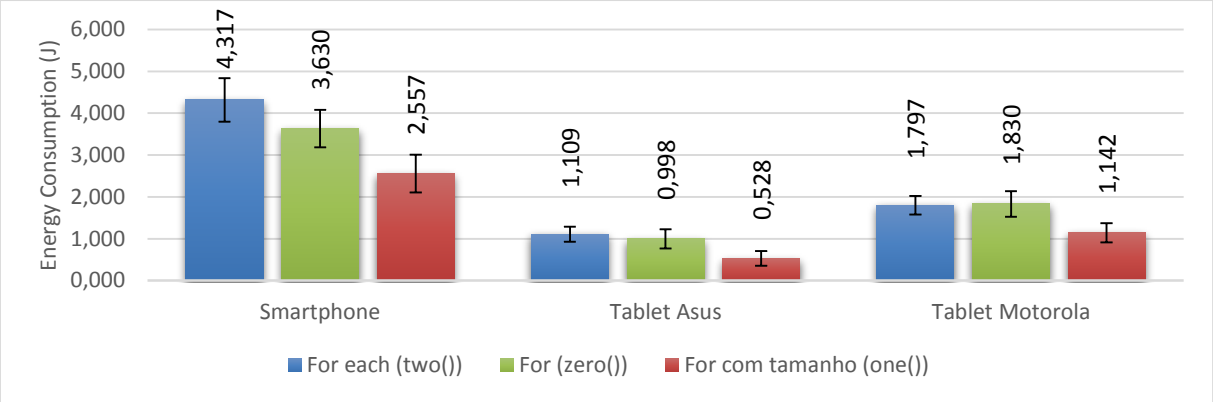

Figura 46: Comparativo de consumo energético - Sintaxe do laço for

Outro experimento foi realizado, alterando o ArrayList por um *array* puro, e novos testes foram executados variando a sintaxe do laço for. Os resultados para esse experimento são demonstrados nas figuras 47 e 48. Inicialmente é possível verificar um ganho de 47% a 68% no tempo de execução e de 58% a 76% comparando o *array* ao ArrayList. Quando comparadas as diferentes sintaxes do for, o Foreach é 23% melhor que for original e similar ao for com tamanho, quando tratamos de consumo energético. Os resultados de performance mostram que o Foreach é 14% mais rápido que o for original e 3% mais rápido que o for com tamanho.

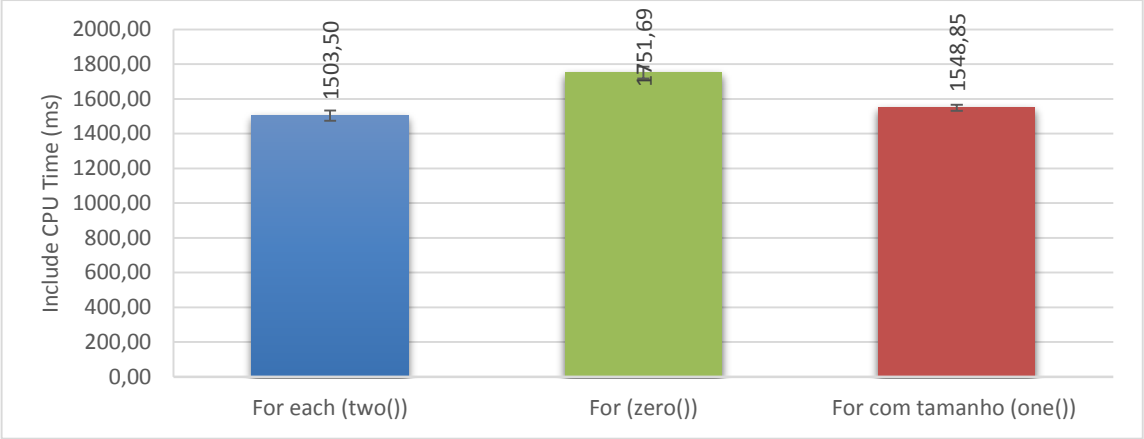

Figura 47: Desempenho das diferentes sintaxes do for utilizando *array* puro

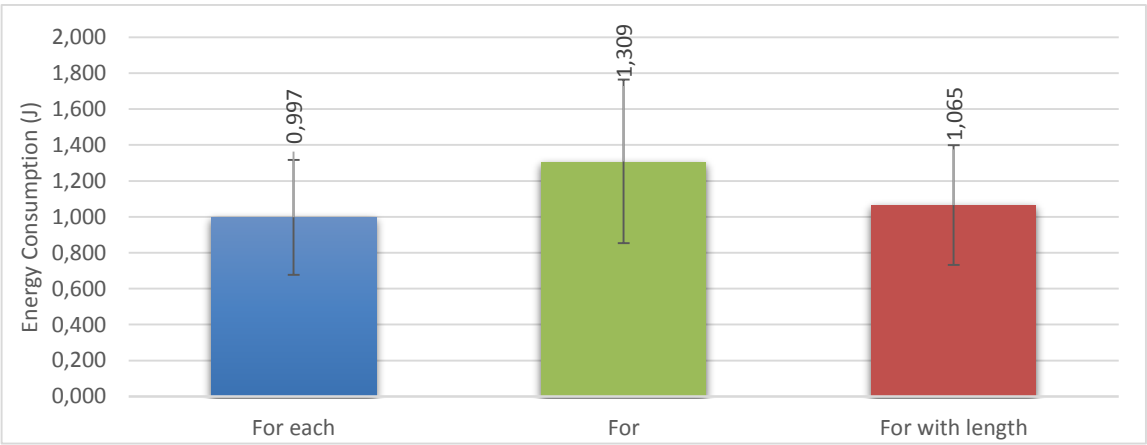

Figura 48: Consumo energético das diferentes sintaxes do for utilizando *array* puro

Através desses testes é possível observar uma tendência entre desempenho e consumo energético, visto que em todos os casos onde ocorreu ganho de desempenho, houve também melhoria no consumo energético. Observa-se ainda, que o uso de boas práticas de programação traz significativas melhorias à eficiência do *software*.

APÊNDICE B – Tabelas de dados dos experimentos

Este apêndice tem por objetivo apresentar as tabelas com os dados extraídos dos experimentos realizados.

Tabela 7: Dados dos experimentos para o algoritmo BubbleSort

Tempo de Execução (s)	**Potência (W) SEMA**	**Potência (W) Circuito de Medição**	**Energia (J) SEMA**	**Energia (J) Circuito de Medição**	**Energia (J) PowerTutor**
16,20099643	1,0170053049	1,54796029	16,47649931	25,07849912	8,9
16,21400338	1,0527302182	1,536234234	17,06897132	24,90850706	9,1
17,14347455	1,0295434023	1,53444186	17,64995111	26,30566498	8,2
17,00408018	0,7958626744	1,531526238	13,53291272	26,04219494	9
17,12068593	2,2380010575	1,541993816	38,31611321	26,39999183	8,7
17,0804853	1,1238278844	1,548105125	19,19552566	26,44238682	9
17,14172418	0,8120502069	1,53632631	13,91994066	26,33528185	9
16,85498372	1,0518148480	1,557279721	17,72832213	26,24792434	9,2
17,96176213	0,9205521703	1,510363226	16,53473911	27,128785	8,7
17,01086822	1,1960340000	1,536608984	20,34557676	26,13905293	9
16,77614947	0,6497235000	1,51229556	10,89985855	25,37049635	9,1
16,94364947	1,0530053953	1,512785001	17,84175431	25,63209877	9,1
16,91277368	0,9220952398	1,499834654	15,5951881	25,36636406	8,7
17,09064459	0,8810853295	1,500270416	15,05831622	25,64058847	9
17,18890534	1,0560443793	1,497311187	18,15224687	25,73714027	9,2
17,09257755	0,9251710983	1,495500059	15,81355874	25,56195073	9
17,02182263	0,7087965087	1,48975594	12,06500845	25,35836137	8,7
16,9343473	0,8064087674	1,503855156	13,65600613	25,46680551	8,7
17,12361347	0,7461871908	1,497431119	12,77742103	25,64143167	8,8
16,64926526	1,3077920119	1,502810022	21,77377611	25,02068269	9
17,50057588	1,7474402938	1,506957891	30,58121146	26,37263093	8,9
17,10036893	1,3486933256	1,49944901	23,06315343	25,64113126	8,3
16,70735176	1,4252858631	1,48885678	23,81275227	24,87485394	9
16,89112088	1,0836752941	1,498939509	18,30449039	25,31876844	9
16,2844583	1,3720629212	1,486297737	22,34330142	24,20355352	9,2
17,11040838	1,1371216127	1,484447234	19,45661517	25,3994984	9,1
16,46325392	1,0825417212	1,488414958	17,82215924	24,5041534	9,1
16,38985893	1,1646034909	1,484914006	19,08768692	24,33753107	9,2
17,02172273	0,8064087674	1,491546214	13,72646645	25,3886861	8,7
16,23534631	1,1238278844	1,481462201	18,2457349	24,05205188	9

Tabela 8: Dados dos experimentos para o algoritmo CountingSort

Tempo de Execução (s)	Potência (W) SEMA	Potência (W) Circuito de Medição	Energia (J) SEMA	Energia (J) Circuito de Medição	Energia (J) PowerTutor
0,553338585	1,029184	1,662492493	0,569487218	0,919921244	0,665
0,572474042	1,029184	1,621370297	0,589181124	0,928192408	0,601
0,595100999	1,029184	1,589461328	0,612468427	0,945890024	0,734
0,539764583	1,029184	1,570063395	0,555517073	0,847464614	0,576
0,69731175	0,81642	1,639520179	0,569299259	1,143256685	0,687
0,54440125	0,81642	1,634993611	0,444460069	0,890092565	0,707
0,539238792	0,81642	1,611655262	0,440245335	0,869067036	0,652
0,722864708	0,81642	1,651772185	0,590161205	1,194007818	0,652
0,533958917	0,81642	1,644798465	0,435934739	0,878254807	0,714
0,710341793	0,81642	1,592532012	0,579937247	1,131242045	0,594
0,513197084	1,110816	1,738679068	0,570067532	0,892285028	0,576
0,621954375	1,110816	1,659665124	0,690876871	1,032235985	0,665
0,522056875	1,110816	1,607542237	0,57990913	0,839228477	0,643
0,548365958	1,110816	1,545065997	0,60913368	0,847261595	0,621
0,667462375	0,83742	1,655775206	0,558946342	1,105167652	0,597
0,609151125	0,83742	1,824890063	0,510115335	1,111633835	0,741
0,558877709	0,83742	1,684282705	0,468015371	0,941308059	0,601
0,6210305	0,83742	1,730211609	0,520063361	1,07451418	0,576
0,53269175	0,67608	1,613796213	0,360142238	0,859655929	0,714
0,595614041	0,67608	1,629619538	0,402682741	0,970624278	0,676
0,537270751	0,67608	1,787499748	0,363238009	0,960371332	0,583
0,693011916	0,67608	1,702678534	0,468531496	1,179976513	0,661
0,539742292	0,560672	1,672848377	0,30261839	0,902907017	0,687
0,535499333	0,560672	1,898637379	0,300239482	1,01671905	0,541
0,603952333	0,560672	1,61183296	0,338619162	0,973470276	0,702
0,551759958	0,560672	1,642569368	0,309356359	0,906304006	0,669
0,737743584	0,560672	1,80066662	0,413632171	1,328430246	0,759
0,566440083	0,425935	1,55027744	0,241266657	0,878139282	0,601
0,564010375	0,875265	1,913820957	0,493658541	1,079414876	0,643
0,693011916	0,875265	1,608520949	0,606569075	1,114724185	0,643

Tabela 9: Dados dos experimentos para o algoritmo HeapSort

Tempo de Execução (s)	Potência (W) SEMA	Potência (W) Circuito de Medição	Energia (J) SEMA	Energia (J) Circuito de Medição	Energia (J) PowerTutor
3,724024669	0,914825	1,526539377	3,406830868	5,684870297	2,2
3,673876544	0,934990	1,513635483	3,43503783	5,560909897	2,1
3,658176002	0,934990	1,529203123	3,42035798	5,594094165	2
3,740083627	0,979506	1,526181431	3,663434353	5,708046183	2,1
3,68477921	0,979506	1,506302559	3,609263345	5,550392354	2,3
3,640068544	0,941630	1,518800882	3,427597743	5,528539314	2
3,651232127	0,941630	1,508492186	3,438109708	5,507855133	2
3,676311127	0,911310	1,554740178	3,350259093	5,715708617	2,1
3,654297752	0,911310	1,515582551	3,330198084	5,53838991	2
3,730540127	0,944700	1,518176069	3,524241258	5,663616744	2,2
3,703745169	0,944700	1,517864921	3,498928061	5,621784868	2,2
3,641865834	0,939550	1,522379799	3,421715044	5,544302977	1,9
3,518978792	0,939550	1,53041568	3,306256524	5,385500322	2
3,644746419	0,945060	1,560780262	3,444504051	5,688648269	2,1
3,546742293	0,945060	1,547565026	3,351884271	5,488814328	2
3,507629544	1,028789	1,502010198	3,608610691	5,268495347	2,3
3,567215877	1,028789	1,505635284	3,669912455	5,370926092	1,9
3,699082752	0,960688	1,519189616	3,553664411	5,619608107	2,4
3,663116001	0,960688	1,535210947	3,519111585	5,623655785	1,9
3,653005501	0,215644	1,524252804	0,787748718	5,568103879	1,9
3,677020127	1,350804	1,513625418	4,966933496	5,565631128	2,1
3,678545252	0,379543	1,542934977	1,396166101	5,675756132	2
3,73590025	0,808088	1,53973816	3,018936161	5,752308175	2,2
3,627635752	0,819376	1,51270753	2,972397672	5,487551918	2,1
3,754634877	0,972270	1,534524147	3,650518852	5,761577881	2,2
3,664426211	0,972270	1,549070127	3,562811672	5,676453177	2,2
3,697802377	1,374154	1,535080235	5,081349928	5,676423342	2,1
3,724283835	1,374154	1,573150642	5,117739529	5,858859505	2
3,702821628	0,914825	1,543542522	3,387433796	5,715462633	2
3,663116001	0,944700	1,549081254	3,460545686	5,674464329	2,1

Tabela 10: Dados dos experimentos para o algoritmo InsertionSort Iterativo

Tempo de Execução (s)	Potência (W) SEMA	Potência (W) Circuito de Medição	Energia (J) SEMA	Energia (J) Circuito de Medição	Energia (J) PowerTutor
3,513213294	0,96681	1,550010972	3,396609745	5,445519152	2,1
3,333690376	0,96681	1,524906364	3,223045192	5,083565671	2
3,284474793	0,5035	1,543575858	1,653733058	5,069835996	1,9
3,519775043	0,5035	1,545886004	1,772206734	5,441170976	2,2
3,504919252	0,94012	1,534615917	3,295044687	5,378704872	2,2
3,369082501	0,94012	1,576917281	3,167341841	5,312764418	2
3,426941044	0,976734	1,540715916	3,347209834	5,279942609	2
3,264418335	0,976734	1,533616159	3,188468378	5,006364707	1,9
3,348254961	0,976734	1,536644697	3,270354461	5,145078229	1,9
3,38572196	0,71596	1,57168731	2,424041494	5,32129624	2,1
3,443750084	0,715963	1,553962665	2,465597641	5,351459059	2,2
3,384366793	0,965104	1,570740981	3,266265929	5,315963617	2,1
3,495537835	0,965104	1,595235522	3,373557547	5,576206124	2
3,333686333	0,994347	1,547392123	3,314841004	5,158519973	2,1
3,385940627	0,994347	1,580771799	3,366799905	5,352399458	2,1
3,417397209	0,632267	1,554458502	2,160707481	5,312202145	1,9
3,284955835	0,632267	1,561470112	2,076969171	5,129360356	2,2
3,38093546	0,632267	1,560084654	2,13765392	5,274545528	2,2
3,34544871	0,43215	1,543091556	1,44573566	5,162333656	2
3,560787417	0,43215	1,620290287	1,538794282	5,769509266	1,9
3,431664209	0,96486	1,561413264	3,311075529	5,358246014	1,9
3,466303294	0,96486	1,586392541	3,344497396	5,498917691	2
3,284359127	0,96486	1,650017487	3,168946747	5,419249992	2,1
3,450538168	0,98208	1,553314545	3,388704524	5,359771124	2,2
3,464314586	0,98208	1,603017205	3,402234069	5,553355884	2
3,465400627	0,96252	1,522117938	3,335517412	5,274748455	1,9
3,443838043	0,96252	1,607766456	3,314762993	5,536887286	2,1
3,358389585	0,96252	1,618250697	3,232517143	5,434716286	2,1
3,431821459	0,965496	1,556718652	3,313409891	5,342380476	2,2
3,4770005	0,58097	1,603607412	2,02003298	5,575743772	2,1

Tabela 11: Dados dos experimentos para o algoritmo InsertionSort Recursivo

Tempo de Execução (s)	Potência (W) SEMA	Potência (W) Circuito de Medição	Energia (J) SEMA	Energia (J) Circuito de Medição	Energia (J) PowerTutor
3,924378585	1,298168	1,565952389	5,094502699	6,14539002	2,2
3,745715668	0,523848	1,545779321	1,962185661	5,790049823	2,2
3,845276085	0,546199	1,620658804	2,100285952	6,231880541	2,2
3,916725751	0,227205	1,535457083	0,889899674	6,013964298	2,4
3,97400021	0,227205	1,553097717	0,902912718	6,172010655	2,2
3,870440002	0,968565	1,547466146	3,748772721	5,989374875	2,4
3,881248501	0,968565	1,527821926	3,759241454	5,92985656	2,3
3,984899252	0,972774	1,551906941	3,876406385	6,18419281	2,3
3,78940046	0,972774	1,567814987	3,686230243	5,941078832	2,1
3,852307294	0,935172	1,551038327	3,602569917	5,975076262	2,2
3,704350586	0,935172	1,549078044	3,464204946	5,738328161	2,2
3,842973794	0,981486	1,583811243	3,771824977	6,0865451	2,4
3,925716668	0,981486	1,585856377	3,85303595	6,225622813	2,1
3,932691709	0,981486	1,569812397	3,859881855	6,173588199	2,4
3,88202571	0,94183	1,553460326	3,656208274	6,030572923	2,3
3,945522459	0,94183	1,585072717	3,716011418	6,253940002	2,2
3,847402709	0,973685	1,534507301	3,746158307	5,903867545	2,4
3,875297669	0,973685	1,531745916	3,773319211	5,935971379	2,4
3,921208544	0,94468	1,553922038	3,704287287	6,093252373	2,2
3,77173546	0,94468	1,522964139	3,563083054	5,744217848	2,2
3,718041627	1,412175	1,540546559	5,250525435	5,727816236	2,3
3,784075544	1,412175	1,534153335	5,343776881	5,805352114	2,1
3,943245252	1,412175	1,582889474	5,568552364	6,241721404	2,1
3,959674544	0,682695	1,537913509	2,703250013	6,089636971	2,3
3,927157419	0,682695	1,527520889	2,681050734	5,998814993	2,2
3,86903846	0,993564	1,572592829	3,844137328	6,084422139	2,4
3,875650293	0,993564	1,63867241	3,850706608	6,350921206	2,3
3,882159961	0,964665	1,63867241	3,744983839	6,361588419	2,4
3,748267544	0,964665	1,63867241	3,61582251	6,14218261	2,2
3,818855835	0,95209	1,63867241	3,635894452	6,257853695	2,1

Tabela 12: Dados dos experimentos para o algoritmo MergeSort

Tempo de Execução (s)	Potência (W) SEMA	Potência (W) Circuito de Medição	Energia (J) SEMA	Energia (J) Circuito de Medição	Energia (J) PowerTutor
7,171068129	0,854792	1,61320597	6,129771668	11,56840992	4,5
7,168703837	0,834150	1,618947293	5,979774306	11,60575367	4,6
7,10317071	0,953419	1,600802557	6,772297915	11,37077384	4,5
7,226123336	0,857590	1,607217552	6,197051112	11,61395226	4,7
7,872779544	0,913160	1,628800554	7,189107368	12,82318768	4,6
7,372879212	1,156435	1,656707665	8,526255572	12,2147055	4,5
7,29730796	0,953704	1,623903906	6,959471791	11,8501269	4,5
7,540599003	0,953704	1,625212667	7,191499432	12,25507702	4,7
7,277717835	0,464162	1,614077442	3,378040066	11,74680019	4,5
7,320095961	0,882815	1,643620269	6,462290516	12,03145809	4,7
7,35747317	1,324724	1,63867241	9,746621288	12,05648829	4,7
7,294314544	0,577200	1,619772933	4,210278355	11,81513326	4,6
7,309697712	0,868207	1,609916298	6,346330721	11,76800148	4,6
7,350215877	1,323835	1,584788384	9,730473036	11,64853674	4,7
7,794574169	0,904569	1,591924234	7,050730161	12,40837151	4,5
7,509471796	0,939170	1,616772306	7,052670627	12,14110603	4,5
7,418492253	0,860687	1,621565209	6,384999842	12,02956894	4,4
8,217010379	0,870848	1,607935631	7,155767055	13,21242377	4,7
8,214128837	1,270548	1,586896078	10,43644497	13,03496883	4,5
7,758715169	0,878073	1,601307228	6,812718305	12,42408668	4,5
7,87151017	0,760716	1,601549812	5,98798373	12,60661563	4,6
7,922012796	1,128644	1,604006808	8,94113221	12,70696246	4,7
7,811886587	1,302312	1,602559349	10,17351364	12,51901188	4,4
7,895695044	0,950607	1,612747962	7,505702979	12,73376609	4,6
7,980273879	1,040382	1,662030377	8,302533299	13,2634576	4,5
8,12478642	1,216915	1,667219361	9,887174466	13,54580122	4,6
7,069911419	0,870310	1,591484339	6,153014607	11,2516533	4,7
7,163116335	0,908960	1,621614897	6,510986224	11,61581616	4,5
7,12035442	0,504200	1,610437131	3,590082699	11,46688314	4,5
7,088798879	0,854792	1,59924335	6,059448571	11,33671447	4,6

Tabela 13: Dados da análise de precisão entre Arduino e Osciloscópio

Tensão (V) sobre resistor *shunt*	
Arduíno	**Osciloscópio**
0,024194	0,022342
0,024582	0,022726
0,021476	0,028366
0,027696	0,01849
0,026808	0,022323
0,02346	0,025862
0,023719	0,023219
0,024302	0,022208
0,022944	0,022435
0,023888	0,023155
0,023795	0,02332
0,023702	0,024485
0,023609	0,02365
0,023516	0,023815
0,023423	0,02398
0,02333	0,024145
0,023237	0,024309
0,023143	0,024474
0,02305	0,024639
0,022957	0,024804
0,022864	0,024969
0,022771	0,025134
0,022678	0,025299
0,022585	0,025464
0,022492	0,022629
0,022399	0,025794
0,022306	0,025959
0,022213	0,026124
0,02212	0,022289
0,022027	0,02521

Tabela 14: Dados dos experimentos com o MP3 Player para áudio de 30s

Tempo de Execução (s)	Potência (W) SEMA	Potência (W) Circuito de Medição	Energia (J) SEMA	Energia (J) Circuito de Medição	Energia (J) PowerTutor
30,4723052	0,549023301	1,867436851	16,73000559	56,90510571	2,9
30,4692333	0,667884961	1,697148147	20,34994266	51,71080276	3
29,7397757	1,349623040	1,389828942	40,13748646	41,33320098	3,2
29,7827454	1,21112784	1,892298279	36,07071209	56,35783785	3,1
30,1719976	1,555195092	1,709822772	46,92334252	51,5887685	3,2
29,9074272	0,595392748	1,613833915	17,80666527	48,26562028	3,2
29,9969219	0,772324874	1,532543808	23,16736892	45,9715969	3
29,8217391	0,591972159	1,607256305	17,65363932	47,93117825	2,8
30,4041679	0,864332646	1,794338126	26,27931488	54,55535762	2,9
30,0188046	1,206804211	1,383833631	36,22681985	41,54103142	2,8
29,7334174	1,870283613	1,773917512	55,60992331	52,74462978	3,1
30,1526921	0,996670566	1,758539472	30,05230069	53,02469925	3,1
29,9890013	0,879468457	1,715344141	26,3743807	51,44145767	3
29,9227681	0,515769669	1,448496935	15,43325619	43,34303787	3
29,9309594	0,85121849	1,765456062	25,47778606	52,8417937	3
30,3316215	0,721970531	1,735664743	21,89853687	52,64552598	3,1
29,7490973	1,42878387	1,417660502	42,50503044	42,17412027	3,2
30,2164948	1,128119016	1,824041698	34,0878024	55,11614649	3,1
29,9748506	2,073666116	1,74315307	62,15783193	52,25075277	3,2
29,8118339	1,560012107	1,594353575	46,50682179	47,53060395	2,9
30,1367742	1,173351204	1,476203572	35,36102032	44,48801375	2,9
30,1125568	0,758664934	2,137116503	22,8453409	64,354042	2,8
30,1421538	0,598917589	1,728641344	18,05266612	52,10497335	3
29,9164555	1,2416526490	1,89466377	37,14584619	56,68162432	3,2
29,8593311	1,71730079734	1,899012404	51,27745317	56,70324022	3,1
30,0711111	1,49331761056	1,74315307	44,90571971	52,41854955	3,2
29,1788998	0,70119515	1,673435894	20,46010302	48,82901827	3
30,5728632	1,1855519	1,535471496	36,24571608	46,94376003	3
29,9249478	0,854440185	2,098205036	25,56907792	62,78867611	2,9
30,396636	0,711358443	1,729795885	21,62290363	52,57997583	3

Tabela 15: Dados dos experimentos com o MP3 Player para áudio de 1 minuto

Tempo de Execução (s)	Potência (W) SEMA	Potência (W) Circuito de Medição	Energia (J) SEMA	Energia (J) Circuito de Medição	Energia (J) PowerTutor
60,12191153	0,9040902682	1,369964694	54,35563512	82,36489615	5,7
60,45551432	0,8854734142	1,33898504	53,53175067	80,94902927	5,6
59,64199511	0,8945140534	1,366503751	53,3506028	81,50101005	5,6
60,59420974	0,8893497381	1,328022432	53,88944456	80,47046979	5,7
59,4940312	1,3192793317	1,726668313	78,48924571	102,7264585	5,8
60,10167103	0,6368933804	1,360377141	38,27835643	81,76093939	5,8
59,94260607	1,1801260266	1,709187939	70,73982952	102,4531793	5
59,89009465	1,0282349450	1,361031277	61,58108818	81,512292	5,5
60,09956832	1,0659260083	1,73442779	64,06169296	104,2383615	5,5
60,49873595	0,9490854719	1,324388511	57,41847136	80,12383084	5,6
59,69980674	1,2990783923	1,732005951	77,55472896	103,4004205	5,6
60,34492249	0,7041678099	1,339731245	42,49295191	80,84597816	5,7
59,57157795	0,6623621940	1,73593984	39,45796107	103,4126755	5,3
60,34408203	0,9575107967	1,736942009	57,78011006	104,8141711	5,4
59,79220799	1,4584902017	1,368235022	87,20634948	81,80979299	5,4
60,35906078	0,8140241653	1,707590648	49,13373407	103,0685677	5
59,97074557	1,2559388754	1,379211867	75,31959075	82,71236395	5,5
60,11865665	0,6114392040	1,736453872	36,75890357	104,3932741	5,6
60,12537665	0,6726331192	1,379832481	40,44231964	82,96294761	5,8
60,07280007	0,7949277226	1,683141528	47,75353415	101,1110245	5,7
60,20311378	1,1474493079	1,335146804	69,08002124	80,37999495	5,6
60,36229457	1,1595814231	1,753171889	69,99499544	105,825478	5,6
60,33225482	0,8451440182	1,347633973	50,98944426	81,30579624	5,8
60,52311465	0,6697286079	1,307523164	40,53406132	79,13537434	5,6
60,33935849	0,7168339554	1,700174138	43,25330101	102,5874168	5,7
60,33137311	0,8615324411	1,672983265	51,97743515	100,9333775	5,7
60,35154861	0,8624461322	1,360749095	52,04995967	82,12331513	5,6
60,26716678	0,7911177070	1,702128217	47,67842279	102,5824451	5,8
60,18545545	1,3088430083	1,3499384	78,77331256	81,24665741	5,4
60,17509745	0,7746138590	1,350067398	46,61246445	81,24043723	5,6

Tabela 16: Dados dos experimentos com o MP3 Player para áudio de 2 minutos

Tempo de Execução (s)	Potência (W) SEMA	Potência (W) Circuito de Medição	Energia (J) SEMA	Energia (J) Circuito de Medição	Energia (J) PowerTutor
120,0624656	1,08367529	1,62187504	130,1087277	194,7263161	11,2
120,2310942	0,97819786	1,301267239	117,6097987	156,452784	12,4
120,4281309	0,84193422	0,941712998	101,3925645	113,4087362	10,8
120,4171221	0,78857067	1,932524534	94,95741025	232,7090427	11,1
120,3785373	0,71490964	1,305490513	86,05977658	157,1530384	11
120,3527076	0,98418922	1,285108128	118,4498374	154,6662428	11,3
120,3852465	0,84718070	1,295502073	101,9880569	155,9593364	10,9
120,524839	0,98647022	1,311204236	118,8941644	158,0326794	10,9
120,3346042	0,90906908	1,286964286	109,3924685	154,8663379	11,1
120,4207803	0,69938795	1,276459136	84,22084231	153,7122051	15,1
120,3383691	0,75999758	1,311284339	91,45686903	157,7978187	11
120,4022857	0,94803045	1,300490493	114,1450327	156,5820278	10,9
120,3865114	0,78421990	1,654412885	94,40949797	199,1689957	10,6
120,4838798	0,66561427	1,296935408	80,19578964	156,2598098	11,1
120,5278567	0,85836895	1,30662037	103,4573701	157,4841527	10,6
120,4715184	0,62311776	1,310963849	75,06794305	157,9338054	10,5
120,4467814	0,95548960	2,246047974	115,0856468	270,5292493	10,4
120,0624656	0,76940341	1,454512642	92,37646993	174,632374	10,6
120,3762295	0,65639328	1,392740027	79,01414816	167,6527931	10,4
121,0971307	0,64452459	1,410475081	78,05007859	172,347561	8,6
120,7081552	0,80368942	1,423217544	97,01186683	170,4865768	10,8
120,7848307	0,73811947	1,412386566	89,15363562	170,5948723	10,7
120,6134641	0,77789217	1,386591223	93,82426961	167,2415707	11
120,4207803	0,75933681	1,422148036	91,43993087	171,2561762	10,7
120,7134087	1,03314299	1,401046981	124,7142115	169,1251568	10,8
120,290557	0,89957885	1,394947515	108,2108412	167,7990136	10,2
120,4799816	0,94387515	1,271385592	113,7180603	153,1765127	10,8
120,7318732	1,01628225	1,482384869	122,6976603	178,971102	10,5
120,520008	0,88962874	1,562610515	107,2180634	188,3258318	10,4
120,6646949	0,77750289	1,550791819	93,81714905	187,1258216	11,1

Tabela 17: Dados dos experimentos com o MP3 Player para áudio de 3 minutos

Tempo de Execução (s)	Potência (W) SEMA	Potência (W) Circuito de Medição	Energia (J) SEMA	Energia (J) Circuito de Medição	Energia (J) PowerTutor
180,3970628	0,733227115	1,288686887	132,272	232,4753	17,2
180,3136134	0,797712261	1,304165341	143,8384	235,1588	16,4
180,4207249	0,665965879	1,290292565	120,154	232,7955	16,6
180,3833165	0,931542275	1,307049553	168,0347	235,7699	16,6
179,8373423	0,694864148	1,302415988	124,9625	234,223	16,8
180,3823441	0,935898924	1,271683407	168,8196	229,3892	16,4
180,2977272	0,725716034	2,309264188	130,845	416,3551	16,5
180,3877565	0,732445603	1,280492974	132,1242	230,9853	17
180,3449028	0,793539913	1,300826633	143,1109	234,5975	17
180,3884884	0,711509863	1,323736533	128,3482	238,7868	17,1
180,4104043	0,684071413	1,30405205	123,4136	235,2646	17,2
180,6264413	0,869845177	1,32029096	157,117	238,4795	16,6
180,2840559	0,791570302	1,302289584	142,7075	234,782	16,7
180,3910736	0,664647526	1,305040427	119,8965	235,4176	16,8
180,4430364	0,851360658	1,31465268	153,6221	237,2199	17,2
180,3715968	0,875479647	1,282699119	157,9117	231,3625	17
180,4611768	0,738896465	1,289930043	133,3421	232,7823	16,4
180,4348772	0,856590159	1,299470012	154,5587	234,4697	17
180,1635616	0,765419892	1,297708866	137,9008	233,7999	16,9
180,206421	0,808116138	1,316929375	145,6277	237,3191	16,5
180,2207903	0,879180453	1,323055782	158,4466	238,4422	16,6
180,3841891	0,744606551	1,309945342	134,3152	236,2934	16,6
186,1462555	0,936374563	1,299053968	174,3026	241,814	16,8
180,7028325	0,687258611	1,312092272	124,1896	237,0988	17
180,3725174	0,739399159	1,312150673	133,3673	236,6759	17
180,3887446	0,75635911	1,328780776	136,4387	239,6971	1,2
180,4555531	0,562767442	1,307449828	101,5545	235,9366	16,8
180,545883	0,862760655	1,328780776	155,7679	239,9059	16,8
180,4459036	0,796219419	1,286575121	143,6745	232,1572	16,6
180,3979095	0,524194874	1,325474417	94,56366	239,1128	16,8

APÊNDICE C – Publicações durante o mestrado

TONINI, A., FISCHER, L., BRISOLARA, L., MATTOS, J.. **Analysis and evaluation of the Android best practices impact on the efficiency of mobile applications**. SBESC. Niterói, RJ: IEEE. 2013. p. 2.

FISCHER, L. BRISOLARA, L., MATTOS, J. **Avaliação de consumo energético em dispositivos móveis – análise das principais técnicas**. In: Encontro de Pós Gradução UFPel, 2013.

TONINI, A. FISCHER, L. BRISOLARA, L. **Sustentatibilidade de software: um estudo de caso da avaliação de aplicativos Android.** Congresso de Iniciação Científica UFPel, 2014.