Universidade Federal de Campina Grande

Centro de Engenharia Elétrica e Informática

Coordenação de Pós-Graduação em Ciência da Computação

Arcabouço de Software Baseado em Componentes para o Desenvolvimento de Aplicações de Gerenciamento de Energia

Diógenes Galdino Gondim

Área de Concentração: Ciência da Computação

Linha de Pesquisa: Engenharia de Software

Hyggo Oliveira de Almeida

(Orientador)

Angelo Perkusich

(Orientador)

Campina Grande, Paraíba, Brasil

Universidade Federal de Campina Grande

Centro de Engenharia Elétrica e Informática

Coordenação de Pós-Graduação em Ciência da Computação

Arcabouço de Software Baseado em Componentes para o Desenvolvimento de Aplicações de Gerenciamento de Energia

Diógenes Galdino Gondim

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Campina Grande - Campus I como parte dos requisitos necessários para obtenção do grau de Mestre em Ciência da Computação.

Área de Concentração: Ciência da Computação

Linha de Pesquisa: Engenharia de Software

Hyggo Oliveira de Almeida

(Orientador)

Angelo Perkusich

(Orientador)

Campina Grande, Paraíba, Brasil

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL DA UFCG

G637a Gondim, Diógenes Galdino.

Arcabouço de software baseado em componentes para o desenvolvimento de aplicações de gerenciamento de energia / Diógenes Galdino Gondim. – Campina Grande, 2014.

56 f. : il. color.

Dissertação (Mestrado em Ciência da Computação) – Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática, 2014.

"Orientação: Prof. Dr. Hyggo Oliveira de Almeida, Prof. Dr. Angelo Perksich".

Referências.

1. Modelagem de Software. 2. Gerenciamento de Energia. 3. Sistemas Operacionais. I. Galdino, Diógenes Gondim. II. Título.

CDU 004.4(043)

"ARCABOUÇO DE SOFTWARE BASEADO EM COMPONENTES PARA O DESENVOLVIMENTO DE APLICAÇÕES DE GERENCIAMENTO DE ENERGIA"

DIÓGENES GALDINO GONDIM

DISSERTAÇÃO APROVADA EM 04/09/2014

HYGGO OLIVEIRA DE ALMEIDA, D.Sc, UFCG
Orientador(a)

ANGELO PERKUSICH, D.Sc, UFCG
Orientador(a)

KYLLER COSTA GORGÔNIO, Dr., UFCG
Examinador(a)

SAULO OLIVEIRA DORNELLAS LUIZ, Dr., UFCG
Examinador(a)

CAMPINA GRANDE - PB

Resumo

A maioria dos sistemas operacionais oferecem suas próprias estratégias de gerenciamento de energia, mas é muito difícil modificar ou ampliar as políticas de energia sem acesso ao código fonte. Várias arquiteturas em gerenciamento dinâmico de energia já foram propostas na literatura, mas elas não são integradas com o sistema operacional subjacente. Neste trabalho, é proposto um arcabouço de software para o desenvolvimento de aplicações gerenciadoras de energia ao nível de usuário, com a flexibilidade arquitetural de se adaptar a diferentes políticas e de se integrar às políticas de gerenciamento do sistema operacional. Para validar o arcabouço, é descrito um estudo de caso mostrando sua viabilidade, demonstrando que a aplicação resultante oferece redução no consumo de energia.

Abstract

Most operating systems implement their own power management techniques, but it is hard to modify or hack their power policies without the source code. Many dynamic power management architectures have been proposed in the literature, but they are not integrated with the underlying OS power manager. In this work, we proposed a software framework for user-level power management, with a flexible architecture to be adapted to different policies and integrated with OS power managers and validated its feasibility with a case study.

Agradecimentos

Agradeço primeiramente aos meus orientadores Hyggo e Angelo. Meus co-orientadores Saulo e Fred que fizeram parte da equipe de pesquisa e foram fundamentais para a conclusão deste trabalho.

Agradeço aos meus amigos Iury, Andryw, Caio e Filipe Lucena que sempre me deram apoio e motivação.

Agradeço aos meus pais (Rejane e Demóstenes) pelo incentivo inicial.

Agradeço à Sonia, que sempre me deu forças pra “moleza” não me pegar!

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Os computadores estão se tornando cada vez mais comuns no cotidiano das pessoas, sendo usados para as mais variadas tarefas, tais como tirar fotos em dispositivos portáteis, nas tarefas do dia a dia no trabalho ou nas centrais de dados das grandes empresas de computação na nuvem. À medida que o uso de computadores tem crescido, também tem crescido a preocupação com a quantidade de energia que eles consomem. No caso dos dispositivos portáteis, o consumo excessivo de energia reduz o tempo de autonomia da bateria; para os computadores *desktop*, um consumo de energia equilibrado é um fator que pode ajudar a reduzir o custo financeiro com energia elétrica. As centrais de dados podem ter seu potencial de escalabilidade limitados aos crescentes custos com energia. Então, nos diversos contextos computacionais, reduzir a energia consumida tornou-se um dos principais focos.

Outro aspecto importante na redução no consumo de energia no desenvolvimento de sistemas computacionais é o aumento na conta dos serviços de energia elétrica e nos custos com infraestrutura elétrica. Além disso, tem-se uma preocupação com as emissões de gases do efeito estufa que são provocados por algumas formas de geração de energia.

O que piora a questão dos custos é que grande parte da infraestrutura de tecnologia das empresas opera numa utilização média de potência baixa, e o consumo de energia de um computador quando ocioso é tipicamente 50-60% da potência total. Se todos os usuários dos computadores desligassem ou desativassem suas máquinas quando não estivessem utilizando-as, sejam elas desktop ou notebooks, haveria uma redução drástica do consumo de energia e, consequentemente, dos custos. Além do mais, ainda seria possível desativar alguns componentes internos dos computadores, tais como a interface de rede sem fio ou brilho da tela. Dessa forma, técnicas de gerenciamento dinâmico de energia são de grande utilidade [BBM00].

Há pelo menos três níveis nos quais a economia de energia pode ser feita. O primeiro nível é de arquitetura de máquina, no qual se tem processadores de múltiplos núcleos como uma forma de economizar energia, e também o aumento dos estados de desempenho dos

dispositivos, conhecidos como *sleep states*. O segundo é o nível de sistema, através de um uso mais eficiente dos recursos de hardware pelo sistema operacional, o que pode incluir técnicas de melhor gerenciamento de processos e *threads* [LBM00], melhor gerenciamento de disco [SMB99] ou até mesmo pela otimização de código com compiladores [LHH02]. O terceiro é o nível de aplicação, que é algo atrativo por várias razões. Primeiramente, esse nível possui maiores informações sobre o custo/benefício entre desempenho de usuário e uso de energia, permitindo que sejam desenvolvidas estratégias mais agressivas de gerenciamento de energia e de forma mais flexível, comparado com os níveis anteriores. Por exemplo, uma aplicação pode reduzir a frequência do processador a um nível aceitável enquanto estiver sendo executada ou mudar a precisão de algum cálculo (mudando de *double* para *float*). Se tais técnicas fossem aplicadas nas camadas anteriores, poderia ocorrer uma negação de serviço, pois elas não possuem informações sobre o desempenho aceitável.

As abordagens no nível de aplicação são, portanto, cientes do contexto dos programas em execução e outras variáveis. Em outras palavras, as aplicações teriam conhecimento dos eventos externos ou internos ao dispositivo, como se o usuário se encontra andando, dentro de um ambiente fechado ou se a bateria se encontra em nível crítico. Ou seja, as aplicações de forma geral são responsáveis por fornecer ao sistema operacional as informações de desempenho e de quais recursos precisam, a fim de que o sistema operacional seja capaz de tomar melhores decisões de gerenciamento. O sistema operacional, por sua vez, seria responsável por fornecer as informações de contexto às aplicações.

1.1 Problemática

A maioria dos sistemas operacionais implementa gerenciamento de energia em algum componente interno. Estão incluídas técnicas como a mudança dinâmica de frequência e tensão do processador (*dynamic voltage and frequency Scaling* – DVFS) e políticas de gerenciamento dinâmico de energia [GLF+00]. Uma política é um algoritmo que busca desativar ou reduzir a potência de um determinado dispositivo quando este está ocioso. Tais políticas podem ser de tempo limite (*timeout*), que se caracteriza por desativar ou diminuir a potência de algum dispositivo após algum período pré-determinado de ociosidade. Uma vez que elas são implementadas em nível de sistema, ou seja, dentro do sistema operacional, há pouca oportunidade de se fazer modificações nessas políticas, o que se dá através apenas de alguns parâmetros, como os valores dos *timeouts*. Ou seja, torna-se necessária alguma solução em gerenciamento de energia que funcione como um complemento ao gerenciamento feito

pelo sistema operacional e que considere as informações de contexto explicadas anteriormente. Essas aplicações gerenciadoras de energia ao nível de usuário são chamadas de *middleware* de gerenciamento de energia, em alguns trabalhos na literatura [XKY09].

Alguns arcabouços de software, modelos e *middlewares* em nível de software para gerenciamento de energia já foram propostos [LSC04], mas nenhum deles faz a integração com o gerenciamento feito pelo sistema operacional. Além do exposto anteriormente, outra motivação deste trabalho é a necessidade das empresas e organizações desenvolverem suas próprias aplicações gerenciadoras de energia, podendo usar os parâmetros de gerenciamento expostos pelo sistema operacional e que seja possível adicionar e remover políticas de gerenciamento de energia.

1.2 Objetivos

Neste trabalho tem-se como objetivo a concepção de um arcabouço de software baseado em componentes para o desenvolvimento de aplicações de gerenciamento de energia. Tais aplicações devem ser capazes de reduzir o consumo de energia em computadores através do uso de políticas de gerenciamento de energia, baseadas no contexto de utilização do computador ou estimativa de consumo de energia.

Uma forma de abstrair o uso pelo usuário final da aplicação gerenciadora de energia proveniente deste arcabouço seria com a ajuda de perfis de energia, que se trata de um agregado de políticas e configurações pré-definidas. Ou seja, o arcabouço proposto também deve ser capaz de reaproveitar as configurações de gerenciamento de energia presentes no sistema operacional da máquina alvo, como por exemplo, os tempos de ociosidade para desligamento de dispositivos e alguns estados padrão de dispositivos.

Essas características são implementadas como serviços providos por componentes de software, os quais interagem entre si para prover funcionalidades ao desenvolvimento das aplicações gerenciadoras de energia, tais como a criação e remoção de políticas, criação e remoção de perfis de energia, que tenha uma baixa sobrecarga de utilização de energia e que facilite o processo de produção por parte da equipe de desenvolvedores. O arcabouço deve vir com algumas definições de políticas prontas, mas principalmente deve prover formas de estendê-lo.

Como forma de validar os serviços desse arcabouço de software, neste trabalho é apresentada a definição e implementação de uma aplicação gerenciadora de energia como estudo de caso, testado em máquinas notebooks, netbooks e desktops. Essa aplicação reúne as

características básicas do arcabouço citadas anteriormente e ainda possui uma interface gráfica para que o usuário final possa escolher entre diferentes perfis de energia, abstraindo os detalhes sobre as políticas subjacentes.

1.3 Relevância

As contribuições deste trabalho incluem os seguintes aspectos:

- A proposta de um arcabouço de software para o desenvolvimento de aplicações gerenciadoras de energia em nível de usuário e mostrando sua eficácia através de um estudo de caso.
- Demonstração de que um gerenciador em nível de usuário pode eficientemente economizar energia, apesar das técnicas de gerenciamento de energia serem acopladas geralmente no nível de sistema.
- Mostrar como um modelo de estimativa de energia pode ajudar na criação de políticas mais eficientes na redução do consumo de energia.
- Contribuiu-se diretamente para o projeto de parceria entre o Laboratório de Sistemas Embarcados e Computação Pervasiva – Embedded, da Universidade Federal de Campina Grande e a Positivo Informática, que teve como objetivo a construção de um gerenciador de energia para dispositivos portáteis.

1.4. Estrutura da Dissertação

O restante deste trabalho está organizado da seguinte forma:

- No Capítulo 2 é apresentada uma visão geral sobre gerenciamento dinâmico de energia, com uma abordagem vertical sobre como ela pode ser aplicada nos diferentes níveis de um sistema computacional. São apresentadas também visões gerais sobre estimativas de potência e um apanhado sobre desenvolvimento de software baseado em componentes.
- No Capítulo 3 é descrita a arquitetura de alto nível do arcabouço de software, e quais os requisitos necessários para o seu desenvolvimento, quais os componentes que compõem o arcabouço, os relacionamentos entre eles, e como estender as funcionalidades do arcabouço.

- No Capítulo 4 é descrito o estudo de caso usado para validar este trabalho. São apresentadas as características da aplicação gerenciadora de energia baseada em componentes, os cenários de testes e os resultados obtidos.
- No Capítulo 5 são apresentados os trabalhos relacionados, com enfoque em trabalhos que apresentam alguma modelagem arquitetural em soluções para gerenciamento de energia, principalmente nos níveis de sistema, aplicação e ambos.
- No Capítulo 6 são apresentadas as conclusões e considerações finais.

Capítulo 2

Fundamentação Teórica

Para melhor entendimento deste trabalho, são descritos os principais conceitos sobre gerenciamento dinâmico de energia, modelos de potência para dispositivos embarcados e software baseado em componentes.

2.1 Gerenciamento Dinâmico de Energia

Várias técnicas de gerenciamento de energia para sistemas computacionais já foram propostas. Essas técnicas são divididas em duas áreas chamadas de técnicas estáticas e técnicas dinâmicas de gerenciamento de energia [BBM00]. As técnicas estáticas são direcionadas ao projeto tanto de hardware quanto de software, como as otimizações feitas no momento da compilação dos programas. Por outro lado, as técnicas dinâmicas usam a informação disponível no momento da execução de um sistema para reduzir o consumo de energia de dispositivos quando eles estão ociosos ou recebendo baixa carga. A redução do consumo de energia em sistemas computacionais via a aplicação de técnicas dinâmicas forma o objetivo principal do arcabouço proposto neste trabalho. Sendo assim, essas técnicas são abordadas com mais detalhes nesta seção.

Gerenciamento Dinâmico de Energia (GDE) trata do desligamento seletivo dos componentes do sistema que estão ociosos ou sutilizados [BBM00]. Essa técnica tem se demonstrado bastante eficaz em reduzir a dissipação de energia nos sistemas computacionais. Porém, aplicar tal técnica também implica em perda de desempenho por causa dos custos de ativação e desativação. Uma política eficaz de GDE deve, portanto, buscar maximizar a economia de energia enquanto mantém a degradação do desempenho em níveis aceitáveis. Nas subseções seguintes discute-se como os dispositivos dão suporte ao gerenciamento de energia, uma visão geral sobre GDE e algumas classes de políticas propostas até então.

2.1.1 Estados de Energia

Para entender melhor como funciona o GDE, é preciso descrever como os dispositivos internos dos sistemas computacionais dão suporte ao gerenciamento. Para o usuário, o sistema

aparentemente se comporta como se estivesse simplesmente ligado ou desligado. Porém, tanto o sistema como um todo quanto os dispositivos possuem múltiplos estados de energia, correspondentes aos estados de energia definidos na especificação *Advanced Configuration and Power Interface* (ACPI)[1]. A ACPI é um padrão desenvolvido por grandes empresas de tecnologia para configuração e gerenciamento de energia do computador. Na Tabela 2.1, descrevem-se os principais estados de energia de um sistema definidos pela ACPI.

Tabela 2.1: Principais estados de energia de um sistema definidos pela ACPI.

Estado Visível	**Estado ACPI**	**Descrição**
Ativo (*Working*)	S0	O sistema está totalmente utilizável. Os dispositivos que não estão em uso podem ser colocados em estados de baixo consumo de energia.
Suspenso (*Sleep*)	S1, S2, S3	O sistema parece estar desligado, sendo reduzido a um consumo mais baixo. Quanto menor o nível de consumo de energia neste estado, mais o sistema demora para voltar ao estado Ativo.
Hibernação	S4	O sistema parece estar desligado. O consumo de energia é reduzido ao menor nível possível e o sistema salva o seu estado da memória em um arquivo de hibernação no disco.
Desligamento suave (*soft off*)	S5	O sistema parece estar desligado. Porém, alguns componentes permanecem ligados para que o sistema possa voltar ao estado Ativo, como por exemplo, poder reativá-lo pela interface de rede.
Desligamento Mecânico	G3	O sistema fica completamente desligado, consumindo energia zero.

Há também os estados definidos especificamente para os dispositivos. Esses estados variam entre o **D0** e **D3**. O D0 representa o estado ativo; o D1 e D2 representam estados de suspensão (*sleep*); o D3 representa o desligamento do dispositivo, fazendo o dispositivo não responder ao seu barramento.

[1] Advanced Configuration Power Interface, http://www.acpi.info, 2014

2.1.2 Visão Geral

Para que um dispositivo passe de um estado para outro (por exemplo, do estado D0 para o D3), há um custo de tempo e de energia. A potência necessária para a mudança de estado é sempre maior que deixar o dispositivo no estado D0. O Gerenciamento de energia é um problema de predição. Em outras palavras, ele procura prever se um período de ociosidade será longo o suficiente para compensar os custos nas mudanças desses estados de energia de ativação e desativação. O tamanho mínimo de um período de energia ocioso a partir do qual possa haver economia de energia é chamado tempo de equilíbrio (*break-even time*). Ele depende individualmente dos dispositivos e é independente de requisição.

Considere um dispositivo cujo tempo de transição entre dois estados quaisquer seja *To* e a energia gasta nesta transição seja *Eo*, para quaisquer transições (por motivos de simplificação). Considere também que este dispositivo consome uma potência *Pa* no estado ativo e uma potência *Ps* no estado de suspensão. Na Figura 2.1 (a), o dispositivo é mantido no estado ativo; na Figura 2.1 (b), o dispositivo é desligado.

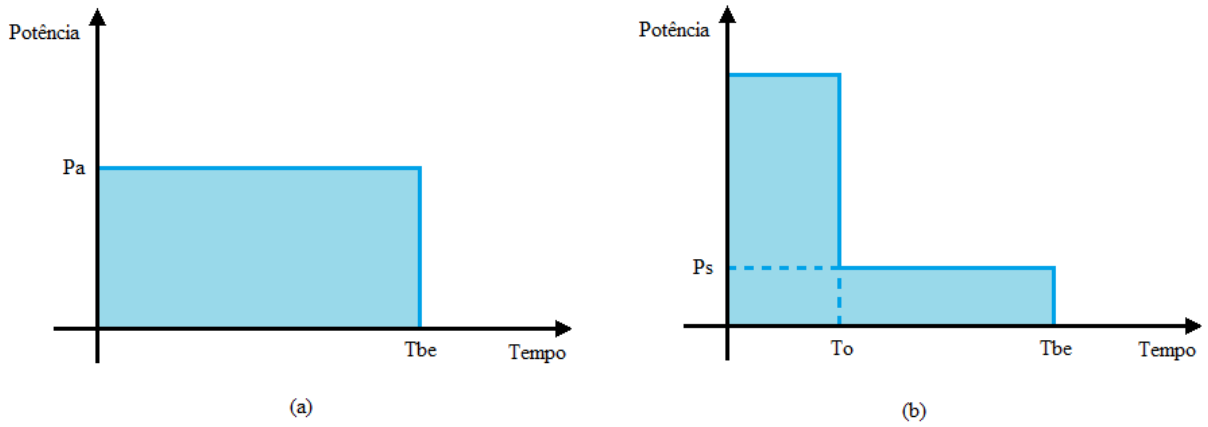

Figura 2.1: Na figura (a) o dispositivo é mantido ativo; na (b), desligado. Ambos gastam a mesma energia.

O tempo de *break-even* (Tbe) iguala o consumo de energia em ambos os casos. Dado que energia é igual o produto do período de tempo pela potência no período, podemos expressar a situação descrita na Figura 2 com a equação (2.1) a seguir.

$$Pa.Tbe = Eo + Ps(Tbe - To) \Rightarrow Tbe = (Eo - Ps.To)/(Pa - Ps) \quad (2.1)$$

Como o tempo de *break-even* deve ser maior que o tempo de transição, podemos concluir que o tempo de *break-even* é melhor expressado com a equação (2.2).

$$Tbe = max[(Eo - Ps.To)/(Pa - Ps),\ To] \quad (2.2)$$

O objetivo principal de qualquer política de gerenciamento dinâmico é fazer o dispositivo entrar no estado de suspensão por pelo menos o tempo *Tbe*. Caso contrário, ele pode acabar consumindo mais energia do que se estivesse sempre ativado. Ou seja, se um dispositivo recebe tantas requisições tal que seu período de tempo entre requisições seja menor que *Tbe*, então não é viável desativar esse dispositivo entre as requisições. Porém, é possível comprometer o desempenho do dispositivo devido aos atrasos percebidos na sua reativação, mesmo havendo redução no consumo de energia.

2.1.3 Políticas de Timeout

Timeout, ou tempo limite, significa o esgotamento de um período de tempo. Políticas de *timeout* são baseadas em um valor de timeout τ. Elas colocam o dispositivo em um estado de suspensão se ele estiver ocioso por um período de tempo maior ou igual a τ. A suposição básica é que se um dispositivo fica ocioso por um período τ, então ele deve ficar ocioso por mais *Tbe* unidades de tempo. A desvantagem dessas políticas é que elas podem desperdiçar energia esperando pelo timeout, enquanto já se poderia entrar em estado de suspensão.

O período de timeout pode ser fixo ou adaptativo. Uma política de *timeout* fixo bem simples seria colocar τ no mesmo valor do *Tbe* daquele dispositivo [KMM94]. Essa política garante que o dispositivo não economizaria mais energia que o dobro da energia economizada por uma política *off-line* ideal (uma política podendo ter o τ menor que *Tbe*, pois sabe que o período entre requisições é suficientemente grande para aquele dispositivo).

Uma política adaptativa modifica dinamicamente τ baseando-se em alguns parâmetros. Em [DKB95], τ é ajustado com base na razão entre o atraso de desempenho e o período de suspensão do período de ociosidade anterior. Se essa razão é alta, significa que o desempenho foi comprometido, e τ deve ser aumentado. Caso contrário, τ deve ser diminuído. Valores de teto e piso foram definidos para τ, para prevenir que a política se tornasse muito agressiva ou muito conservadora.

2.1.4 Políticas Preditivas

Políticas preditivas têm o objetivo de prever a duração do próximo período de ociosidade. Sendo assim, a decisão de suspender ou não o dispositivo depende da previsão desse período de ociosidade ser maior ou menor que *Tbe*.

Hwang e Wu apresentaram um algoritmo de predição que estima o tamanho do próximo intervalo de ociosidade baseando-se no intervalo de ociosidade anterior e na estimativa do intervalo anterior [HW97]. Usando essa abordagem, é possível dar pesos diferentes para valorizar mais a estimativa anterior do que a duração do intervalo anterior e vice-versa. Esse trabalho também propõe estimativas de reativações para reduzir atrasos de desempenho.

2.1.5 Políticas Estocásticas

Um processo estocástico é uma família de variáveis aleatórias $\{X(t), t \in T\}$ definidas em um espaço de probabilidade, indexados pelo parâmetro t (geralmente tempo). O conjunto T é chamado de espaço de parâmetro. Os valores assumidos por X(t) são chamados de estados. Uma variável aleatória é uma função que associa elementos de um espaço amostral a valores numéricos. Resumidamente, um processo estocástico modela variações aleatórias de estados no tempo. Tanto T quanto X(t) podem ser discretos ou contínuos. Quando T é discreto, o processo pode ser chamado de *sequência aleatória*. Quando X(t) é discreto, o processo pode ser chamado *cadeia*. Processos Markovianos ou Cadeias de Markov são processos estocásticos especiais em que o estado futuro depende apenas do estado presente e não dos estados passados. Este tipo de processo também pode ser chamado de processo sem memória (*memoryless process*) [HPS86].

Políticas estocásticas modelam a chegada de requisições ao dispositivo e suas mudanças de estados como processos estocásticos. Sendo assim, minimizar o consumo de energia e atrasos de desempenho se tornam problemas estocásticos de otimização. Há trabalhos que modelam a requisição ao dispositivo, o requisitante e o dispositivo como Cadeias de Markov, tanto discretas quanto contínuas [QP99]. Estes trabalhos se mostraram bastante eficientes tanto na redução do consumo de energia quanto nos atrasos de desempenho, mas se baseiam em distribuições específicas de probabilidade para os tempos de chegada das requisições. Tais abordagens não se mostraram eficazes em situações do mundo real [LCS+00].

2.1.6 Técnicas ao Nível de Processador

Enquanto opera no estado Ativo (D0), um dispositivo ou o processador pode estar em um de vários Estados de Desempenho (*power-performance states*). A implementação desses estados varia entre os dispositivos. *P0* representa o estado de desempenho máximo e *Pn* representa o estado de desempenho mais baixo. O processador é capaz de implementar esses estados modificando a tensão e/ou velocidade de clock (frequência) do seus núcleos. Essa técnica é denominada Escalonamento Dinâmico de Tensão e Frequência (DVFS). A potência dinâmica de um processador é diretamente proporcional a sua utilização, frequência e o quadrado da sua tensão [GCW95]. Por causa dessa relação não linear entre tensão e potência do processador, é melhor espalhar as cargas de trabalho reduzindo o período de ciclo e tensão do processador, do que colocá-lo em um estado de desempenho máximo (P0) para uma carga curta para depois deixá-lo ocioso. Saber quando usar ou não a potência máxima requer a cooperação do escalonador do sistema operacional [BBS00]. A seguir, descrevem-se duas formas de como o escalonador ajuda na DVFS.

Escalonador Baseado em Intervalo

Já foram propostos escalonadores baseados em intervalos que dividem o tempo em intervalos uniformes e analisam a utilização do processador nos períodos anteriores para determinar a tensão do próximo intervalo.

Govil, Chan e Wasserman propuseram alguns algoritmos complexos que estimam a carga futura baseando-se em dois parâmetros: porcentagem de utilização e ciclos em excesso [GCW95]. Porcentagem de utilização representa a fração de ciclos ativos no intervalo anterior, e ciclos em excesso são os ciclos que sobraram do período anterior devido à velocidade insuficiente para completar o trabalho. Neste trabalho, foram propostos, comparados e discutidos sete algoritmos. Em resumo, os algoritmos preditivos propostos obtiveram desempenho pior que os algoritmos simples baseados na média dos períodos anteriores.

Weiser, Welch, Demers e Shenker propuseram três classes de escalonadores baseados em simulação a partir de dados da execução de uma determinada carga de trabalho [WBD94]. O primeiro trata-se de uma abordagem ótima que espalha a computação a ser feita por todo o período da simulação da execução, eliminando assim, todos os períodos de ociosidade. O segundo algoritmo utiliza uma janela de tempo no futuro para determinar a frequência mínima. O terceiro utiliza uma janela de tempo passada para predizer o futuro. Como os dois

primeiros algoritmos são impraticáveis, eles serviram apenas de base de comparação para o terceiro algoritmo, que conseguiu economizar até 50% em circuitos com design conservativo (ex.: 3.3V).

Escalonador para Sistemas de Tempo Real

Escalonar baseado em intervalos é simples e de fácil implementação, mas frequentemente esta estratégia falha em predizer as cargas futuras, comprometendo a qualidade do serviço. Em sistemas que não são tempo real, os ciclos em exceções do período anterior podem ser espalhados nos próximos ciclos. Em sistemas de tempo real, as tarefas a serem computadas possuem restrições de tempo de início, tempo final e quais recursos serão necessários. O escalonamento de tensão e frequência deve ser feito de tal forma que esses requisitos não sejam ignorados. Ou seja, as tarefas devem ser completadas antes do prazo final e de forma a minimizar o total de energia consumida.

Algoritmos de escalonamento que tentam garantir a execução inteira das tarefas dentro dos prazos e restrições pré-determinadas foram propostas por Quan e Hu [QH01]. Sabendo-se dos parâmetros de tempo de execução de cada tarefa, foram propostos dois algoritmos de escalonamento para um conjunto N de tarefas. O primeiro algoritmo é da ordem de $O(N^2)$ de tempo de execução e encontra a menor frequência constante necessária para finalizar cada tarefa. O segundo algoritmo é da ordem de $O(N^3)$ de tempo de execução, baseado no primeiro e produz novos valores de frequência para cada tarefa e define períodos críticos de execução, resultando em períodos ociosos. Sendo assim, esse segundo algoritmo é mais eficiente que o primeiro, pois o processador pode ser desligado nesses períodos de ociosidade.

2.2. Modelos de Potência

Nesta seção são discutidas as formas de estimar a potência consumida em um sistema computacional. Por potência entende-se a grandeza que determina a quantidade de energia consumida a cada unidade de tempo. O Watt é a unidade de potência do Sistema Internacional de Unidades (SI) e é equivalente a um Joule por segundo. Sendo assim, um modelo de estimação de potência também consegue estimar o gasto de energia do sistema em um dado período de tempo, pelo produto da potência média pelo tempo desse período.

Um modelo de estimação do consumo de potência obtém os dados relacionados à utilização do sistema em questão e constrói o modelo a partir deles. A obtenção desses dados

pode ser feita em tempo de execução, chamados modelos em tempo de execução, ou pode ser feita após a execução de uma carga de trabalho, chamados modelos off-line. O arcabouço proposto neste trabalho objetiva estimar potência em qualquer instante de tempo. Sendo assim esta seção dará mais enfoque aos modelos em tempo de execução. Dessa forma, há dois passos necessários para construção desse modelo: escolher o formato dos valores de entrada para o modelo e a identificação de uma ferramenta para treinar e validar o modelo. Nas subseções a seguir são descritos esses dois passos com mais detalhes, quais as estratégias para modelar o consumo do processador e quais estratégias para modelar o consumo de todo o sistema.

2.2.1 Entradas do Modelo

Modelos de estimação de potência se utilizam de informações como a utilização dos subsistemas (como por exemplo, CPU e memória) ou da informação fornecida pelos contadores de monitoramento de desempenho (*performance monitoring counters* – PMC), também referenciados na literatura como contadores de desempenho de hardware (*hardware performance counters* – HPC). A informação de utilização não é obtida diretamente do subsistema, mas sim obtida, calculada e fornecida pelo Sistema Operacional. Por exemplo, tanto o núcleo do Linux quanto o do Windows calculam a utilização de CPU (e por núcleo), utilização de disco, e utilização de largura de banda. Dessa forma, essas informações são classificadas como *entradas de SO*.

Ao contrário das informações de utilização, os contadores de monitoramento de desempenho são fornecidos diretamente pela CPU. Essas informações são classificadas como *entradas de hardware*. Um CPU moderno pode fornecer um ou mais registradores específicos que podem ser usados para contar certos eventos de microarquitetura. Por exemplo, tais eventos podem ser gerados quando o processador finaliza uma instrução ou quando acessa o cache. A quantidade de eventos contáveis vem crescendo a cada geração, família e modelo de processadores.

O uso de contadores de monitoramento de desempenho não é simples. Primeiramente, a quantidade de eventos que podem ser contados simultaneamente é limitada e varia de arquitetura para arquitetura. Logo, é muito importante selecionar os contadores que mais se relacionam com o consumo de energia. Segundo, a seleção dos contadores dependerá de qual estratégia de treinamento é utilizada. Por exemplo, aplicações de treinamento que geram mais

operações com inteiros do que outras operações não alimentarão os contadores relacionados às operações de ponto flutuante [SSI].

2.2.2 Estratégias de Treinamento e Validação

A maioria das abordagens envolvem benchmarks para treinar e testar os modelos de estimação de potência. Portanto, é apropriado descrever brevemente o que são benchmarks e porque eles são usados em modelos de estimação de potência.

Benchmarks são programas especialmente feitos para realizar cargas de stress em alguns subsistemas de um sistema computacional de uma maneira controlável e repetível. Enquanto o benchmark está sendo executado, conjuntos específicos de indicadores de desempenho são amostrados e avaliados para estabelecer relação entre a utilização do subsistema e seu desempenho ou consumo de potência. Sabendo-se que o nível de stress induzido pelo benchmark é bem definido, espera-se que sejam comparáveis com os resultados de diferentes instâncias de um mesmo subsistema (por exemplo, diferentes modelos de processador), que rodaram o mesmo benchmark. Benchmarks que geram carga por um intervalo de tempo fixo são do tipo baseado em rendimento (*throughput-based*). O outro tipo complementar de benchmark é chamado de baseado em tempo (*time-based*), pois executa uma carga toda para depois medir quanto tempo foi gasto. Benchmarks que combinam esses dois aspectos são chamados híbridos [SPEC].

Compreensivelmente, um modelo de estimação treinado e testado com uma variedade maior de benchmarks pode apresentar um erro de estimação maior que um modelo treinado e testado por um número limitado de benchmarks. No primeiro caso, o modelo tem de correlacionar um número limitado de valores de entrada a uma variedade grande de características de consumo de potência. Enquanto que no segundo caso, o modelo treinado com um número menor de benchmarks tenderá a ser enviesado a esses benchmarks [BBC+12].

2.2.3 Modelagem do Consumo da CPU

O consumo de potência de microchips baseados em CMOS (*Complementary Metal-Oxide-Semiconductor*) pode ser classificado em dinâmico e estático [LJ06]. O consumo dinâmico de potência é dissipado devido às atividades de chaveamento de níveis lógicos, enquanto que o consumo estático de potência é devido às correntes de fuga. O consumo de potência total da

CPU é dado pela soma dessas duas potências. A velocidade do processador ou capacidade computacional é quase linearmente proporcional à velocidade de *clock* f, e o consumo dinâmico de potência da CPU é proporcional à multiplicação da velocidade de *clock* pelo quadrado do suprimento de tensão V [CSB92]. Então, o consumo dinâmico de potência da CPU pode ser formulado pela equação (2.3) a seguir.

$$P_{dynamic} = \alpha C V^2 f \tag{2.3}$$

A variável $P_{dynamic}$ representa o consumo dinâmico da CPU, C representa a capacitância e α representa o fator de atividade do processador (porcentagem de chaveamentos feitos a cada ciclo).

Quando o processador é exposto a um baixo nível de tensão, a frequência é reduzida por causa de um maior atraso no circuito. Sendo assim, é possível dizer que $f \approx \frac{(V-V_T)^2}{V}$, onde V_T é a tensão de limiar, que é bem menor que a tensão V fornecida, fazendo V ser ignorável [HKQ+99]. Deixando $f = kV$, onde k é uma constante, a equação (2.4) pode ser reescrita na forma da equação (2.4).

$$P_{dynamic} = \alpha . C . f^3 / k^2 \tag{2.4}$$

O consumo estático é dado pela equação $P_{static} = I_q V$, onde I_q é a corrente de fuga e V é tensão fornecida. Como discutido anteriormente, é possível simplificar fazendo $f = kV$ com uma constante k, o consumo estático pode ser reescrito com a ajuda da equação (2.5) a seguir, assumindo que há pouca mudança na corrente de fuga.

$$P_{static} = \gamma . f \tag{2.5}$$

Outros componentes com grande consumo de energia em sistemas multicore são as memórias *cache*. A maioria dos processadores de hoje têm memórias *cache* integradas no circuito, que muitas vezes ocupam grande área no chip. Portanto, o consumo de energia devido a acessos à memória *cache* deve ser contabilizado. O consumo de energia das memórias *cache* será proporcional ao número de acessos a elas. É assumido que os diferentes níveis de memória *cache* (L1, L2, L3) consomem aproximadamente a mesma potência. O consumo de energia das memórias *cache* é dado pela equação (2.6), onde N é a quantidade de acessos ao *cache* e ε uma constante:

$$P_{cache} = \varepsilon N \tag{2.6}$$

Gupta et. al. observaram que as subpartes que não fazem parte do núcleo não são escaláveis [GBK+12]. Neste caso, é possível supor que o consumo de energia estática quase não é alterado com o número de núcleos. Portanto, o número de núcleos ativos afeta o consumo de energia dinâmica, enquanto que o estado do próprio processador de múltiplos afeta o consumo de energia estática. Com esta premissa, o consumo de energia da CPU é modelado da seguinte forma, na equação (2.7) a seguir, onde c é o número de núcleos ativos.

$$P_{CPU} = P_{dynamic} \cdot c + P_{static} + P_{cache} = \beta f^3 c + \gamma f + \varepsilon N \quad (2.7)$$

2.2.4 Modelagem do Consumo dos Outros Subsistemas

Embora a CPU seja o consumidor de energia predominante de um computador, dispositivos de memória principal, armazenamento de dados e de rede também consomem uma quantidade considerável de energia que não pode ser desconsiderada. Desenvolver modelos distintos para todos estes componentes não é trivial. A alternativa é estimar o consumo total de energia de todo o computador.

Heath et al. [HDC+05] propõem um modelo de potência para sistemas desktop em geral. O consumo de potência é estimado com um modelo linear que emprega unicamente métricas de utilização. O modelo para cada computador desktop é dado pela fórmula (2.8) a seguir, onde P_{idle} é a potência ociosa consumida, M_r é a matriz dos valores máximos de consumo para um determinado recurso de hardware *r*, U_r é a utilização do recurso de hardware *r* e C_r é a matriz de capacidade do recurso *r*.

$$P = P_{idle} + \sum_r M_r \frac{U_r}{C_r} \quad (2.8)$$

Da mesma forma, Economou et al. propõem um modelo de equação linear para estimar o consumo de potência de um servidor em um cluster usando métricas de utilização como entradas do modelo [ERK+06]. Estas métricas referem-se à utilização da CPU (u_{cpu}), acessos à memória principal (u_{mem}), à taxa de E/S do disco rígido (u_{disk}), e à taxa de E/S da interface de rede (u_{net}). Os autores empregam um benchmark feito por eles mesmos para aplicar cargas de trabalho em isolado para cada subcomponente, medindo também o seu respectivo consumo de potência. Um servidor *Blade* com um processador AMD Turion e um servidor com quatro processadores Intel Itanium 2 são usados para treinar o modelo e para a obtenção dos coeficientes do modelo linear. Os modelos de potência para o servidor Blade e para o Intel são descritos nas equações (2.9) e (2.10) a seguir, respectivamente.

$$P_{blade} = 14.45 + 0.236.u_{cpu} - (4.47E-8).u_{mem} + 0.00281.u_{disk} + (3.1E-8.unet \quad (2.9)$$

$$P_{itanium} = 635.62 + 0.1108.u_{cpu} - (4.05E-8)u_{mem} + 0.00405.u_{disk} + 0.u_{net} \quad (2.10)$$

Fan et al. propõem um modelo de equação não linear usando a utilização da CPU como a entrada do modelo [FWB07]. A fase de formação de modelo é similar ao trabalho de [ERK+06], mas para contabilizar a relação não linear entre o desempenho e o consumo de potência, eles estenderam a estrutura de um modelo linear com um fator de correção de erro. O modelo resultante tem a forma descrita na equação (2.11), onde P_{busy} representa a potência da CPU sob uma utilização de 100%, r é a correção ou fator de calibração, minimizando o erro quadrado do modelo.

$$P = P_{idle} + (P_{busy} - P_{idle})(2u_{cpu} - u_{cpu}^{r}) \quad (2.11)$$

Em resumo, os modelos de consumo de potência para os outros subsistemas tomam uma diversidade de parâmetros de entrada para a tarefa de estimativa. Entre estes estão os indicadores de utilização relacionados com a CPU e o subsistema de memória, o que indica que estes subsistemas são os consumidores predominantes da energia total fornecida ao sistema como um todo.

2.3. Desenvolvimento Baseado em Componentes

O Desenvolvimento Baseado em Componentes (DBC) surgiu como uma nova perspectiva de desenvolvimento de software caracterizada pela composição de partes já existentes. Conforme Szyperki, construir novas soluções pela combinação de componentes desenvolvidos e comprados aumenta a qualidade e dá suporte ao rápido desenvolvimento, levando à diminuição do tempo de entrega do produto ao mercado [SZY99]. Os sistemas definidos através da composição de componentes permitem que sejam adicionadas, removidas e substituídas partes do sistema sem a necessidade de sua completa substituição. Com isso, DBC auxilia na manutenção dos sistemas de software, por permitir que os sistemas sejam atualizados através da integração de novos componentes e/ou atualização dos componentes já existentes.

2.3.1 Definições

Inúmeras definições de componentes, e componentes reusáveis, são encontradas na literatura. A definição apresentada por Sametinger considera que “componentes de software reusáveis são artefatos autocontidos, facilmente identificáveis que descrevem e/ou executam funções específicas e têm interfaces claras, documentação apropriada e uma condição de reuso definida” [SAM97]. Nessa definição, é apresentado que os componentes podem ser vistos como alguma parte do sistema de software que é identificável e reutilizável, ou como o estado seguinte de abstração depois de funções, módulos e classes.

A segunda definição a ser considerada é apresentada por Brown, na qual componente é caracterizado como “conjunto independente de serviços reutilizáveis” [BRO97]. Esta definição apresenta dois elementos principais, “independente” e “serviços reutilizáveis”, que podem ser considerados independentemente.

Por “serviços reutilizáveis”, indica-se a necessidade da existência de uma especificação que apresente o que o componente faz e como se comporta quanto utilizados. Em geral, a implementação pode ser realizada em diferentes linguagens de programação ou plataformas. Um componente pode ser substituído por outro em uma aplicação, desde que ambos atendam à mesma especificação.

O segundo elemento da definição de Brown, “independente”, indica a ausência de vínculo do componente com o contexto em que ele pode ser usado. A expectativa de que os componentes cooperem entre si para completar uma solução não deve estar vinculada à existência de dependências entre eles, pois os componentes não devem ser desenvolvidos com dependências fixas entre si.

Sumarizando, a definição de Brown ressalta um aspecto mais maduro de DBC, onde se preocupa exclusivamente em componentes como sendo a abstração seguinte a funções, módulos e classes. Em contrapartida, a definição de Sametinger trata componentes não apenas como componentes de código, mas também como todo possível tipo de componente independente de forma. Essa conceituação ressalta melhor quando o foco não estiver apenas no reuso de código, e sim de componentes em todas as fases do processo de desenvolvimento de software.

2.3.2 Modelo e Arcabouço de Componentes

Conforme [BAC02], existem diferenças quanto a como essas categorias são nomeadas e não quanto às categorias propriamente ditas. Existe um relativo consenso na literatura quanto ao uso do nome modelo de componentes para identificar o conjunto de padrões e convenções com as quais os componentes devem estar em conformidade. Entretanto, a infraestrutura que dá suporte a estes modelos é encontrada na literatura tanto identificada como arcabouço de componentes como infraestrutura de componentes [BAC02] [BRO00].

Segundo Bachmann [BAC02], um modelo de componente representa um elemento da arquitetura do sistema na qual são definidos os padrões e convenções impostas aos componentes do sistema, de modo a descrever a função de cada um e como eles interagem entre si. Com isso, busca-se expressar restrições de projeto arquitetural ou global.

Já arcabouço de componente, conforme Brown, é a implementação de serviços que dão suporte ou reforçam o modelo de componentes [BAC02]. A função do arcabouço é gerenciar os recursos compartilhados pelos componentes e prover um mecanismo que possibilite a comunicação (interação) entre eles. A infraestrutura oferecida pelos arcabouços de componentes impõe restrições e regras no projeto e implementação, as quais devem ser consideras pelo modelo de componentes. O relacionamento entre componentes, modelo e arcabouço de componentes pode ser identificado pela Figura 2.2.

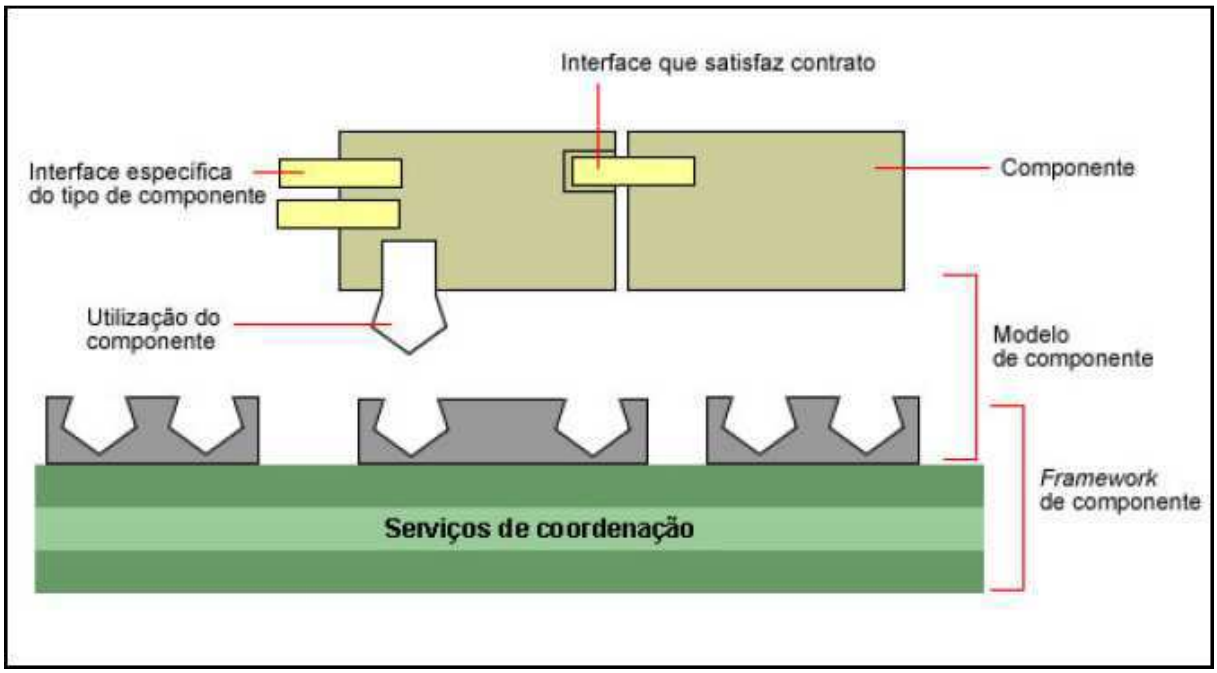

Figura 2.2: Relacionamento entre componente, modelo e arcabouço de componentes.

A Figura 2.1 permite identificar o arcabouço de componentes como infraestrutura de suporte à comunicação e ligação dos componentes, através do fornecimento de serviços de coordenação. Já o modelo de componentes identifica as definições quanto a tipos de componentes, formas de interação e recursos necessários. Os componentes definidos para executarem determinadas funcionalidades devem estar em conformidade com as definições do modelo de componentes, pertencendo a um dos possíveis tipos e respeitando as formas de interação definidas, além de se utilizarem dos serviços disponibilizados. Por fim, tem-se que a utilização dos componentes caracteriza a ligação entre o modelo e o arcabouço de componentes.

Capítulo 3

Arquitetura para o Desenvolvimento de Aplicações para Gerenciamento de Energia

Nesta seção são descritos os requisitos funcionais e não funcionais necessários para seu desenvolvimento. Em seguida é apresentada a arquitetura do arcabouço para desenvolvimento de aplicações gerenciadoras de energia propriamente dita, destacando seus principais componentes, com as respectivas responsabilidades e como eles interagem entre si e como estender suas funcionalidades.

3.1 Requisitos de um Aplicativo para Gerenciamento de Energia

De uma maneira geral, devem-se fornecer funcionalidades que respondam a questionamentos, tais como:

- Como obter as informações de utilização e de contexto da máquina?
- Como controlar ou mudar o estado os dispositivos internos da máquina?
- Como definir uma política?
- Como definir qual abordagem para o modelo de potência?
- Como criar novos perfis?

A fim de possibilitar o desenvolvimento de aplicações de gerenciamento de energia, a arquitetura para gerenciamento de energia deve permitir a criação de políticas de economia de energia. Para que uma política funcione, ela deve ser capaz de medir a utilização de recursos de hardware da máquina alvo e outras informações do contexto da máquina como um todo, como também controlar os diversos dispositivos de hardware da máquina. A combinação dessas duas estratégias (monitoramento e controle) forma o alicerce para qualquer aplicação nesse domínio. Outro requisito fundamental é que a aplicação gerenciadora de energia deve

realizar baixa utilização de recursos de hardware e, consequentemente, de energia. Obviamente, ela não deve consumir mais energia do que é capaz de economizar.

Independente do desempenho do hardware subjacente, sendo ele de última geração ou não, o que faz um computador suprir as necessidades do usuário final são as aplicações que nele rodam. Muitas estratégias em gerenciamento dinâmico de energia são específicas para uma camada de hardware e software subjacente. Porém, as restrições de tempo de resposta e desempenho em geral que as aplicações impõem ao sistema operacional e hardware subjacente podem ser especificadas independentemente de plataforma. Como elaborado no trabalho PASA [CRM02], que define uma interface padrão a fim de que o gerenciamento de energia seja independente de plataforma, é preciso definir um conjunto de interfaces bem relacionadas a fim de garantir uma troca de informações bem feita sobre consumo e restrições.

Além dos requisitos funcionais expostos até então, a estimativa de consumo de energia da máquina subjacente é outro ponto pouco abordado em trabalhos sobre aplicações para gerenciamento de energia. Muitas estratégias para definições de políticas baseiam-se nos dados de utilização dos recursos de hardware. Porém, como apontado em [BBC+12], nem sempre a utilização de energia é diretamente proporcional às taxas de utilização de hardware, como processamento, bytes de entrada e saída, entre outros, compondo assim um modelo de equações multivariadas. Em alguns casos, a utilização de energia é mais bem expressa em termos dos eventos gerados pelos dispositivos internos à máquina, fazendo com que o consumo de energia seja mais bem explicado por um modelo de autômatos. Sendo assim, a arquitetura deve prover meios para a construção de modelos de potência tanto baseados em eventos quanto baseados em taxas de utilização.

Outro ponto, não muito abordado nos trabalhos relacionados, é a possibilidade da agregação de diferentes políticas, formando assim um perfil de energia. Um perfil é o conjunto composto por uma ou mais políticas, a fim de facilitar e abstrair a utilização do gerenciador para o usuário final. Como abordado no Windows a partir da versão Vista[2], são definidos três diferentes perfis: o perfil de economia, o perfil balanceado e o perfil de alto desempenho.

Para complementar os requisitos funcionais descritos até aqui, também se faz necessário abordar alguns requisitos não funcionais. Para que os desenvolvedores de aplicações possam criar suas aplicações a partir de uma arquitetura para gerenciamento de energia, eles devem poder saber como definir políticas. A arquitetura deve disponibilizar serviços que maximizem

[2] Managing Power in Windows Vista http://msdn.microsoft.com/en-us/library/ms700612(v=vs.85).aspx

a produtividade dos desenvolvedores e consequentemente abstraiam o processamento interno realizado.

Na Tabela 3.1 são sumarizados os requisitos funcionais descritos anteriormente, e na Tabela 3.2 são sumarizados os requisitos não funcionais. A implementação desses requisitos constitui o projeto da arquitetura apresentada neste trabalho.

Tabela 3.1: Requisitos funcionais da arquitetura.

Código	Requisito Funcional
RF1	Economizar energia da máquina subjacente.
RF2	Criação de políticas.
RF3	Acesso às informações de contexto de utilização da máquina.
RF4	Controle no estado dos dispositivos da máquina.
RF5	Criação de perfis de energia.
RF6	Estimação de energia da máquina.

Tabela 3.2: Requisitos não funcionais da arquitetura.

Código	Requisito Não Funcional
RNF1	Pouca utilização de energia pela aplicação de gerenciamento.
RNF2	Facilidade na criação de políticas.
RNF3	Possibilidade de extensão da arquitetura.

3.1.1 Plataforma Alvo

Antes de descrever a arquitetura propriamente dita, é interessante primeiro definir qual o escopo das características da máquina alvo da aplicação de gerenciamento de energia. Um sistema computacional com recursos de gerenciamento de energia será chamado aqui de Sistema de Gerenciamento de Energia ou Provedor de Serviços (PS), como proposto no trabalho de Luiz, Perkusich e Lima [LPL10]. Como mostrado na Figura 3.1, um sistema com recursos de gerenciamento de energia é composto por subsistemas tais como processador, memória, interfaces de rede, discos rígidos e um display. O Sistema de Gerenciamento de Energia recebe requisições de um Requisitante de Serviços (RS), o que representa a carga do sistema, como por exemplo, operações de entrada e saída do disco rígido, ou operações de *download* e *upload* das interfaces de rede. Os subsistemas possuem um conjunto de estados

de energia, cada um com um consumo de energia e nível de desempenho diferente. Se algum desses subsistemas estiver ocioso ou subutilizado, então a economia de energia será alcançada alterando o estado desses subsistemas.

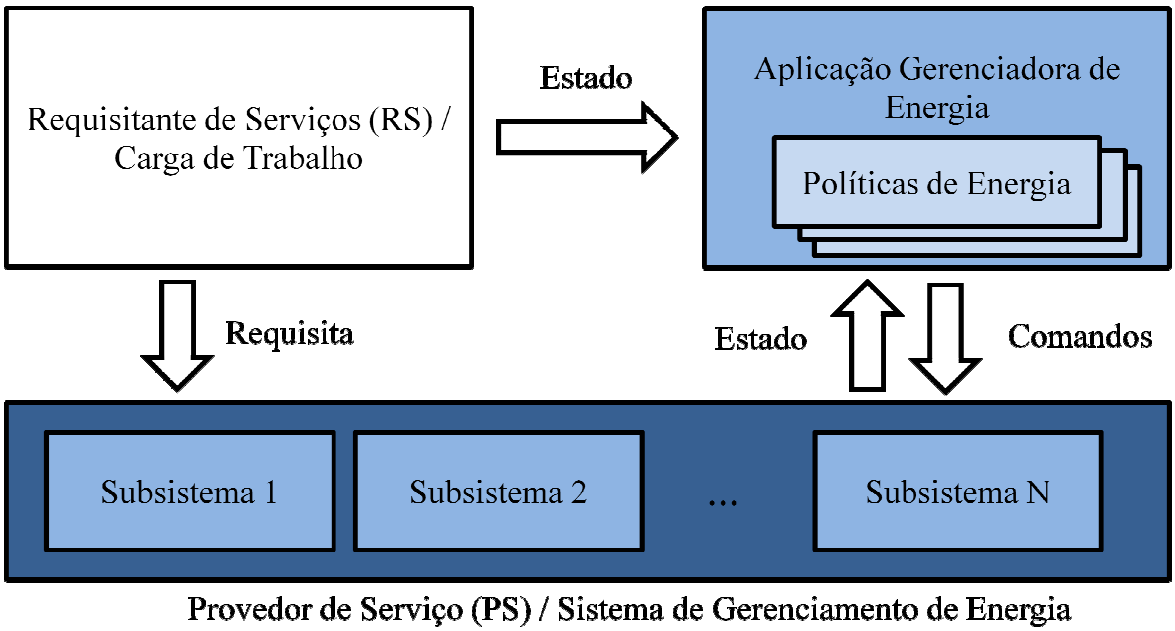

Figura 3.1: Modelo básico de um sistema de energia gerenciável.

3.2 Arquitetura

A partir dos requisitos descritos anteriormente, optou-se pelo Modelo Baseado em Componentes para a elaboração da arquitetura. Dentre os modelos de desenvolvimento existentes, o Modelo Baseado em Componentes possibilita que os sistemas sejam modificados pela integração de novos componentes e/ou substituição dos componentes existentes. A arquitetura descrita nesta seção é a representação de maior nível de abstração da solução proposta neste trabalho. Como parte dos objetivos propostos, ela se instancia em forma de um arcabouço de software, formando assim a parte mais concreta da solução. Nas próximas subseções, são apresentados os componentes, as instâncias e a forma de reutilizá-la.

A arquitetura para o desenvolvimento de aplicações para gerenciamento de energia está descrita na Figura 3.2. Ela é composta de oito subcomponentes, e seis deles estão divididos em três pacotes, formando os principais componentes. Ou seja, cada pacote é tratado aqui como um componente. O componente GUI, que é um acrônimo para *Graphical User Interface* (Interface Gráfica de Usuário) bastante conhecido, deverá conter os elementos

gráficos para que o usuário acesse e configure o gerenciador de energia. Isso inclui, por exemplo, alterar o estado dos dispositivos e configurar as políticas de energia; a GUI também deve oferecer interfaces para que haja uma comunicação voltando dos outros componentes a fim de notificar eventos. O componente Fachada oferece uma interface única a fim de facilitar o desenvolvimento do componente GUI, sendo baseada no padrão de projeto *Fachada* [MBS12].

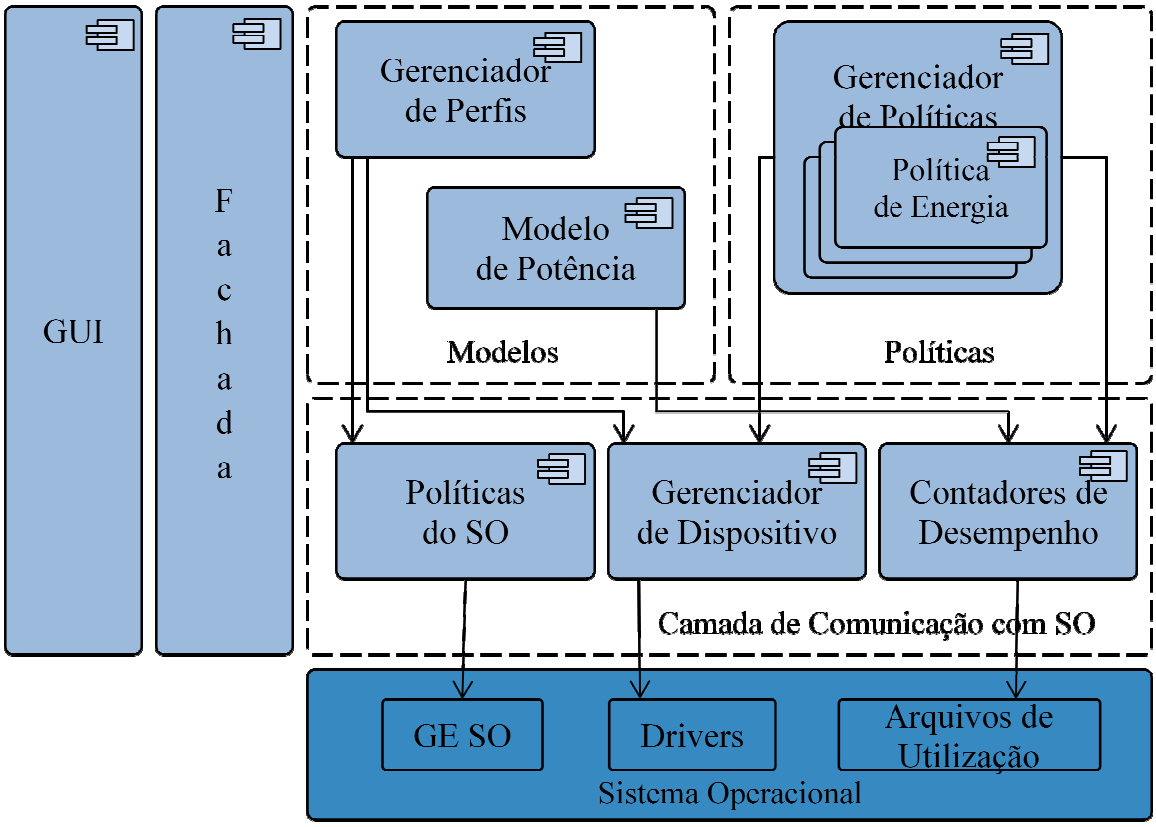

Figura 3.2: Visão geral da arquitetura.

As subseções a seguir descrevem os pacotes Modelos, Políticas e Camada de Comunicação com o SO da Figura 3.2, detalhando seus componentes internos e seus relacionamentos.

Modelos

Esse pacote congloba dois componentes chamados Gerenciador de Perfis e Modelo de Potência. Eles são responsáveis por guardar os estados importantes para o gerenciador de energia, como quais as constantes que compõem o modelo de potência e os estados de um perfil.

O Gerenciador de Perfis modela e contém as funcionalidades para os perfis de energia. Cada perfil é composto por um conjunto de configurações do gerenciamento de energia do sistema operacional subjacente, como tempos de desativação ociosa de dispositivos e quais estados padrão para os dispositivos; e por um conjunto de políticas. As configurações definidas em um perfil são aplicadas quando tal perfil é **ativado**. Só pode haver um perfil ativado por vez.

O Modelo de Potência é responsável por estimar o consumo de potência geral do sistema gerenciável de energia. Para este propósito, ele precisa utilizar o componente Contadores de Desempenho para obter as informações sobre o estado dos subsistemas e suas taxas de utilização, seja via acesso direto ou orientado a eventos.

Políticas

Esse pacote é composto por um único componente chamado Gerenciador de Políticas. Ele é responsável por criar e gerir uma ou mais Políticas de Energia. Embora este componente também, de certa forma, guarde estados da aplicação gerenciadora de energia, o foco dele é mais na lógica da aplicação. Uma política de energia nada mais é que um algoritmo usado para controlar o estado dos subsistemas, tendo como entrada as informações de utilização dos subsistemas, como tempo de ociosidade por exemplo. Para que uma política de energia funcione, ela precisa obter tais informações de utilização do componente Contadores de Desempenho, definir o novo estado do dispositivo, controlá-lo via o componente Gerenciador de Dispositivos e notificar a Fachada através de eventos.

Camada de Comunicação com o Sistema Operacional

Esse pacote é composto de três componentes responsáveis basicamente por obter as informações de estado e utilização dos subsistemas e controlá-los. Como citado anteriormente, o componente *Contadores de Desempenho* é responsável por obter as taxas de utilização do sistema, como utilização do processador, quantidade de memória principal utilizada, taxas de bytes de entrada e saída do disco rígido, entre outras. O componente *Gerenciador de Dispositivos* é responsável por controlar o estado dos subsistemas, ativando-os ou desativando-os, alterando o brilho da tela e mudando o nível de desempenho dos subsistemas, por exemplo. Finalmente, o componente *Políticas do SO* é responsável por fazer

a ponte entre o Gerenciador de perfis e as configurações de energia pré-existentes no sistema operacional.

Sistema Operacional

Esse componente como descrito na Figura 3.2 representa o sistema operacional subjacente ao gerenciador de energia. Para que o arcabouço seja efetivamente implementado em uma aplicação de gerenciamento, é necessário que o sistema operacional forneça meios de obter informações de utilização e controle dos componentes, e fornecer algum gerenciamento de energia interno.

3.3 O Projeto do Arcabouço

Nesta subseção será descrito o projeto do arcabouço de software para o desenvolvimento de aplicações gerenciadoras de energia baseadas na arquitetura descrita anteriormente. Também será descrito como fazer para desenvolver uma aplicação baseada nesse arcabouço, descrevendo quais interfaces devem ser implementadas e como funcionam os algoritmos genéricos de alterações no arcabouço.

Para o escopo deste trabalho, um arcabouço de software é uma plataforma para o desenvolvimento de aplicações em software. Ele permite que os desenvolvedores estendam suas funcionalidades ou modifiquem as existentes. O arcabouço proposto foi desenvolvido usando a linguagem de programação C# com o auxílio da plataforma .Net 4.5 Framework da Microsoft. As seções seguintes descrevem os componentes de cada pacote da Figura 3.2 a fim de explicar a implementação de cada componente e o fluxo de desenvolvimento do arcabouço em uma abordagem orientada a objetos.

3.3.1 Fachada

Além de fornecer uma interface única para o componente GUI, este componente contém a classe responsável por tratar todos os métodos de fábrica para a inicialização dos outros componentes e é a peça chave para modificar e estender o arcabouço. Esta classe chama-se *PowerManager* e possui os métodos principais de operação da aplicação gerenciadora de energia, o qual pode ser sumarizado em poder inicializar e parar suas funcionalidades como um todo. Como essa classe trata-se de uma fachada, ela não possui lógica de negócio. Na Tabela 3.3 são sumarizados os métodos disponibilizados pela classe *PowerManager*.

Tabela 3.3: Métodos da classe PowerManager.

Método	Descrição
init()	Ativa os recursos de gerenciamento de energia da aplicação.
stop()	Desativa os recursos de gerenciamento de energia.
enableDevice()	Coloca um dispositivo específico no estado ligado.
disableDevice()	Cola um dispositivo específico no estado desligado.
enablePowerEstimative()	Inicializa o módulo de estimação de energia.
hasDevice()	Verifica se um dispositivo específico está presenta na máquina.
getPMOSConfiguration()	Obter uma configuração específica das configurações de gerenciamento de energia do sistema operacional.
setPMOSConfiguration()	Modifica uma configuração específica das configurações de gerenciamento de energia do sistema operacional
enablePolicy()	Inicia a aplicação de uma política específica.
disablePolicy()	Para a aplicação de uma política específica.
enableProfile()	Ativa um perfil de energia específico.
getCurrentProfile()	Obtém qual perfil de energia está ativo no momento.
getAvailabeProfiles()	Retorna uma lista com todos os perfis disponíveis.

3.3.2 Gerenciador de Perfis

Este componente é responsável por adicionar, listar, remover e apagar perfis de energia. Os perfis são salvos na forma de arquivos de propriedade, ou seja, uma listagem no estilo chave-valor contendo todas as configurações do perfil. Um arquivo de propriedades facilita a modificação das políticas de energia em tempo de execução, permitindo que os desenvolvedores realizem testes mais facilmente para criar melhores políticas para plataformas específicas.

Este componente contém quatro classes, uma para modelar o perfil de energia, outra para transformar um perfil de energia em arquivo, outra para fazer essa operação inversa e uma para gerenciar e ativar um determinado perfil. Por isso, ele usa o componente *Políticas do SO* para modificar as configurações de energia do sistema operacional estados. Se alguém quiser adicionar recursos extras a um perfil, é possível estender as classes de modelagem do perfil e as classes responsáveis por analisar os arquivos de texto.

3.3.3 Modelo de Potência

Componente capaz de estimar o consumo de potência do sistema de energia gerenciável, tanto da energia como um todo quanto da energia consumida por cada um de seus subsistemas ou dispositivos. Esses subsistemas são, por padrão do arcabouço, o processador, tela, interface Wi-Fi, interface Ethernet, câmera embutida e driver de CD. A estimativa do consumo de energia é feita com o auxílio de um modelo de regressão linear, armazenando os coeficientes da equação em arquivo xml. Como mostrado na Figura 3.3, este componente é composto por classes capazes de fazer a análise e conversão dos arquivos xml, modelagem dos modelos de potência e estimação do consumo propriamente dita.

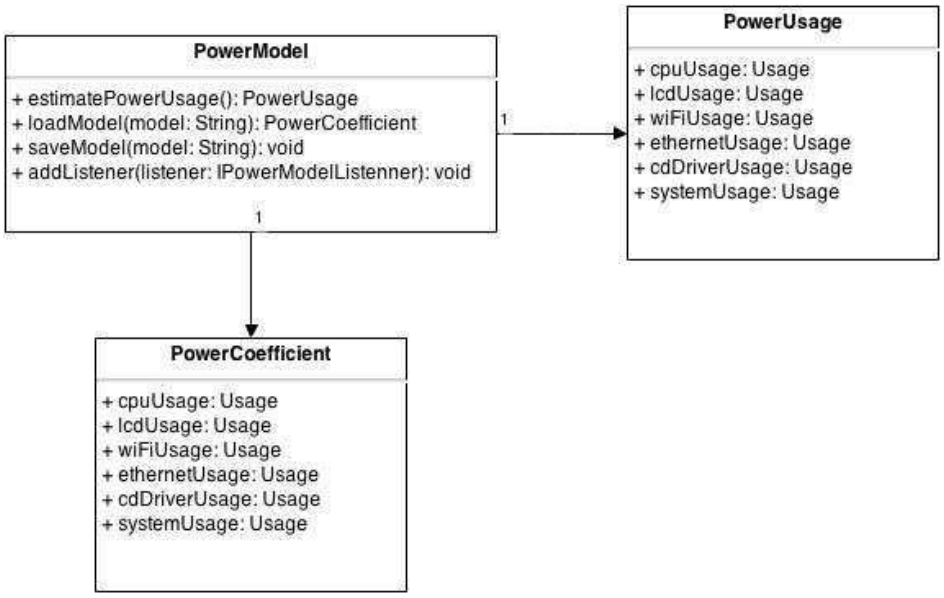

Figura 3.3: Diagrama de classes do componente Modelo de Potência.

Para estender o funcionamento do modelo de potência, é possível estender a classe *PowerModel* e sobrescrever o método responsável por estimar o consumo de potência, chamado *estimatePowerUsage()* e mudar a inicialização deste componente em *Fachada*. A classe *PowerModel* implementa o padrão Observer [MBS12], possuindo métodos para adicionar *Listeners* a serem notificados quando a estimação de energia estiver maior que um patamar predeterminado. A interface ouvinte é chamada *PowerUsageListener* e pertence ao compoenente GUI.

3.3.4 Gerenciador de Políticas

Uma política de energia é definida aqui como sendo uma estratégia para modificar e controlar a configuração do sistema de energia gerenciável baseado em alguns parâmetros. A estratégia de controle da política pode ser desencadeada por algum evento de sistema ou após um intervalo periódico de tempo.

Para implementar uma política, é necessário estender a classe abstrata chamada *PowerPolicy* e escrever o método responsável por aplicar a estratégia de controle, chamado *applyControl()*. Também é necessário implementar o método que decide se a estratégia de controle deve ser aplicada, no caso da política baseada em periodicidade, chamado *isReadyToApply()*. As políticas também são inicializadas no componente Fachada, sendo adicionadas através da classe *PowerManager* na classe *PolicyManager*.

A classe *PolicyManager* possui internamente um escalonador para as políticas baseadas em periodicidade. O escalonador trata-se de um laço que excuta os métodos isReadyToApply() periodicamente, de acordo com as periodicidades das respectivas políticas. Para adicionar uma nova política baseada em periodicidade, basta chamar o método *addPeriodicPolicy()* na classe *PolicyManager* passando como parâmetro o objeto da política e sua periodicidade, em segundos.

Para as políticas de energia baseadas em eventos, a classe *PolicyManager* faz o uso da API do Windows para o registro de eventos e não faz o uso de checagens periódicas, a fim de diminuir o overhead de utilização do software de gerenciamento de energia. São definidos alguns eventos pré-definidos, como o de bateria abaixo de um patamar, mudança de estado de dispositivo, entre outros. Sendo assim, a classe *PolicyManager* guarda um mapeamento entre um tipo de evento e uma lista de políticas disparáveis por esse evento. Sendo assim, quando o evento ocorre, é chamado direto o método *applyControl()* das respectivas políticas relacionadas ao determinado evento. Para adicionar uma nova política baseada em evento, deve-se chamar o método *addEventPolicy()* passando a política e o identificador do evento pré-definido. Na Figura 3.4 é ilustrada a relação entre essas entidades.

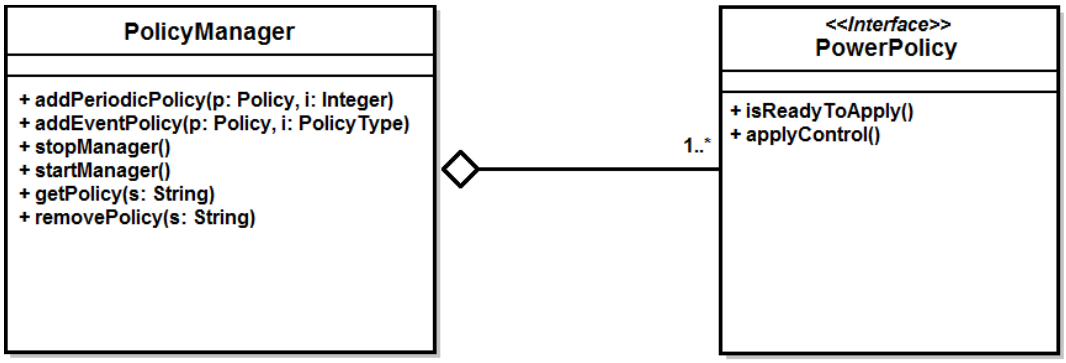

Figura 3.4: Modelo de classes do Gerenciador de Políticas.

3.3.5 Componentes da Camada de Comunicação com o SO

Estes três componentes são definidos por três interfaces chamadas *IPowerSettings*, *IDeviceManager* e *PerformanceCounter*, respectivamente. Elas são criadas e gerenciadas por uma classe chamada *OSFachada*, que fornece uma única interface para acessar as instâncias dessas classes. Na Figura 3.5, é mostrado um diagrama de classes de como estes elementos estão relacionados e como a classe *PowerManager* os acessa. A classe ProfilesManager é usada para o gerenciamento de perfis, pertencendo ao componente *Gerenciador de Perfis*.

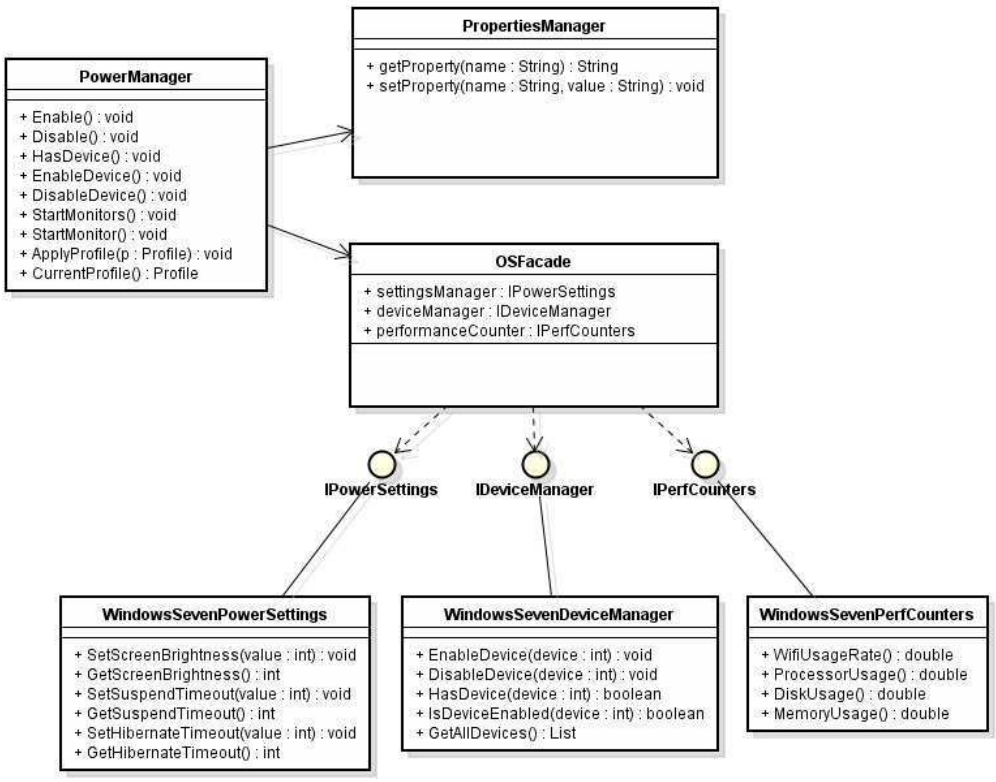

Figura 3.5: Diagrama de classes do relacionamento entre as entidades da camada de comunicação com o SO.

Capítulo 4

Estudo de Caso

Com o intuito de validar o arcabouço de software para o desenvolvimento de aplicativos gerenciadores de energia, neste capítulo é apresentado o desenvolvimento de um estudo de caso para demonstrar como os requisitos propostos se relacionam com a arquitetura. Inicialmente, é realizada uma breve descrição da aplicação, apresentando suas principais funcionalidades e como elas se integram com o arcabouço. Ao final, apresentam-se os resultados dos testes de potência e outras avaliações.

4.1 Descrição da Aplicação

O Bateria, nome da aplicação desenvolvida como estudo de caso deste trabalho, tem por finalidade geral o aumento do tempo de autonomia da bateria em computadores portáteis, através da redução do consumo de energia. Embora atualmente os notebooks já possuam recursos para aumentar a autonomia, ainda é muito frustrante ter o seu trabalho interrompido pela descarga da bateria.

Não há um perfil de usuário específico para o Bateria. Ele foi desenvolvido com a finalidade de possuir perfis de energia que fossem adequados para várias cargas de trabalho. Como, por exemplo, ser adequado a cargas de trabalho de escritório, para execução de aplicações de edição de texto e edição de planilhas, ou perfis domésticos, adequados à execução de aplicativos tocadores de mídia e navegação na Internet.

Diferentemente do arcabouço descrito no capítulo anterior, cujo objetivo incluía a criação de políticas de energia, a aplicação do estudo de caso possui um número fixo de políticas. A ideia é agregá-las na forma de perfis de energia, e reduzir a quantidade de tomadas de decisões necessárias para configurar a aplicação gerenciadora. Basta o usuário ativar o perfil correspondente à sua demanda e utilizar o notebook normalmente. Além de ter esses diferentes perfis disponíveis, o usuário também é capaz de mudar as configurações das políticas existentes e alterar também as configurações das políticas pré-existentes no sistema operacional do notebook.

Na Tabela 4.1 está sumarizado o conjunto de funcionalidades providas pela aplicação Bateria. Processador e a tela LCD são os dispositivos que mais gastam energia do sistema como um todo. O Windows 8 já fornece configurações de políticas para esses dispositivos, como limitar a frequência do processador entre dois patamares mínimo e máximo e esmaecimento da tela, respectivamente. Mas, optou-se por incluir uma política para o processador com base no trabalho de Melo [MLS+14], a fim de observar se os mesmos resultados obtidos na plataforma Linux se repetiriam no Windows. Optou-se também por incluir uma política para a Wi-Fi, pois notebooks tendem a fazer um uso maior dessa interface ao invés da Ethernet por se tratar de um dispositivo portátil.

Tabela 4.1: Funcionalidade da aplicação Bateria

Código	**Funcionalidade**
F1	Haver uma política para a interface Wi-Fi.
F2	Haver uma política para o processador.
F3	Exibição e Controle do Estado dos Dispositivos.
F4	Exibição e Modificação das Configurações das políticas.
F5	Exibição da Estimativa de Energia.
F6	Haver alguns perfis pré-determinados.

4.2 Desenvolvimento da Aplicação

Nesta seção é descrito como foi realizado o desenvolvimento da aplicação Bateria, a partir da implementação das funcionalidades descritas na Tabela 4.1.

A aplicação Bateria foi desenvolvida usando a plataforma .Net 4.5 , linguagem C# e o Microsoft Visual Studio 12 como IDE de desenvolvimento por todos os membros da equipe. O Bateria é parte do projeto de cooperação desenvolvido pelo Laboratório de Sistemas Embarcados e Computação Pervasiva da UFCG e a Positivo Informática. A aplicação gerenciadora foi instalada e avaliada em um notebook Positivo Premium N9325, que possui as características presentes na Tabela 4.2.

Tabela 4.2: Características do Notebook

Característica	Descrição
Sistema Operacional	Microsoft Windows 8

Processador	Intel i3 2.1 GHz
Memória	6 GB
Armazenamento	320 GB
Webcam	1.0 Megapixel
WLAN	IEEE 802.11 b/n/g
Tela	LCD 14” Widescreen
Ethernet	10/100/1000Mbps, Gigabit Ethernet
Bateria	Li-ion, 4 células 2200mAh

4.3 Implementação das Funcionalidades da Aplicação

Nesta seção é descrito como as funcionalidades do Bateria foram implementadas, de acordo com o arcabouço de software para gerenciamento de energia. Como este estudo de caso foi construído sobre a plataforma Windows, as interfaces do pacote de comunicação com o SO foram implementadas a partir de duas APIs fornecidas pelo Windows.

A primeira veio da plataforma .Net, que forneceu basicamente os procedimentos necessários para obtenção das taxas de utilização dos dispositivos e das configurações gerais do gerenciamento de energia do Windows, como por exemplo os tempos limites de esmaecimento e desligamento da tela. Ou seja, serviram para a implementação das interfaces *IPowerSettings* e *IPerfCounter*. A segunda API fornecida veio por meio das bibliotecas da instalação do Windows, como User32 e System32. Elas forneceram os procedimentos necessários para ativação e desativação dos dispositivos, listagem de dispositivos e obtenção dos estados de energia dos dispositivos. Ou seja, serviram para implementação da interface *IDeviceManager*.

Além da implementação das interfaces definidas no pacote de comunicação com o SO, a implementação das interfaces do componente GUI é fundamental para qualquer aplicação gerenciadora de bateria baseada no arcabouço proposto. Sendo assim, foi definida uma interface gráfica baseada no subsistema gráfico da Microsoft chamado WPF – *Windows Presentation Foundation*, possuindo uma janela composta por três abas onde se encontram distribuídos os controladores gráficos relacionados às funcionalidades descritas nas seções a seguir.

4.3.1 Política para a interface Wi-Fi

Essa funcionalidade tenta reduzir o consumo de energia do notebook através da desativação da interface Wi-Fi, quando detectado ociosidade. A interface Wi-Fi, quando conectada a uma rede, envia e recebe bytes a todo o momento. Uma das funções oferecidas pelas API do Windows é a obtenção do total de bytes recebidos e enviados desde a última chamada a essa mesma função. Sendo assim, foi possível definir uma política de controle da interface Wi-Fi baseada nessa função [LPC+13].

A partir da função citada anteriormente, foi possível implementar o método *WifiUsageRate()* da interface *IPerfCounter* para que retornasse a taxa total de bytes trocadas no intervalo durante duas chamadas consecutivas. Como não há uma função na API Windows que notifique os agentes interessados sobre uma alta taxa de bytes pela WiFi, foi necessário criar uma política periódica. Ela foi definida com um período de 6 minutos, cujo método *isReadyToApply()* verificava se a taxa total de bytes trocadas no período excedeu um limite de 1.447kB/s. Se sim, uma notificação era enviada ao componente GUI para que este exibisse uma notificação ao usuário se ele desejaria desativar o dispositivo WiFi, com o apoio de um *checkbox* onde ele marcaria se desejaria que a aplicação memorizasse essa decisão para aplicar em futuras ocasiões.

4.3.2 Política para o Processador

Essa funcionalidade tenta reduzir o consumo de energia do notebook através de uma estratégia de mudança de frequência do processador, baseada no trabalho de Melo [MLS+14]. Como no Windows não foi encontrada uma função nativa de notificação na mudança de utilização do processador, essa política foi implementada como sendo do tipo periódica a cada 2 segundos. Nesse caso, o método *isReadyToApply()* calcula a nova frequência máxima a ser imposta no processador, armazena numa variável e retorna *true* se a nova frequência é diferente do valor anterior, retornando *false* caso contrário. Para definir a frequência do processador, foi utilizada a configuração de energia do Windows que define dois patamares de utilização de frequência. O Windows então utiliza um algoritmo próprio de seleção de frequência que varia entre esses dois patamares mínimo e máximo. Assim foi implementado o método *applyControl()* dessa política. Na Figura 4.1 a seguir é descrito em detalhes do algoritmo de escolha do valor da próxima frequência, extraído do trabalho de Melo [MLS+14].

```
Dados: Frequências, Freq_Min, baixoIntervalo, médioIntervalo,
       altoIntervalo, altíssimoIntervalo.
Entrada: utilizaçãoDeProcessador
Saída: próximaFrequencia
Frequências ← {50, 66, 81, 100};
baixoIntervalo ← [0, 35%];
médioIntervalo ← (35%, 75%];
altoIntervalo ← (75%, 85%];
altíssimoIntervalo ← (85%, 100%];
if utilizaçãoDeProcessador ∈ baixoIntervalo then
    próximaFrequencia ← Frequências[0];
else if utilizaçãoDeProcessador ∈ médioIntervalo then
    próximaFrequencia ← Frequências[1];
else if utilizaçãoDeProcessador ∈ altoIntervalo then
    próximaFrequencia ← Frequências[2];
else utilizaçãoDeProcessador ∈ altíssimoIntervalo then
    próximaFrequencia ← Frequências[3];
```

Figura 4.1: Algoritmo de escolha da próxima frequência do processador.

4.3.3 Exibição e Controle do Estado dos Dispositivos

Essa funcionalidade diz respeito à possibilidade do usuário ter controle de quais dispositivos estão ligados para poder desligá-los a fim de economizar energia. Como se sabe, alguns dispositivos possuem estados de energia intermediários entre o ligado e desligado. Porém, esses estados não são acessíveis às aplicações em nível de usuário. Sendo assim, foi restringido ao uso apenas desses dois estados. Portanto, a interface gráfica do Bateria possui um controlador de modo interruptor (ou *switch*) para fazer tanto exibição quanto controle do estado dos dispositivos. A lista dos dispositivos controláveis pelo Bateria inclui: interface Wi-Fi, interface Ethernet, Webcam, driver de CD, modem 3G, dispositivo de som, leitor de cartões e dispositivo Bluetooth.

4.3.4 Exibição e Modificação das Configurações das Políticas

Essa funcionalidade diz respeito à possibilidade de alterar valores de configurações das políticas a fim de obter melhores resultados na economia de energia. Por exemplo, poder alterar o patamar a partir do qual é definido se a interface Wi-Fi se encontra em ociosidade,

como também alterar os diversos tempos limites das configurações de energia definidas pelo Windows. Para tanto, a interface gráfica possui uma aba onde é possível alterar esses valores e aplicá-los para que tomem efeito. A seguir, temos a lista das configurações modificáveis através da interface gráfica:

- Limiar mínimo em bytes da política de Wi-Fi.
- Tempo de esmaecimento da tela após ociosidade.
- Nível de brilho da tela ao esmaecer.
- Nível padrão do brilho da tela.
- Tempo de desligamento da tela após ociosidade.
- Tempo de hibernação após ociosidade.
- Tempo de suspensão após ociosidade.
- Ação ao fechar a tampa do notebook (hibernar, suspender, fazer nada).

4.3.5 Exibição da Estimativa de Energia

Como exposto anteriormente, o arcabouço dispõe de duas formas para estimativa de energia: uma orientada a taxas de utilização, e outra orientada a eventos. Porém, o modelo de estimativa de energia orientado a eventos se encaixa melhor em dispositivos de menor porte como os smartphones e celulares. Sendo assim, optou-se pelo modelo de potência baseado em utilização, reaproveitando o que o arcabouço oferece para a estimativa de potência, como os módulos que fazem a leitura dos arquivos de constantes.

A classe *PowerModel* possui uma thread que executa o método *estimatePowerUsage()* a cada 2 segundos, notificando os interessados, independentemente do valor dessa estimativa. A potência é estimada a partir de um modelo de regressão multivariada, em função das taxas de utilização do notebook. Dadas as variáveis descritas na Tabela 4.3, a equação que representa esse modelo é mostrada na Equação 4.1. Os símbolos C_X (onde x é uma letra) representam os coeficientes das respectivas variáveis X e o símbolo C_0 representa o coeficiente livre do modelo.

Tabela 4.3: Símbolos usados no modelo de potência.

Símbolo	Descrição
P_i	Utilização, em porcentagem do i-ésimo núcleo do processador.
F_i	Frequência do i-ésimo núcleo do processador, em Hz.

B	Porcentagem do brilho da tela.
EIn	Taxas de bytes de entrada da interface Ethernet em bytes.
EOut	Taxa de bytes de saída da interface Ethernet em bytes.
WIn	Taxas de bytes de entrada da interface Wi-Fi em bytes.
WOut	Taxa de bytes de saída da interface Wi-Fi em bytes.
D	Utilização do driver de CD/DVD: {0, 1}.
WC	Utilização do webcam: {0, 1}.
P	Estimativa de potência.

$$P = \sum P_i * C_{Pi} * F_i^3 + B * C_B + EIn * C_{EIn} + EOut * C_{EOut} + WIn * C_{Win} + WOut * C_{WOut} + D * C_D + WC * C_{WC} + C_0. \quad (4.1)$$

A interface gráfica do Bateria exibe a estimativa de potência em forma de um gráfico de linha, que chega a armazenar um histórico de 40 estimativas. Sendo assim, como a estimativa é feita a cada 2 segundos, a interface é capaz de mostrar o histórico da estimativa do consumo de potência nos últimos 80 segundos.

4.3.6 Perfis pré-determinados

Além de definir que há três perfis de energia, devem existir formas de ativá-los e editá-los na interface gráfica. Sendo assim, foi colocada uma caixa de combinação (*combobox*), por onde é possível listar os perfis existentes e escolher algum para ativar. Da mesma forma, foi colocada uma aba para edição das configurações dos perfis existentes. Foram definidos três perfis de energia, chamados Padrão, Multimídia e Avião. As configurações dos três perfis são mostradas a seguir na Tabela 4.4.

Tabela 4.4: Configuração dos perfis pré-determinados.

Nome do Perfil/ Configuração	**Padrão**	**Avião**	**Multimídia**
Estado das Interfaces de Rede	Ligado	Desligado	Desligado
Brilho da Tela	28%	42%	70%
Política de Processador	Ativada	Ativada	Ativada

Tempo de esmaecimento da tela	120s	150s	1800s
Brilho da tela ao esmaecer	0%	0%	50%
Tempo de desligamento da tela	300s	300s	3600s
Tempo de hibernação	1800s	900s	1200s
Tempo de suspensão	600s	600s	600s

4.4 Avaliação do Consumo de Energia do Notebook sob Gerenciamento da Aplicação

Para avaliação do consumo de energia do notebook sob o gerenciamento de energia da aplicação Bateria, foi utilizado um módulo de aquisição de dados chamado NI USB-6210, com a finalidade de medir a corrente e tensão fornecidas ao notebook. O módulo de aquisição realiza 100 amostragens por segundo. A potência é calculada pela multiplicação entre esses dois valores de corrente e tensão. A ideia é comparar se o Bateria consegue economizar mais energia que o gerenciamento de energia nativo do Windows 8, mais especificamente o perfil Equilibrado. Na Tabela 4.5, são descritas as características das configurações do perfil Equilibrado.

Tabela 4.5: Características do perfil Equilibrado

Nome do Perfil/ Configuração	**Equilibrado do Windows**
Estado das Interfaces de Rede	Ligado
Brilho da Tela	28%
Política de Processador	Ativada
Tempo de esmaecimento da tela	300s
Brilho da tela ao esmaecer	50%
Tempo de desligamento da tela	600s
Tempo de hibernação	Nunca
Tempo de suspensão	1800s

Foram definidas cinco cargas de trabalho diferentes para simular o uso do notebook. Essas cargas foram gravadas na forma de um script do programa Macro Recorder, que é capaz de reproduzir cliques do mouse e digitações no teclado, demorando cinco minutos para ser reproduzido, sendo um minuto de simulação para cada carga. Foram feitas 3 rodadas da execução desse script para cada perfil de energia, incluindo o perfil Equilibrado do Windows. Na listagem a seguir, tem-se a descrição da carga de trabalho desse script.

- [0:00, 1:00 min) período ocioso.
- [1:00, 2:00 min) Abrir uma música no programa Windows Media player até o final do período.
- [2:00, 3:00 min) Abrir um arquivo em formato pdf com o programa Adobe pdf reader e fechá-lo no fim do período.
- [3:00, 4:00 min) Abrir um vídeo no Windows Media Player no modo tela cheia e pará-lo ao final do período.
- [4:00, 5:00 min) período ocioso.

A média de consumo para cada perfil de energia é mostrada na Figura 4.2. Cada minuto da carga gerada pela execução do script gerou 6.000 amostras de energia. Sendo assim, a execução do script todo gera 30.000 amostras de potência durante os cinco minutos para cada perfil. Como foram 3 rodadas de execução para cada perfil, totalizou 9.000 amostras de potência para cada perfil durante todo o experimento. Cada barra representa o consumo de potência médio das 90.000 amostras para cada perfil de energia do Bateria, com exceção da última barra em vermelho que representa o perfil de gerenciamento do Windows. É possível perceber visualmente uma pequena redução entre o perfil Equilibrado e o perfil Multimídia, mas uma grande diferença entre o perfil Equilibrado e o perfil Avião. Como mostrado na seção anterior, o perfil Avião possui uma estratégia mais agressiva pois desabilita todas as interfaces de rede e reduz o brilho da tela consideravelmente, sendo assim, era esperado que ele tivesse uma maior vantagem entre os outros perfis.

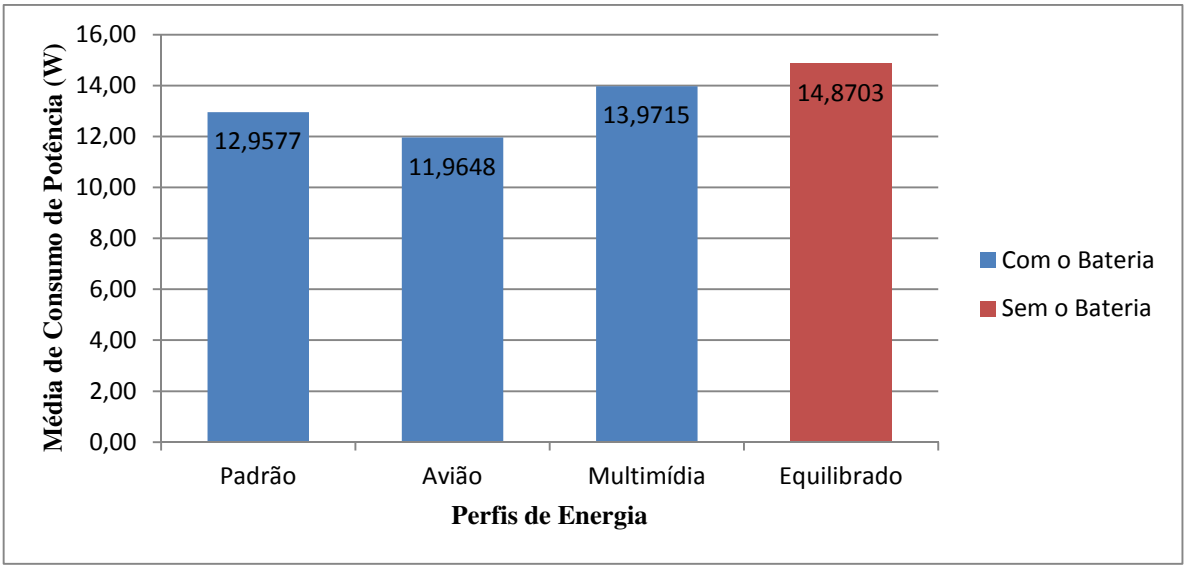

Figura 4.2: Média de Potência entre os perfis de Energia.

Além dos experimentos de potência, foram realizados também testes de descarga total da bateria. O objetivo desses testes foi comparar os perfis de energia Padrão e Windows Equilibrado para testar a eficiência das políticas de mecanismos ociosos de sugestão de desligamento de dispositivo, como é o caso da política de Wi-Fi. Os testes foram realizados no mesmo notebook descrito anteriormente. Foram realizados testes manuais, sem a utilização de scripts, pois as notificações de ociosidade são mais difíceis de prever quando aparecerão, inviabilizando a automatização do teste.

O procedimento de teste de descarga foi realizado da seguinte maneira: ao iniciar o teste, abrir uma música no Windows Media Player deixando em tela cheia, no volume mudo e em repetição por 20 minutos. Logo após, abrir o browser Google Chrome, que tem como página inicial o link http://www.youtube.com/watch?v=VASeeu8xoqw , deixando aberto por 15 minutos. Após isso, abrir um vídeo no Media Player e deixá-lo executando em repetição por 20 minutos. Depois, abrir um arquivo pdf por 10 min no Adobe Reader enquanto o vídeo continua sendo executado no mudo, deixando a operação seguir por 20 minutos. Após isso, fechar todos os programas e deixar o computador ocioso até sua descarga completa. Na Figura 4.3 são ilustrados os resultados desses testes. O perfil Padrão do Bateria aumentou o tempo de descarga total em relação ao perfil Equilibrado em 15 minutos, cerca de 12%.

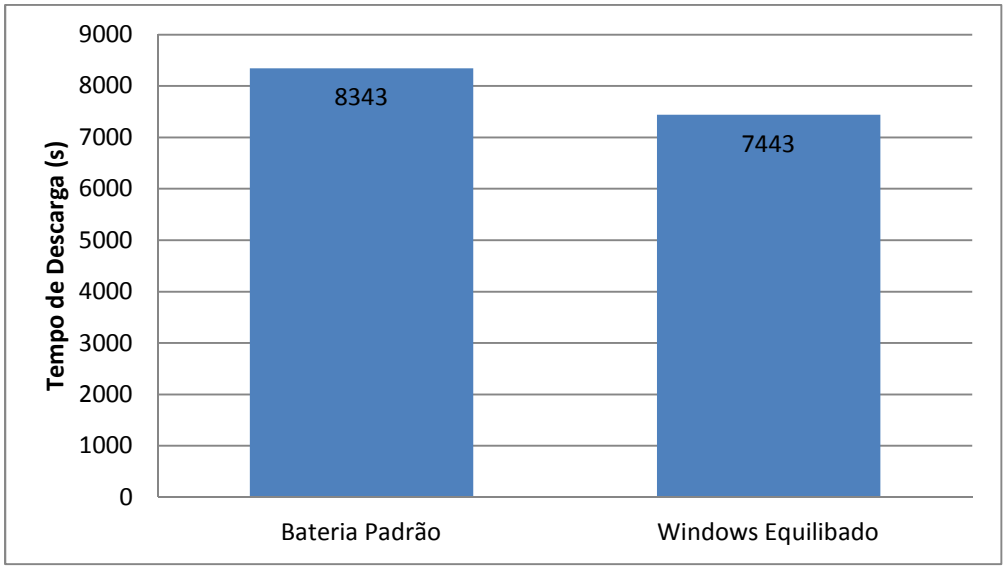

Figura 4.3: Tempo de descarga total entre perfis.

Capítulo 5

Trabalhos Relacionados

Neste capítulo são apresentados os trabalhos relacionados ao arcabouço de software para desenvolvimento de aplicações gerenciadoras de energia. Dentre os trabalhos relacionados, destacam-se as soluções que desenvolveram um esquema arquitetural para os componentes responsáveis pelo gerenciamento de energia, e as soluções que desenvolveram uma aplicação gerenciadora de energia como proposta principal. As soluções dos trabalhos relacionados apresentados neste capítulo também podem ser divididas em soluções ao nível do sistema operacional, ao nível de aplicativo e ao nível híbrido, que envolve as duas abordagens.

Das buscas realizadas, foram selecionados oito trabalhos que abordam de alguma maneira pelo menos um dos seguintes problemas:

- Modelagem de políticas de energia e Gerenciamento Dinâmico de Energia.
- Definição de uma interface padronizada para obtenção dos dados de utilização dos subsistemas e controle desses subsistemas.
- Incorporação de um modelo de estimativa de energia na modelagem da solução.
- Independência de plataforma, de forma que a solução pudesse ser reproduzida para outras plataformas.
- Salvar o estado das configurações de gerenciamento de energia em arquivo.

5.1 Arquiteturas e Arcabouços em Gerenciamento de Energia

5.1.1 HAPPI

Este trabalho parte da observação que uma política de energia que é melhor que outra, nem sempre é melhor em qualquer circunstância. Sendo assim, é muito difícil ou até mesmo impossível desenvolver a “melhor” política para todos os computadores. Para escolher a melhor política em tempo de execução sem a intervenção de um usuário ou administrador, foi desenvolvido o arcabouço de software chamado HAPPI (*Homogeneous Architecture for*

Power Policy Integration) [PL09]. A arquitetura proposta é portável em plataformas que rodem Linux. O HAPPI especifica requisitos em comum para as políticas e fornece uma interface para simplificar a implementação delas em um Sistema Operacional.

Para implementar uma política de energia no HAPPI, o desenvolvedor da política deve fornecer: (1) uma função que prediz a ociosidade e controla os estados de energia de um dispositivo e (2), uma função de estimação que recebe como entrada um histórico das requisições ao dispositivo, determina as ações que a função de controle deve tomar e retorna o consumo de energia e atrazos de acesso para as ações de controle. Este trabalho não fornece detalhes ao nível de classes e interfaces, apenas fornece diagramas de alto nível de como se estrutura a arquitetura.

Foram usadas medições físicas para demonstrar que a abordagem proposta pode economizar de forma comparável à melhor política individual, sem o conhecimento a priori de hardware ou carga de trabalho. A técnica de seleção automática de políticas é aplicada e avaliada em desktops e servidores. Os experimentos indicam que o HAPPI alcança economias de energia de até 4% da melhor política pré-determinada e evita selecionar políticas que desperdiçam energia. Porém, economias de energia adicionais além da melhor política individual são difíceis, devido à função de estimativa de energia que é baseada em histórico de acessos.

5.1.2 PASA

A proposta deste trabalho parte da premissa que o Sistema Operacional sozinho não consegue fazer gerenciamento dinâmico de energia de forma eficiente. É preciso que as aplicações troquem informações com o SO para que ele consiga fazer as avaliações custo-benefício necessário. Sendo assim, foi proposto o PASA [CRM02], que trata de uma arquitetura baseada em camadas para sistemas embarcados com wireless. A arquitetura é definida juntando parte de um kernel de um Sistema Operacional de Tempo Real (SOTR) com uma API para ser usada pelas aplicações. O foco das estratégias de gerenciamento de energia foi no escalonador de processos desse SOTR.

Os requisitos dessa arquitetura são basicamente a capacidade do SO monitorar os recursos de hardware, restrições e deadlines (de performance e energia) dos processos e também de prover uma interface para troca de informações entre SO e aplicações. A arquitetura é composta de duas camadas de software e uma camada de kernel. A primeira camada contem funções a serem inseridas nas aplicações para que ela informe ao SO quando

está executando um trecho importante e os deadlines de tempo dessa execução. Também define funções de *callback* para que o SO responda as predições de tempo de execução. A segunda camada se posiciona no mesmo nível do SO e contém funções para alterar as políticas de processos e hardware. A terceira camada possui funções de monitoramento e controle dos dispositivos de hardware, a fim de melhorar seu monitoramento e controle.

A fim de mostrar a sua utilidade, o PASA foi incorporada em sistema operacional eCos rodando em um sistema de tensão variável baseado em um processador *XScale*. Foram implementadas quatro algoritmos diferentes de DVS, desde um simples esquema de desligamento até um algoritmo DVS dinâmico preditivo e adaptativo, sendo coletados dados para mostrar a economia de energia em um sistema operacional integrando o PASA. Os resultados mostram um ganho de até 38% quando comparado para a execução sem qualquer gerenciamento de energia incorporado.

5.1.3 Uma Arquitetura de Software para Gerenciamento de Energia em SOTRs Usando Orientação a Objeto

Neste trabalho é proposto um conceito abstrato sobre a efetuação de gerenciamento de energia em SOTRs (Sistema Operacional de Tempo Real) embarcados [ARP06]. A motivação é ter uma interface entre aplicações e SOTRs. As aplicações precisam informar o SO sobre seus requisitos de energia e quais dispositivos querem usar. O SO precisa informa às aplicações sobre os estados dos dispositivos, como o nível de bateria, para que eles tomem as decisões de gerenciamento. É demonstrado que o gerenciamento de energia ao nível de sistema pode ser feito usando Orientação a Objeto.

São modelados três componentes principais responsáveis pelo gerenciamento de energia no SOTR: *DeviceDrivePolicyManager*, *ApplicationPowerManager* e *PolicyManager*. São fornecidas as interfaces, diagramas de interação, diagramas de sequência e diagramas de classes para modelagens desses componentes, utilizando UML como linguagem. Essa representação OO dos componentes necessários para o gerenciamento de energia ajuda a elucidar a interação do gerenciador de energia em conjunto com as aplicações, dispositivos e processador com o desenvolvedor das aplicações.

Se o sistema operacional e os drivers de dispositivos forem desenvolvidos de acordo com a arquitetura proposta nesse trabalho, a tarefa de gerenciar energia a partir de aplicações pode ser bastante simplificada, resultando em maior autonomia de bateria em alguns casos. A

interface orientada a objeto para gerenciamento de energia também simplificaria testar diferentes cenários de consumo que não foram testados anteriormente.

5.1.4 Um Arcabouço Ciente de Contexto para Dispositivos Embarcados

Este trabalho parte da problemática de reconhecer que o gerenciamento de energia não deve ser apenas feito ao nível de sistema e que a tarefa de gerenciamento de energia está ficando tão complexa quanto os dispositivos móveis e portáteis disponíveis no mercado. Sendo assim, é proposto um arcabouço para gerenciamento de energia ciente de contexto para dispositivos embarcados [YBK+12].

O arcabouço é dividido em duas definições de políticas. Uma é a definição de contexto e a outra é a definição de controle. A primeira definição basicamente envolve as condições que disparam uma ação de controle, ou seja, as condições de contexto. Por exemplo, são desenvolvidas condições que detectam se o usuário está caminhando, correndo, olhando para a tela do dispositivo, ou até em um lugar fechado. Elas se relacionam usando o padrão *Composite*. A Segunda definição envolve as ações a serem tomadas, por exemplo, reduzir o brilho em 10%. Ambas essas definições são feitas em arquivos XML.

O arcabouço foi implementado sobre um SOTRs T-kernel. A implementação foi dividida em dois módulos. Um é o Power Manager, responsável por aplicar as ações. O outro é o Context Manager, responsável por calcular as restrições a partir de informações que obtem dos dispositivos. Os componentes de software se comunicam entre si usando o arcabouço D-Bus.

5.1.5 Um Arcabouço para Gerenciamento de Sistemas Multinúcleos

Neste trabalho, é proposto um arcabouço para gerenciamento termal e energético para sistemas multinúcleos [AHC09]. O arcabouço é composto de um *daemon* (processo em background) de kernel, um *daemon* de usuário e de um módulo gerenciador de políticas de energia. Os *daemons* de kernel e usuário monitoram a execução dos programas. O gerenciador de políticas controla o estado dos núcleos de acordo com as informações coletadas pelos *daemons*.

Uma política é definida na forma de um módulo do Linux. Ela possui a capacidade de receber tanto informações de utilização quanto de *system calls*. Para adicionar uma nova

política, é preciso definir seu algoritmo de condições e critérios para sua aplicação, e uma prioridade.

O arcabouço foi implementado em duas plataformas diferentes: Intel Centrino Duo e ARM-11 MPCore. Os resultados experimentais mostraram que o arcabouço foi capaz de reduzir o consumo de energia e reduzir a temperatura dos processadores. Os experimentos foram conduzidos utilizando um programa de reprodução de multimídia como benchmark, pois havia o interesse em realizar várias operações de leitura na memória e instruções de calculo nos decodificadores de vídeo.

5.1.6 UPMF

Trata-se de um trabalho simples cujo objetivo é a criação de um arcabouço capaz de manipular as várias aplicações multimídia de um único dispositivo, chamado *Unified Power Management Framework* – UPMF [ITB+07]. O arcabouço fica disposto entre as aplicações e o sistema operacional. Este trabalho não modela as partes do arcabouço, mas dá detalhes gráficos arquiteturais de como as partes se relacionam, incluindo um diagrama de fluxo de controle.

O arcabouço é divido em quatro partes e são descritos os requisitos de cada uma. A primeira parte são classes de dispositivos. Essa parte, por sua vez, classifica os subsistemas em quatro níveis de acordo com seus padrões de uso. A segunda parte são as classes de configurações, que é responsável por atribuir configurações às classes definidas pela primeira parte. A terceira parte trata dos estados de dispositivos, que modela situações como o subsistema esteja ligado ou não e o nível de brilho da tela. Por fim, a quarta parte são estratégias de gerenciamento, responsável por gerenciar os requisitos de energia e desempenho das aplicações.

5.1.7 Uma abordagem Ciente de Aplicação

A abordagem desse trabalho é centrada na análise do comportamento das aplicações, ao invés da análise dos recursos de hardware. Trata-se um arcabouço ciente de aplicação, que se baseia na informação das necessidades de recursos das aplicações e em técnicas de gerenciamento dinâmico de energia focadas na interface wireless a fim de reduzir o consumo de energia em tempo de execução [LL09].

De um ponto de vista operacional, aplicações são orientadas a trabalhar com dados e invocação de serviços. Seria interessante se uma estratégia de energia pudesse saber quais os serviços disponíveis à aplicação, e de preferência, saber quanto tempo dura para executar um serviço ou, pelo menos, quando o resultado do serviço deve ser obtido. Baseado nesses dois conceitos foi desenvolvida uma linguagem baseada em XML para modelar o comportamento desejado e esperado de uma aplicação. Baseado nessa informação de comportamento, o arcabouço será capaz de decidir não apenas quando se conectar aos serviços, mas também decidir qual dado dever ser enviado ou recebido.

O arcabouço foi desenvolvido seguindo a arquitetura cliente-servidor. O lado cliente do arcabouço é composto por cinco componentes, responsáveis por sincronização, gerenciamento de políticas, invocação de serviços, notificação de eventos e migração de contexto, respectivamente. O módulo de gerenciamento de políticas é o responsável por decidir quando ativar um processo de comunicação (usando o componente de sincronização e/ou o componente de invocação de serviços) baseado na informação colhida dos outros componentes. Além do arcabouço, foi proposta uma nova técnica de gerenciamento de energia chamada CIST (*Consumption Improviment by Stuffing Time*) que baseia-se no tempo de comunicação e ociosidade das requisições.

Para avaliar o arcabouço, foi desenvolvido um protótipo baseado no arcabouço feito na linguagem de programação C# com o uso da plataforma Microsoft .Net Framework. Os experimentos foram conduzidos em um PDA 2K Pocket PC, incluindo cinco diferentes cenários para três variações do protótipo. Foi verificado que a variação do protótipo que incluía a técnica CIST superou as outras variações (que não a continham) em todos os cinco cenários.

5.1.8 Um Middleware Ciente de Energia

É proposto um programa que observa informações do contexto do celular, como intensidade do sinal e níveis de bateria, entre outras, para tomar medidas de controle ou notificar o usuário para que ele tome uma decisão [XKY09]. Trata-se de um middleware, e como tal, é a instanciação de uma arquitetura. Ou seja, não se trata de uma solução extensível. Possui uma arquitetura baseada em componentes, dividida em seis módulos: power estimator, processing engine, Gerenciador de Políticas, resource manager, application classifier e messaging. Apresenta um diagrama de classes para o resource Manager.

Inova ao criar uma forma de classificar aplicações, baseado em aprendizado de máquina, usando o algoritmo de Bayes básico. Classifica as aplicações em três tipos, baseando-se nos padrões de acesso à memória, utilização da rede e utilização de CPU. A avaliação é feita de forma simples: colocaram um vídeo de youtube em execução, em seguida é comparada a utilização de bateria com e sem o middleware rodando.

Capítulo 6

Considerações Finais

Neste trabalho é apresentado um arcabouço de software para o desenvolvimento de aplicações de gerenciamento de energia, com um estudo de caso para validação do arcabouço. Neste capítulo são descritas as contribuições e os possíveis trabalhos decorrentes do desenvolvimento desse arcabouço.

6.1 Contribuições

O arcabouço de software para o desenvolvimento de aplicações de gerenciamento de energia proposto neste trabalho foi projetado com o objetivo de resolver diversos problemas. Esses problemas podem ser sumarizados como sendo a criação de perfis de energia, estimação do consumo de potência da máquina subjacente, reutilização das configurações de gerenciamento de energia do SO subjacente e possuir elementos gráficos acessíveis de controle dos dispositivos. Para isso, foi adotada uma arquitetura baseada em componentes dividida em três grandes componentes, subdivididos em outros subcomponentes.

Foi demonstrado ainda, por meio de um estudo de caso, que o desenvolvimento de aplicações gerenciadoras de energia com o arcabouço proposto é possível. Neste estudo de caso, foi desenvolvida uma aplicação gerenciadora de energia composta por duas políticas de energia, com reaproveitamento das configurações de energia do sistema operacional, organizando-as em três perfis de energia. Foram realizados experimentos de medição de potência em um computador notebook entre os três perfis e também testes de descarga de bateria, verificando-se que foi possível reduzir o consumo de energia em alguns cenários.

A partir dos trabalhos relacionados, é possível verificar que o arcabouço proposto contempla vários aspectos das soluções em arcabouço em gerenciamento de energia. Como por exemplo, a representação do estado das configurações de energia em arquivo, ser aplicável a mais de um nível na hierarquia de gerenciamento e possuir uma modelagem de políticas que permite a extensão para diferentes estratégias. Além disso, o arcabouço proposto

consegue ser voltado a várias plataformas de hardware, enquanto que os trabalhos relacionados são direcionados mais a plataformas móveis ou dispositivos portáteis.

6.2 Trabalhos Futuros

O arcabouço proposto é uma solução que facilita o desenvolvimento de aplicações gerentes de energia. Porém, para melhorar a avaliação do desempenho das políticas em tempo de execução, seria interessante se o arcabouço oferecesse alguma linguagem ou sintaxe de descrição de políticas, para que estas pudessem ser definidas em um arquivo. No momento, as políticas só podem ser alteradas no momento do desenvolvimento da programação, e a forma para testar as diferentes políticas se dá por meio dos perfis de energia. Mecanismos que facilitem a construção de soluções utilizando o arcabouço proposto são os principais trabalhos futuros planejados. Através dessa sintaxe, seria possível modificar as estratégias de decisão das políticas no momento da execução do gerenciador, e não só apenas modificar as variáveis nas quais essas decisões se baseiam.

Referências Bibliográficas

[AHC09] Y. Ahn, Y.-S. Hwang, K.-S. Chung. "Flexible framework for dynamic management of multi-core systems". International SoC Design Conference (ISOCC), 2009.

[ARP06] A. Agarwal, S. Rajput, AS. Pandya. "Power Management System for Embedded RTOS: An Object Oriented Approach". Canadian Conference on Electrical and Computer Engineering, 2006.

[BAC02] F. Bachmann et al. Volume II: "Technical Concepts of Component-Based Sostware Engineering - 2a Edição. 2002.

[BBC+12] R. Bertran, Y. Becerra, D. Carrera, V. Beltran, M. Gonza lez, X. Martorell, N. Navarro, J. Torres, E. Ayguade, "Energy Accounting for Shared Virtualized Environments Under DVFS Using PMC-Based Modelo de Potências" Future Gener. Comput. Syst., vol. 28, no. 2, pp. 457-468, Fev. 2012.

[BBM00] L. Benini, A. Bogliolo, G. De Micheli, “A survey of design techniques for system-level dynamic power management,” IEEE Trans. Very Large Scale Integr. Syst., vol. 8, no. 3, pp. 299–316, Jun. 2000.

[BBS00] D. Brooks, P.Bose, S.Schuster, H.Jacobson, P.Kudva, A.Buyuktosunoglu, J.Wellman, V.Zyuban, M.Gupta and P.Cook, “Power Aware Microarchitecture: Design and Modeling Challenges for Next-Generation Microprocessors”, IEEE Micron, Dez. 2000, p26-44.

[BRO00] A. W. Brown, K. Wallnau. "The Current State of CBSE". IEEE Software, v.15, n.5, p.37-46. Set. 1998.

[BRO97] A. Brown, W. Short, Keith. "on Components and Objects: The Foundation of Component-Based Development". International Symposium on Assessment of Sofware Tools and Technologies, Maio, 1997.

[CRM02] C. Pereira , R. Gupta , M. Srivastava. “PASA: A Software Architecture For Building Power Aware Embedded Systems”. Proceedings of the IEEE CAS Workshop on Wireless Communication and Networking. 2002.

[CSB92] A. Chandrakasan, S. Sheng, R. Brodersen, “Low-Power CMOS Digital Design”, IEEE J. of Solid-State Circuit, vol. 27, no. 4, (1992), pp. 473-484.

[DKB95] F. Douglis, P. Krishnan, B. Bershad. "Adaptive Disk Spin-Down Policies for Mobile Computers". Computing Systems, volume 8, pages 381–413, 1995.

[ERK+06] D. Economou, S. Rivoire, C. Kozyrakis, P. Ranganathan. ‘‘Full-System Power Analysis and Modeling for Server Environments,’’ in Proc. 2nd Workshop MoBS, Boston, MA, USA, 2006, pp. 70-77.

[FWB07] X. Fan, W.-D. Weber, and L.A. Barroso. "Power Provisioning for a Warehouse-Sized Computer". ACM SIGARCH Comput. Maio, 2007. Archit. 35(2), pp. 13-23.

[GBK+12] V. Gupta, P. Brett, D. Koufaty, D. Reddy and S. Hahn. “The Forgotten ‘Uncore’: On the Energy-Efficiency of Heterogeneous Cores”, 2012 USENIX Annual Technical Conference, Short Paper, (2012).

[GCW95] K.Govil, E.Chan, H.Wasserman, “Comparing Algorithms for Dynamic Speed-Setting of a Low-Power CPU”, MobiCom 1995.

[GLF+00] D. Grunwald, P. Levis, K. Farkas, C.B. Morrey III, M. Neufald, “Policies for Dynamic Clock Scheduling,” Proc. Fourth Usenix Symp. Operating Systems Design and Implementation (OSDI ’00), pp. 73-86, Out. 2000.

[HDC+05] T. Heath, B. Diniz, E.V. Carrera, W.M., Jr., and R. Bianchini, ‘‘Energy Conservation in Heterogeneous Server Clusters,’’ Proc. 10th ACM SIGPLAN Symp. PPoPP, 2005, pp. 186-195.

[HKQ+99] I. Hong, D. Kirovsky, G. Qu, M. Potkonjak and M. Srivastava, “Power Optimization of Variable Voltage Core-based Systems”, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 18, no. 12, (1999), pp. 1702-1714.

[HPS86] Hoel, P. G., Port, S. C. e Stone, C. J., Introduction to Stochastic Processes, Segunda Edição. Waveland Press,1986.

[HW97] C.-H. Hwang, A. C. Wu. A Predictive System Shutdown Method for Energy Saving of Event-Driven Computation. International Conference on Computer-Aided Design, pag. 28–32, 1997.

[ITB+07] A. Iyenggar,A. Tripathi, Ajit Basarur, I. Roy. "Unified Power Management Framework for Portable Media Devices". IEEE International Conference on Portable Information Devices, 2007.

[KMM94] A. Karlin, M. Manesse, L. McGeoch, S. Owicki, "Competitive Randomized Algorithms for Nonuniform Problems", Algorithmica, pp. 542-571, 1994.

[LBM00] Y.H. Lu, L. Benini, G. De Micheli, “Operating System directed power reduction,” in International Symposium on Low Power Electronics and Design, 2000, pp. 37–42.

[LCS+00] Y.H. Lu, E.Y. Chung, T. Simunic, L. Benini, G. De Micheli. "Quantitative Comparison of Power Management Algorithms". Design Automation and Test in Europe, pages 20–26, 2000.

[LHH02] Z. Lu, J. Hein, M. Humphrey, M. Stan, J. Lach, K. Skadron, “Control-Theoretic Dynamic Frequency and Voltage Scaling for Multimedia Workloads,” Proc. Third ACM/IEEE Int’l Conf. Compilers, Architecture, and Synthesis for Embedded Systems (CASE’01), pp. 156-163, Oct. 2002.

[LJ06] P. Langen, B. Juurlink, “Leakage-Aware Multiprocessor Scheduling for Low Power”, 20th International Parallel and Distributed Processing Symposium, (2006), pp. 80.

[LL09] A. B. Lago, I. Larizgoitia. "An application-aware approach to efficient power management in mobile devices". In Proceedings of the Fourth International ICST Conference on COMmunication System softWAre and middlewaRE, 2009.

[LPC+13] S. O. D. Luiz, A. Perkusich, B. M. J. Cruz, B. H. M. Neves, G. M. da S. Araújo. “Optimization of timeout-based power management policies for network interfaces,” IEEE Trans. Consum. Electron. , vol. 59, no. 1, pp. 101–106, 2013.

[LPL10] S. O. D. Luiz, A. Perkusich, A. M. N. Lima. "Multisize sliding window in workload estimation for dynamic power management". IEEE Transactions on Computers, 59(12):1625–1639, 2010.

[LSC04] X. Liu, P. Shenoy, M. Corner, “Chameleon: Application-Controlled Power Management with Performance Isolation,” Technical Report 04-26, Univ. of Massachusets, Amherst, 2004.

[MBS12] D. Milam, L. Bartram, M. Seif El-Nasr. "Design patterns of focused attention". In Proceedings of the First Workshop on Design Patterns in Games, 2012.

[MLS+14] T. Melo, S. O. D. Luiz, J. Silva, B. Neves. " Experimental Platform for Power Measurements of Computer Systems". IEEE Fourth International Conference on Consumer Electronics - Berlin. 2014.

[PL09] N. Pettis, Y.H. Lu. “A Homogeneous Architecture for Power Policy Integration in Operating Systems”. In IEEE Transactions on Computers, 2009.

[QH01] G. Quan, X. Hu, “Energy Efficient Fixed-Priority Scheduling for Real-Time Systems on Variable Voltage Processors”, Design Automation Conference, 2001.

[QP99] Q. Qiu, M. Pedram. Dynamic Power Management Based on Continuous-Time Markov Decision Processes. In Design Automation Conference, pages 555–561, 1999.

[SAM97] J. Sametinger. "Sofware Engineering with Reusable Components". New York: Springer, 1977.

[SMB99] T. Simunic, G. De Micheli, L. Benini. “Event-driven power management of portable systems”, in ISSS, 1999, pp. 18–23.00.

[SPEC] SPEC Power and Performance Benchmark Methodolody V2.1, Standard Performance and Evaluation Corporation (SPEC), 2011.

[SSI] EPS12V Power Supply Design Guide (SSI), 2nd ed., Intel Corp.,Santa Clara, CA, USA, 2004.

[SZY99] C. Szyperki . "Componente Software: beyond object-oriented programming". Harlow: Addison-Walesy, 1999.

[WBD94] M.Weiser, B.Welch, A.Demers, S.Shenker, “Scheduling for Reduced CPU Energy”, Usenix Association, Nov. 1994.

[XKY09] Y. Xiao, R. S. Kalyanaraman, A. Ylä-Jääski. "Middleware for energy-awareness in mobile devices". In Proceedings of the Fourth International ICST Conference on COMmunication System softWAre and middlewaRE. 2009.

[YBK+12] T. Yashiro, T. Ban, S. Kobayashi. "A framework for context-aware power management on embedded devices". In IEEE 1st Global Conference on Consumer Electronics (GCCE), 2012.