UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Geovani Ricardo Wiedenhoft

Gestão de Energia para Sistemas Embarcados de Tempo Real Usando Técnicas da Computação Imprecisa

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de mestre em Ciência da Computação.

Prof. Dr. Antônio Augusto Medeiros Fröhlich
(Orientador)

Florianópolis, Setembro de 2008

Gestão de Energia para Sistemas Embarcados de Tempo Real Usando Técnicas da Computação Imprecisa

Geovani Ricardo Wiedenhoft

Esta Dissertação foi julgada adequada para a obtenção do título de mestre em Ciência da Computação, área de concentração Sistemas de Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

Prof. Dr. Frank Augusto Siqueira
(Coordenador)

Banca Examinadora

Prof. Dr. Antônio Augusto Medeiros Fröhlich
(Orientador)

Prof. Dr. Fabiano Passuelo Hessel

Prof. Dr. Rômulo Silva de Oliveira

Prof. Dr. Lau Cheuk Lung

Para minha mãe e meus irmãos

Agradecimentos

Agradeço ao Professor Antônio Augusto Fröhlich pelo apoio, dedicação e auxílio durante o tempo do meu mestrado, o qual permaneci em um excelente ambiente de pesquisa – o Laboratório de Integração Software e Hardware (LISHA). Muito obrigado aos meus colegas do LISHA pelas valiosas discussões técnicas, científicas e pelas risadas de cada dia que tornaram o trabalho uma diversão.

Agradeço à minha mãe, Iria, ao meus irmãos, Gilnei e Gilmar, pelo amor e apoio constante. Muito obrigado também aos meus amigos Gelson e Lucas que considero como irmãos e suas respectivas famílias que são importantes na minha vida pessoal e profissional. Finalmente, agradeço à minha amada, que reencontrei depois de tantos anos, por tornar cada dia o dia mais feliz da minha vida.

Este trabalho contou com apoio material da CAPES (bolsa de mestrado), Fundo de Incentivo à Pesquisa/UFSC, FINEP, Atmel, Inc. , Pró-Reitoria de Pós-Graduação/UFSC e Diretoria do Centro Tecnológico/UFSC. Agradeço a todas instituições pela oportunidade.

Resumo

As técnicas de gerência de energia, como DVS e hibernação de recursos, são essenciais para a modelagem de sistemas embarcados capazes de gerenciar seus próprios consumos de energia; contudo, elas, freqüentemente, ocasionam sobrecusto e/ou latência. Esses fatores não podem ser ignorados em sistemas que buscam garantir as restrições de tempo real e/ou de QoS. No entanto, frisamos que não é suficiente garantir as restrições de tempo real e/ou de QoS se, fazendo isso, o sistema consome o orçamento de energia disponível e não é capaz de completar suas tarefas críticas.

Neste trabalho, utilizamos o tempo de duração da bateria esperado como um parâmetro de QoS – QoS em termos de energia. O objetivo não é apenas economizar energia, mas aumentar a utilidade da aplicação, garantindo o tempo de duração da bateria desejado e, ainda, preservando os *prazos* das tarefas *hard real-time*. O controle dos níveis de QoS foi inspirado na Computação Imprecisa que divide cada tarefa em parte obrigatória e em parte opcional. Dado um conjunto de tarefas, seus tempos de execução e seus consumos de energia, disponibilizamos equações para verificar a sua escalonabilidade em tempo de projeto. Em tempo de execução, um escalonador preemptivo para tarefas imprecisas descarta partes opcionais quando existe a possibilidade dos *prazos* e/ou do tempo de duração do sistema requerido serem perdidos. Nos períodos ociosos criados, o escalonador invoca um gerente de energia oportunista.

Implementamos um protótipo usando o EPOS para fornecer suporte ao algoritmo de escalonamento com tarefas imprecisas e execuções condicionais aos parâmetros desejados. Com base em um estudo de caso, mostramos que esse mecanismo permite que as aplicações alcancem seus compromissos entre QoS e consumo de energia.

Abstract

Power management techniques such as DVS and resources hibernation are essential to the design of a power-aware embedded system. However, their use often incurs in overhead and/or latency. These factors cannot be ignored if the system have to respect real-time or QoS constraints. On the other hand, it is not enough to guarantee real-time or QoS constraints if, while doing so, the system exhausts its energy budget (e.g. discharges its batteries) and is unable to complete its tasks.

In this work, we use an application's energy budget, that is, its expected battery lifetime, as a QoS parameter – QoS in terms of energy. Our goal is not to only save energy, but also to improve the application utility, ensuring that the application's energy budget and the deadlines of hard real-time tasks are met. QoS level control of applications was inspired by Imprecise Computation, which divides each task into two sub-tasks, one mandatory, and one optional. We provide equations to analyse the schedulability of a set of tasks in terms of deadlines and energy budget at project time. At runtime, a preemptive scheduler for imprecise tasks prevents the optional subtasks of being executed whenever there is the possibility of a deadline loss or battery exhaustion. In the idle period, the scheduler executes the opportunistic power manager.

We implemented a prototype using the EPOS operating system in order to support our scheduling algorithm with imprecise tasks and executions conditionated to time and energy parameters. Based on a case study, we show that this mechanism allows applications to achieve their compromise between quality-of-service and energy consumption.

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

Tennenhouse [Ten00] identificou que apenas 2% do total dos processadores fabricados no mundo estão em sistemas de propósito geral, que são sistemas projetados para executar toda e qualquer aplicação que o usuário deseja, como, por exemplo, desktops e notebooks. Entretanto, a grande maioria dos processadores, ou seja, os 98% restantes, está em sistemas embarcados, que são plataformas computacionais dedicadas a executar um determinado conjunto de tarefas com objetivos específicos. Exemplos de sistemas embarcados incluem sistemas de controle de equipamentos, ipods, faxes, controle de freios ABS, etc. Normalmente eles monitoram e/ou controlam o ambiente nos quais estão inseridos, e devem responder aos estímulos externos. Além disso, de acordo com Tennenhouse, 97% dos processadores fabricados possuem arquiteturas inferiores a 32 bits, mais simples que as atuais de 64 bits.

Normalmente, os sistemas embarcados apresentam rigorosas limitações em termos da capacidade de processamento e de memória, pois, por muitas vezes, os usuários exigem baixo custo e tamanho reduzido desses sistemas. Além disso, muitos deles são alimentados por baterias com uma carga de energia limitada, devido aos requisitos de caráter móvel de suas aplicações. Com isso, o tempo de duração da bateria é um fator determinante para a disponibilidade desses sistemas, e, em muitos casos, a interferência humana para a troca da bateria pode ser inaceitável por questões de tempo, custo ou acesso ao sistema. Um exemplo poderia ser a troca das baterias em uma grande

rede de sensores sem fio com centenas ou milhares de dispositivos que monitoram a temperatura e a umidade de uma plantação com diversos hectares de extensão (aplicação de agricultura de precisão). Outro exemplo, poderia ser o de sistemas embarcados que monitoram as migrações de animais, pois nesses casos, os equipamentos estão em constante movimentação, não sendo prática a troca de baterias. Todas essas limitações demandam um gerenciamento eficiente do consumo de energia e que não ocasione interferência significativa na execução da aplicação.

1.1 Motivação

Técnicas de gerenciamento de energia permitem que certos componentes do sistema sejam ligados e desligados dinamicamente ou que operem em diferentes níveis de consumo de energia ao longo do tempo. Com isso, elas possibilitam que o sistema economize energia quando esses componentes não estão sendo utilizados. Dentre as diversas técnicas de gerenciamento de energia encontradas na bibliografia, estão as de *Dynamic Voltage Scaling* (DVS) [PLS01, WWDS94] e as de hibernação de recursos [KK98, HLS96, FS99a].

A utilização dessas técnicas em sistemas embarcados é importante para a economia de energia. Entretanto, freqüentemente, elas ocasionam sobrecustos – *overhead* – (causados por um gerente ou por um monitor de energia), ou então latência, definida como o tempo requerido para o componente entrar em um modo de economia de energia e sair dele. Por um lado, esses fatores não podem ser ignorados em um sistema que precisa atender determinadas restrições especificadas pelo projetista, como restrições de tempo real e/ou de qualidade de serviço(QoS). Por outro, não é suficiente garantir tais restrições se a carga da bateria acaba antes do tempo previsto de funcionamento do sistema. Desta forma, considera-se que o tempo mínimo de funcionamento do sistema, ou tempo mínimo de duração da bateria, seja também uma importante restrição de projeto em muitos casos.

Alguns trabalhos na literatura exploram a integração das técnicas de gerenciamento de energia com abordagens que garantem o atendimento de restrições

de tempo real ou qualidade de serviço (QoS em termos de processamento, memória ou comunicação). Entretanto, a maioria dessas abordagens busca apenas minimizar o consumo de energia com o foco principal nessas métricas tradicionais, e não avaliam o tempo de duração previsto para a carga da bateria.

No contexto apresentado, a energia não pode continuar marginal às restrições de projeto a serem atendidas. Quando especificado, o sistema deve garantir que a bateria dure no mínimo o tempo especificado pelo projetista, garantindo pelo menos certa quantidade de energia para realizar as tarefas críticas, possivelmente com prejuízo às tarefas opcionais ou não-críticas.

1.2 Objetivo

Neste trabalho, consideramos a energia como um parâmetro de QoS para atender ao tempo de duração da bateria especificado pela aplicação e, com isso, consideramos QoS em termos de energia em sistemas embarcados de tempo real. O objetivo não é apenas reduzir o consumo de energia, mas aumentar a utilidade da aplicação em um sistema com uma carga limitada de energia, garantindo o tempo de duração da bateria e ainda preservando os *prazos* das tarefas *hard real-time*. Na abordagem proposta, o projetista define o período no qual o sistema embarcado deve estar operacional, ou seja, o tempo em que a carga da bateria deve ser suficiente para executar o sistema. Mediante o monitoramento da carga da bateria, um escalonador é capaz de selecionar as tarefas que serão executadas ou, caso seja necessário, pode diminuir os níveis de QoS para reduzir o consumo de energia e aumentar o tempo de duração do sistema. Com base nesse objetivo geral, são apresentados os seguintes objetivos específicos:

- Prover QoS em termos de energia em sistemas embarcados de tempo real por intermédio de diminuições controladas dos níveis de QoS da aplicação.
- Preservar os *prazos* das tarefas críticas de tempo real.
- Aumentar a utilidade do sistema com a execução de tarefas não críticas, sempre que for possível.

- Auxiliar na redução do consumo de energia com a utilização de mecanismos de gerenciamento de energia nos períodos ociosos criados pelas diminuições dos níveis de QoS da aplicação.

1.3 Solução Proposta

Para alcançar o objetivo proposto, o controle dos níveis de QoS da aplicação foi inspirado em técnicas de Computação Imprecisa [LSL+94, LNL87a, LNL87b]. A Computação Imprecisa divide as tarefas em duas partes: uma implementando o fluxo de execução obrigatório e outra implementando o fluxo opcional. O fluxo obrigatório é a parte *hard real-time* da tarefa e sempre deve ser executado dentro do *prazo* especificado. O fluxo opcional é a parte "melhor esforço" da tarefa, o qual é executado somente se todos os requisitos temporais das partes obrigatórias são atendidos e se ainda há capacidade de processamento ociosa. Com essa divisão, o escalonador da Computação Imprecisa impede a execução das partes opcionais quando existe a possibilidade do *prazo* de alguma parte obrigatória ser perdido e, assim, reduz a demanda por processamento do sistema. Além disso, no escalonador proposto, as partes opcionais são impedidas de executar quando o nível de energia não é suficiente para atender ao requisito de tempo de duração do sistema especificado pela aplicação. Esse controle cria períodos ociosos no sistema, que permitem ao escalonador usar técnicas de gerenciamento de energia para diminuir o consumo pelos componentes durante esses períodos.

O escalonador proposto é baseado no algoritmo *Earliest Deadline First* (EDF) [LL73], no qual as tarefas com menores *prazos* possuem maiores prioridades. Um protótipo desta proposta foi implementado no *Embedded Parallel Operating System* (EPOS) [MJWF06], um sistema operacional embarcado tendo por base componentes hierárquicos. O EPOS disponibiliza um conjunto de mecanismos para o gerenciamento de energia, desde uma infra-estrutura que permite que as aplicações realizem o apropriado gerenciamento [HWF06], até um gerente com diferentes modos de operação que efetua esse gerenciamento [WJF07]. Além desses mecanismos, o EPOS possui um sistema de monitoramento da carga da bateria que informa a energia restante [GF07].

1.4 Organização do texto

A organização deste texto segue a estrutura abaixo:

- O capítulo 2 descreve as técnicas básicas de gerenciamento do consumo de energia disponíveis na literatura. Nesse capítulo, as técnicas são abordadas com ênfase para o gerenciamento em *software*: o gerenciamento dirigido pela aplicação, o dirigido pelo sistema operacional e o cooperativo.

- O capítulo 3 apresenta técnicas de escalonamento em sistemas embarcados. Nesse capítulo, enfatizamos uma técnica que visa atender aos parâmetros temporais especificados pelo usuário utilizada em sistemas de tempo real, chamada computação imprecisa e, também, o algoritmo de escalonamento EDF.

- O capítulo 4 trata do papel da energia no contexto de QoS e aborda o algoritmo do escalonador proposto com as equações de escalonabilidades das tarefas.

- No capítulo 5 realiza-se uma descrição da implementação do escalonador proposto no sistema operacional embarcado EPOS.

- O capítulo 6 apresenta as considerações finais da pesquisa e alguns trabalhos futuros.

Capítulo 2

Gerenciamento de Energia em Sistemas Embarcados

Em geral, sistemas embarcados apresentam limitações de recursos e, em muitos casos, são alimentados por baterias com uma quantidade limitada de energia. Em função disso, o tempo de vida da bateria é um fator determinante para a disponibilidade desses sistemas, o que torna necessária a utilização de uma forma eficiente de gerenciamento do consumo de energia.

Na literatura, diversas pesquisas surgiram para atender a essa necessidade, como a apresentada por Venkatachalam em [VF05]. Esses trabalhos abrangem tanto elementos relacionados a *hardware* [EGC04, KGS04, CCLH04, KKJ01, SN02, JWL+03, LFZE00, KGMS97], quanto a *software* – foco principal deste trabalho. Uma das principais técnicas de *software* é baseada em mecanismos de troca de modos de operação de determinados dispositivos.

O gerenciamento de energia baseado em mecanismos de trocas de modos de operação é realizado por meio de dispositivos que possibilitam operar em diferentes modos e por intermédio de interfaces que acessam diretamente os modos desses dispositivos. As interfaces permitem consultas e trocas dos modos de operação corrente dos dispositivos. Um fator importante a ser observado neste ponto é que os mecanismos de trocas consomem tempo e energia para realizar uma determinada mudança de

modo de operação. Com isso, o gerenciamento realizado pelo *software* deve ser cauteloso com relação à identificação de pontos específicos em que o recurso está ocioso e permite operar em um modo reduzido de operação que consome menos energia, ou mesmo, ser desligado. Caso contrário, o gerenciamento de energia será ineficiente, pois haverá perda de desempenho e consumo adicional de energia (energia consumida para realizar a troca que foi considerada errada e mais a energia gasta para desfazer essa troca).

Um dos mecanismos utilizados para realizar as trocas de modos de operação de determinados dispositivos é chamado de DVS (*Dynamic Voltage Scaling*) [PS01]. DVS permite realizar trocas de modos de operação em tempo de execução mediante mudanças na freqüência de *clock* e na tensão fornecida ao processador. Para um gerenciamento de energia ser eficiente por meio dessa técnica, é necessário identificar os momentos em que uma redução na operação do processador pode ser realizada sem prejudicar o desempenho ou sem aumentar o consumo de energia. Entretanto, essa tarefa é mais complexa do que parece a princípio, pois podem ocorrer indeterminismos e anomalias nos sistemas reais que dificultam as estimativas de cargas das tarefas, principalmente em sistemas que interagem com o meio. Pesquisas surgiram com diversos algoritmos e políticas de escalonamento que buscam identificar com um certo grau de corretude os momentos oportunos para as trocas de modos de operação do processador [PS01, PLS01, WWDS94].

Outro mecanismo citado na literatura para a troca de modo de operação é a hibernação de recursos. Esse mecanismo permite trocar os modos de operação de diferentes recursos que estão em períodos ociosos no sistema. Entretanto, a identificação desses períodos é uma tarefa complexa como ocorre no mecanismo DVS. Algumas pesquisas buscam identificar os períodos ociosos dos discos rígidos [DKB95, HLS96], das interfaces de rede [KK98, SK97], dos monitores [FS99a], dentre outros.

Considerando-se os diversos algoritmos e políticas de gerenciamento do consumo de energia relativos a *software* existentes na literatura, é possível classificá-los em três abordagens principais: o gerenciamento de energia dirigido pela aplicação, o gerenciamento de energia dinâmico e o gerenciamento de energia cooperativo. Nas próximas seções, são apresentadas essas três classificações.

2.1 Gerenciamento de Energia Dirigido pela Aplicação

Uma primeira abordagem com relação a *software* é o gerenciamento de energia dirigido pela aplicação. Essa estratégia é baseada nos conhecimentos do programador no que diz respeito à aplicação desenvolvida. O programador realiza o gerenciamento diretamente no código da aplicação mediante a inserção de funções de gerência em pontos que são conhecidos para o tratamento do consumo de energia. Dessa forma, o gerenciamento dirigido pela aplicação tem uma reduzida interferência na execução da aplicação, pois não realiza o processamento de políticas de escalonamento e de algoritmos para detectar os principais momentos a serem gerenciados.

A reduzida interferência ocasionada por essa estratégia é uma característica vantajosa para sistemas embarcados que apresentam limitações de recursos, uma vez que qualquer interferência pode prejudicar a execução do sistema. Outra característica vantajosa dessa técnica é que a tarefa do gerenciamento de energia é simplificada em muitos casos pelo fato de os sistemas embarcados, normalmente, apresentarem apenas uma aplicação.

Para aplicações determinísticas, existem ferramentas, como AEON [LWG05] e PowerTOSSIM [SHrC+04], que auxiliam o programador a detectar os pontos que utilizam mais energia antes da execução da aplicação. A detecção desses pontos é realizada por meio de análises baseadas em simulações de *hardware*. Essas ferramentas direcionam a aplicação a um determinado consumo de energia.

Entretanto, o gerenciamento dirigido pela aplicação pode ser uma tarefa complicada em aplicações complexas ou que possuem um certo grau de interatividade com o meio. Nessas aplicações, as estimativas dos momentos em que devem ser tratados os consumos de energia são prejudicadas pelo fato de o programador desconhecer o tempo exato do fluxo de execução (aplicações não determinísticas). Além disso, dependendo da complexidade do código da aplicação, a inserção manual de instruções de gerenciamento de energia pode ser um empecilho para a utilização dessa técnica.

Hoeller [HWF06, JWdO+06] desenvolveu um trabalho neste contexto, uma infra-estrutura de gerenciamento de energia que disponibiliza uma interface uni-

forme implementada por todos os componentes do sistema. Essa infra-estrutura permite realizar o gerenciamento em diferentes níveis por intermédio da hierarquia de componentes do sistema, desde uma abstração de alto nível até o acesso direto aos modos de operação do *hardware*. Além disso, a infra-estrutura permite realizar o gerenciamento de subsistemas (processamento, comunicação) ou mesmo do sistema na sua totalidade pelo gerenciamento do componente global, que "conhece" todos os componentes instanciados no sistema. As relações entre os componentes foram formalizadas por redes de Petri para permitir a geração automática dos métodos de troca de modo de operação em cada componente, com um reduzido sobrecusto e de uma forma consistente.

A interface implementada por todos os componentes do sistema é composta por dois métodos: um para acessar o modo de operação corrente do componente e outro para mudá-lo. Com essa interface, o programador insere diretamente as instruções de gerência de energia no código da aplicação; possibilitando um gerenciamento de energia sem ocasionar sobrecustos significativos ao sistema. Por exemplo, se em um certo ponto da aplicação o programador sabe que o componente UART não será utilizado por um longo período de tempo, ele manualmente insere uma instrução para mudar o modo de operação do componente. No momento em que o componente é acessado novamente, a infra-estrutura de gerenciamento de energia implementada assegura que o componente retorne ao seu modo de operação anterior.

A abordagem proposta por Hoeller não restringe a grande variedade dos modos de operação, normalmente, existentes nos dispositivos dos sistemas embarcados, e que as interfaces padrões da indústria não permitem utilizar. A infra-estrutura disponibiliza quatro modos básicos e configuráveis em cada dispositivo *FULL, LIGHT, STANDBY* e *OFF* para não ocasionar grandes complexidades no desenvolvimento do sistema. O programador pode configurar os modos básicos conforme as opções oferecidas pelo dispositivo a ser gerenciado e as necessidades da aplicação. Em cada modo disponível, existe uma relação de desempenho do dispositivo e de energia economizada. A figura 2.1 ilustra (da esquerda para a direita) a ordem crescente com relação à economia de energia e (da direita para a esquerda) a ordem crescente com relação ao desempenho do dispositivo.

Como o gerenciamento de energia é uma propriedade não funcional no

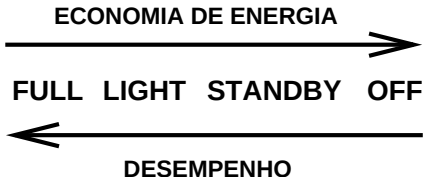

Figura 2.1: Relação entre economia de energia e desempenho do dispositivo

contexto de sistemas computacionais [LSPS05], a infra-estrutura de gerência de energia proposta por Hoeller foi implementada pelo Programa de Aspecto [KLM+97]. Com isso, a infra-estrutura é isolada do restante do sistema e deve ser ativada ou desativada conforme as necessidades de cada aplicação.

Hoeller implementou a infra-estrutura de gerenciamento de energia no sistema EPOS(*Embedded Parallel Operating System*) [MJWF06, Frö01]. O EPOS será descrito com maiores detalhes na seção 5.1 .

2.2 Gerenciamento de Energia Dirigido pelo Sistema Operacional

Uma segunda abordagem com relação ao *software* é o gerenciamento de energia dinâmico [BBM98] ou gerenciamento de energia dirigido por um agente externo à aplicação (sistema operacional ou BIOS). Essa abordagem é baseada na execução de um monitor de recursos que reúne informações por meio da análise do comportamento do sistema, verificando estatísticas de uso de diferentes componentes. As informações coletadas são utilizadas para guiar as estratégias do gerenciamento de energia e para decidir em quais momentos os modos de operação de determinados recursos serão alterados.

Nessa abordagem, não são inseridas instruções de gerenciamento diretamente no código da aplicação, pois o gerenciamento de energia é abstraído do programador. Com isso, a preocupação principal do programador concentra-se apenas no desenvolvimento da aplicação e não na energia que ela consome.

Contudo, o uso do gerenciamento dinâmico pode causar um sobrecusto

para a aplicação, pois essa técnica exige a execução de um monitor. O sobrecusto ocasionado depende da complexidade dos algoritmos executados no gerenciamento de energia. Sendo assim, é necessário ser cauteloso para que a execução dos algoritmos de gerenciamento não interfira na execução normal da aplicação, sendo a qualidade desses algoritmos o elemento que influenciará na economia de energia obtida.

A abordagem do gerenciamento dinâmico é largamente utilizada no escopo de sistemas não dedicados. Um fator que contribui para isso é o processamento do monitor ser insignificante com relação à execução global de tais sistemas. Além disso, a complexidade das aplicações que rodam nesses sistemas dificulta a utilização de técnicas que exijam alterações manuais pelo programador. Nesse contexto, existem dois sistemas padrões de gerenciamento de energia que são os mais utilizados pela indústria, *Advanced Power Management* (APM) [CC96] e *Advanced Configuration and Power Interface Specification* (ACPI) [CCC+04].

- *Advanced Power Management* foi desenvolvido como um padrão industrial pela Intel, pela Microsoft e pela IBM. No projeto de APM, a BIOS realiza as decisões com relação ao gerenciamento de energia apenas com o monitoramento do *hardware*. Os requisitos das aplicações são ignorados e existe uma falta de controle do sistema operacional com relação ao gerenciamento realizado pela BIOS – principais problemas com relação ao padrão APM.

- *Advanced Configuration and Power Interface Specification* é uma importante interface de gerenciamento de energia proposta pelas indústrias Hewlett-Packard, Intel, Microsoft, Phoenix Technologies e Toshiba. ACPI substituiu o antigo padrão de gerenciamento APM. No projeto de ACPI, o sistema operacional conhece as características do *hardware* e as necessidades e comportamentos das aplicações com relação aos recursos. ACPI disponibiliza para o sistema operacional acesso ao gerenciamento de energia e aos modos de operação dos dispositivos. O gerenciamento realizado diretamente pelo sistema operacional baseado nas necessidades das aplicações é uma grande vantagem de ACPI com relação ao antigo padrão APM.

Os padrões APM e ACPI apresentam certos inconvenientes para os sis-

temas embarcados, apesar de serem, freqüentemente, utilizados em sistemas não dedicados. O principal inconveniente é que esses padrões, normalmente, exigem recursos adicionais de *hardware* ou mesmo capacidades de processamento que podem impossibilitar a execução em sistemas embarcados.

A limitação dos sistemas embarcados com relação ao poder de processamento impede a utilização de algoritmos de gerenciamento de energia complexos. Dessa forma, nesses sistemas, a abordagem de gerenciamento dinâmico deve ser mais cautelosa para o processamento dos algoritmos não interferir na execução da aplicação. Neste contexto, pesquisas surgiram com propostas de algoritmos eficientes e com baixos sobrecustos para realizar o gerenciamento dinâmico [BBM98, PS01, BR03].

Yuan propôs GRACE-OS [YN03, Yua04], um sistema operacional eficiente em termos de energia para aplicações móveis de multimídia com tarefas *soft real-time*, como, por exemplo, câmeras de celulares. Suportar os requisitos de QoS requeridos pelas aplicações e economizar energia para uma maior disponibilidade do sistema são os principais objetivos do GRACE-OS. GRACE-OS utiliza técnicas de adaptações multicamadas para garantir QoS em sistemas com *software* e *hardware* adaptativos – sistemas que podem alterar seus comportamentos durante a execução. Além disso, GRACE-OS combina o escalonamento de tarefas *soft real-time* com os mecanismos de DVS para gerenciar o consumo de energia. No escalonamento realizado pelo GRACE-OS, são decididos a velocidade de execução de uma determinada tarefa e o tempo que esta permanecerá em execução. As decisões de escalonamento são realizadas tomando por base a probabilidade de distribuição de ciclos exigidos pela aplicação. A distribuição dos ciclos necessários pela aplicação é obtida em tempo de execução com um baixo sobrecusto por estimativas e análises individuais dos comportamentos das tarefas.

As estimativas em tempo de execução é uma vantagem do GRACE-OS, pois em outras propostas as estimativas são feitas antes da execução da aplicação. Outra vantagem que distingue GRACE-OS das demais abordagens é que o processador inicia a execução de uma determinada tarefa em um modo baixo de operação. O modo de operação do processador é alterado para um modo mais alto de operação caso a estatística do desempenho requerido não seja alcançada. Essa proposta prevê que a grande

maioria das tarefas de multimídia não utilizam todo o processamento que é alocado para elas. Dessa forma, GRACE-OS reduz o tempo em modo ocioso e fica mais tempo executando tarefas em modo baixo de operação e, assim, economiza energia. GRACE-OS foi implementado sobre o sistema operacional Linux.

Scordino propôs, de uma forma semelhante à do GRACE-OS, um novo algoritmo para o escalonamento de sistemas embarcados com tarefas *soft real-time* e, também, com tarefas *hard real-time*, chamado GRUB-PA [SL04] – desenvolvido no sistema operacional Linux. As tarefas escalonadas por esse algoritmo podem ser periódicas, esporádicas ou aperiódicas. O algoritmo apresentado baseia-se no *framework* de reserva de recursos e nas técnicas de DVS. O objetivo de GRUB-PA é selecionar a tarefa e a freqüência em que ela será executada para economizar energia, mas sem colocar em risco os *prazos* das tarefas.

Essas propostas apresentadas não possuem o objetivo de atender a um determinado tempo de duração da bateria, mas apenas economizar energia em diferentes cenários para aumentar o tempo de disponibilidade do sistema. Com base nisso, Zeng [ZELV02] apresentou um *framework* que suporta a energia como recurso de primeira classe do sistema operacional. O principal objetivo dessa proposta é atender ao tempo de duração do sistema por meio de limitações da descarga da bateria de acordo com as preferências do usuário.

O modelo de Zeng é baseado em uma “moeda” corrente que as aplicações utilizam para “pagar” (alocar) e utilizar recursos do sistema (CPU, disco, rede), chamada “currentcy”. Um “currentcy” equivale a uma determinada quantidade de energia possível de ser gasta dentro de um certo período de tempo. Esse modelo unifica o cálculo de energia sobre os diferentes dispositivos de *hardware* e proporciona uma satisfatória alocação de energia entre as aplicações.

Para ativar um tempo de duração da bateria especificado pelo usuário, o algoritmo do escalonador de Zeng segue as seguintes etapas:

- O escalonador divide o tempo em períodos.
- A cada período o escalonador distribui “currentcies” para as tarefas de acordo com

a fórmula que determina o grau de descarga possível para alcançar o tempo de vida exigido. A divisão dos “currentcies” entre as tarefas pode ser estabelecida de acordo com as preferências do usuário com relação às suas prioridades.

- Após a distribuição dos “currentcies”, o escalonador escalona as tarefas que possuem “currentcies” para “gastar” e alocar recursos. As tarefas que não possuem ou já gastaram os seus “currentcies” são deixadas para os próximos períodos.

Zeng implementou o ECOSystem, que é baseado em Linux e incorpora a proposta do modelo de “currentcy”. ECOSystem foi implementado para demonstrar como o *framework* proposto suporta explícito controle sobre o recurso da energia da bateria. Entretanto, essa proposta de Zeng pode não ser aplicável em sistemas embarcados que apresentam limitações de recursos, pois a execução dos algoritmos pode exigir processamento demasiado e ocasionar interferência significativa na aplicação.

2.3 Gerenciamento de Energia Cooperativo

Uma terceira abordagem é o gerenciamento de energia cooperativo [WBB02] ou, também chamado, gerenciamento de energia híbrido. Essa abordagem é baseada em características das técnicas anteriores: gerenciamento de energia dirigido pela aplicação e gerenciamento de energia dinâmico.

O gerenciamento de energia cooperativo é realizado por intermédio da colaboração entre o conhecimento do programador (gerenciamento dirigido pela aplicação) e do monitor que reúne informações a respeito do comportamento do sistema (gerenciamento dinâmico). Dessa forma, um compromisso é estabelecido entre as duas técnicas para realizar um tratamento de energia eficiente e sem sobrecustos significativos para a aplicação.

O compromisso estabelecido entre as técnicas de gerenciamento permite a execução de algoritmos mais simples no monitor, pois o programador auxilia com “dicas” inseridas diretamente no código da aplicação. Além disso, as partes mais complexas da aplicação, das quais o programador “desconhece” o fluxo exato de execução, são

tratadas pelo monitor. Com isso, o gerenciamento cooperativo permite um auxílio mútuo entre as duas técnicas.

Na literatura, um compromisso entre as técnicas de gerenciamento de energia foi proposto por Anand [ANF04]. A colaboração é estabelecida da seguinte forma: o monitor realiza o tratamento do consumo de energia, e a aplicação informa ao monitor quando ele erra esse tratamento de energia. A característica vantajosa dessa proposta é que, na requisição de um determinado dispositivo, o sistema permite que seja verificado o "melhor dispositivo" em um conjunto de dispositivos passíveis de serem utilizados no mesmo acesso, ou seja, o dispositivo que está no modo de operação de acordo com o acesso e que economiza mais energia. Com isso, o gerenciamento de energia proposto por esse trabalho analisa os diferentes dispositivos existentes no sistema e não somente o que foi requisitado para o acesso.

A interface descrita por Anand permite que as aplicações requisitem os modos correntes de operação dos dispositivos de entrada e saída para realizar a escolha do dispositivo a ser acessado. Com isso, essa interface possibilita que a aplicação avalie o desempenho e o custo em termos de energia no acesso a estratégias alternativas, como para escrita e leitura de dados.

Na proposta de Anand, a colaboração é realizada com a atribuição de um valor de erro ao acesso ao dispositivo pela aplicação, chamado *ghost hint*. Essa atribuição é feita no momento em que a aplicação é forçada a utilizar um determinado dispositivo, pois o ideal está em um modo baixo de operação e consumiria mais energia em ligá-lo do que acessar outro. O valor *ghost hint* indica "o acesso deveria ter sido realizado por" e corresponde a quanto a aplicação teria economizado caso o dispositivo ideal estivesse ligado. A cada atribuição de um *ghost hint* pela aplicação, o monitor verifica se o valor de erro ultrapassa um *limite* em um determinado período de tempo. No caso de o valor de erro ultrapassar o *limite*, o monitor identifica que a aplicação está constantemente tentando acessar o determinado dispositivo e, desse modo, o monitor reativa o dispositivo para ser acessado. Com isso, a abordagem permite que a aplicação descubra erros de gerenciamento de energia na presença de múltiplos dispositivos e informe ao monitor esses erros.

Essa abordagem é relevante e estende o tempo de duração da bateria, mas não objetiva atender ao tempo de duração do sistema especificado pelo usuário. Nesse contexto, Flinn propôs [FS99a] uma abordagem que busca atender ao tempo especificado pelo usuário com técnicas de gerenciamento cooperativo entre o sistema operacional e a aplicação. Na abordagem de Flinn, o sistema operacional realiza o monitoramento da energia fornecida e da energia necessária para executar as tarefas. Com essas informações, o monitor é capaz de selecionar o estado correto entre economia de energia e qualidade da aplicação. Esse trabalho também demonstra como as aplicações podem dinamicamente alterar seus comportamentos (qualidade da aplicação) com o objetivo de economizar energia.

Para validar essa proposta, Flinn estendeu uma plataforma da computação móvel, chamada Odyssey, para estimar a energia futura que será necessária por meio de medições do uso no passado. Nessa validação, Flinn utilizou a ferramenta PowerScope [FS99b] para coletar informações dos consumos de energia das tarefas. A ferramenta PowerScope permite identificar o consumo de energia de uma tarefa específica ou de um procedimento específico dentro de um processo.

Odyssey notifica a aplicação a que ela deve se adaptar na ocorrência da estimativa de falta de energia para alcançar o tempo de duração da bateria exigido pelo usuário. Nesse caso, a aplicação utiliza a qualidade (“fidelidade”) dos dados para realizar a economia de energia. A partir disso, a redução da qualidade dos dados é dirigida pela aplicação, como, por exemplo, a perda *frames* na reprodução de um vídeo.

Odyssey realiza periodicamente três tarefas principais para atender ao tempo de duração da bateria exigido pelo usuário:

- Determina o restante da bateria.
- Estima a energia necessária para executar as tarefas.
- Decide se a aplicação deve realizar uma adaptação com base nas informações coletadas.

O monitor deve decidir qual aplicação notificar em sistemas com

aplicações concorrentes. A aplicação de mais baixa prioridade é notificada no caso da necessidade de redução da qualidade dos dados, e a de mais alta prioridade é notificada no caso da necessidade de aumento da qualidade dos dados.

Capítulo 3

Escalonamento de Tempo Real

Sistemas de tempo real são, normalmente, descritos por meio do modelo de execução baseado em tarefas. Essas tarefas possuem parâmetros temporais que exigem garantias da qualidade de serviço no domínio do tempo. Para elas, não bastam os dados de saída estarem logicamente corretos, elas necessitam, também, que os dados de saída sejam gerados dentro de um *prazo* satisfatório. Caso contrário, apesar de um cálculo correto, o resultado obtido é considerado errado e as conseqüências podem ser críticas.

A figura 3.1 apresenta uma divisão conforme a previsibilidade oferecida pelas abordagens básicas para o escalonamento das tarefas com requisitos temporais. As abordagens do primeiro grupo (*Garantia no Projeto*) apresentam uma previsibilidade determinística durante o projeto. Essas abordagens são utilizadas em sistemas em que é necessário uma garantia de que todos os *prazos* sejam atendidos em tempo de projeto. Elas podem ser subdivididas em *Executivo Cíclico* e *Teste de Escalonabilidade + Prioridades*.

O *Executivo Cíclico* possui um teste de escalonabilidade que verifica se é possível atender aos *prazos* e realiza o cálculo da escala de execução das tarefas em tempo de projeto. O resultado é uma grade de execução que determina qual tarefa irá executar e em qual instante. O *Teste de Escalonabilidade + Prioridades* também possui o teste de escalonabilidade em tempo de projeto. A diferença dessa abordagem para a anterior é que a escala de execução é feita em tempo de execução por intermédio de um

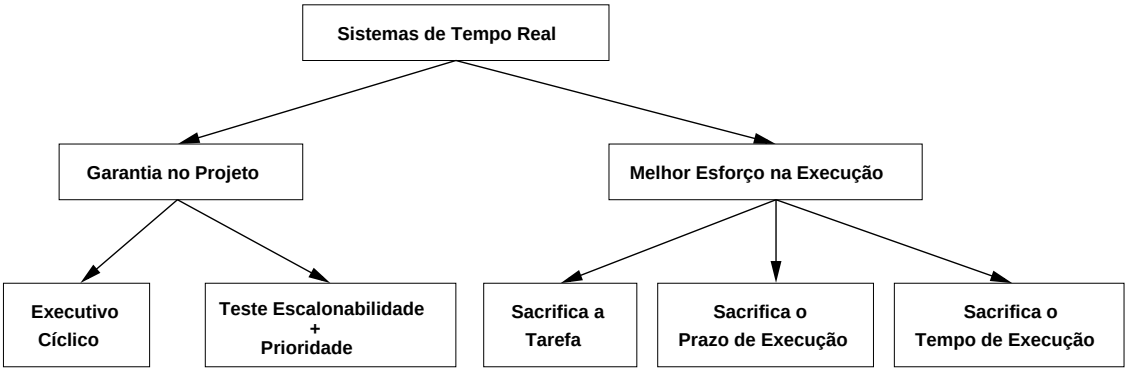

Figura 3.1: Abordagens básicas para o escalonamento de tempo real

escalonador preemptivo baseado em prioridades das tarefas.

O problema com relação à garantia no projeto é a necessidade de o conjunto de tarefas ser limitado e conhecido previamente. Outra questão é a necessidade de utilizar estimativas de recursos para os piores casos de execuções. A utilização das estimativas para os piores casos, por muitas vezes, subutiliza o sistema, pois pode apresentar valores muito mais altos do que os valores obtidos na média. Com isso, esse grupo de abordagens realiza uma previsibilidade determinística, mas perde na flexibilidade e pode subutilizar os recursos do sistema.

As abordagens do segundo grupo (*Melhor Esforço na Execução*) não apresentam garantias em tempo de projeto. Elas executam, na medida do possível, as tarefas com relação aos seus respectivos *prazos*. A ocorrência de sobrecarga nessas abordagens é possível e deixa de ser uma anomalia do sistema. O sistema está em estado de sobrecarga quando ele não consegue atender aos *prazos* de todas as tarefas em execução, o que torna a maior parte dessas abordagens impraticável em aplicações críticas de tempo real. As abordagens desse grupo podem ser subdivididas conforme as técnicas de tratamento da sobrecarga: descarte completo de tarefas, perda de *prazos* ou sacrifício do tempo de execução de algumas tarefas.

No contexto deste trabalho, um exemplo com relação a essas técnicas é observado em Niu [NQ05]. Niu propôs minimizar a anergia consumida para sistemas *soft real-time* enquanto garante requisitos de QoS. Esse objetivo é alcançado por meio de um

algoritmo de escalonamento híbrido (estático/dinâmico) que utiliza DVS e por meio de técnicas de particionamento do conjunto de tarefas em: tarefas obrigatórias e em tarefas opcionais. Nesse trabalho, os requisitos de QoS são qualificados pela restrição *(m,k)*, a qual especifica que as tarefas devem atender, no mínimo, à *m deadlines* em qualquer *k* liberações de tarefas consecutivas. Em um trabalho semelhante, Harada [HUN06] busca resolver o compromisso entre a maximização dos níveis de QoS e a minimização do consumo de energia, pela alocação de ciclos e freqüência do processador com garantias de QoS. Nesse trabalho, cada tarefa é dividida em parte obrigatória e em parte opcional, premissa básica para a Computação Imprecisa [LNL87a, LNL87b, LSL+94] que será descrita neste capítulo. Entretanto, o objetivo dessas pesquisas não é atender a um determinado tempo de duração do sistema.

Apesar de existirem diversos algoritmos dentre as abordagens mencionadas, neste trabalho daremos maior ênfase à técnica da Computação Imprecisa. A Computação Imprecisa faz parte das abordagens do tipo *Melhor Esforço* com o sacrifício do tempo de execução das tarefas, diminuições dos níveis de QoS. Além dessa técnica, neste capítulo entraremos em maiores detalhes no algoritmo EDF [LL73] que faz parte das abordagens do tipo *Teste de Escalonabilidade + Prioridades*.

Na próxima seção, apresentam-se a Computação Imprecisa, as tarefas imprecisas e as técnicas para as diminuições controladas dos níveis da qualidade de serviço da Computação Imprecisa. Além disso, descrevem-se com maiores detalhes as modelagens das tarefas imprecisas e as possibilidades de aplicações na prática da computação imprecisa. Finalizando, aborda-se o algoritmo EDF.

3.1 Computação Imprecisa

A Computação Imprecisa é uma técnica para o escalonamento de tarefas de tempo real que possui o objetivo de atender aos parâmetros temporais por intermédio de diminuições controladas dos níveis de QoS. Com a Computação Imprecisa, o sistema realiza o trabalho possível no prazo estipulado [dO97]. Esse sistema, chamado sistema impreciso, perde em termos de QoS do resultado, mas ganha em termos das garantias

temporais.

3.1.1 Tarefas Imprecisas

Na Computação Imprecisa, as tarefas, chamadas tarefas imprecisas, são divididas em duas partes: uma parte obrigatória e outra parte opcional. A parte obrigatória gera resultados imprecisos, mas com o mínimo da qualidade de serviço necessária para a continuidade do sistema. Além disso, a execução da parte obrigatória é garantida mesmo nas ocorrências de sobrecargas no sistema.

A execução da parte obrigatória com a parte opcional gera resultados ditos precisos. A parte opcional realiza o melhoramento do resultado, com o retorno igual ou superior ao resultado gerado apenas pela parte obrigatória. Entretanto, a execução da parte opcional não é garantida a priori, pois, caso haja sobrecarga no sistema, ocorrerão os descartes parciais ou totais de partes opcionais. A escolha de quais partes opcionais serão descartadas permite uma diminuição controlada dos níveis de QoS do sistema.

A Computação Imprecisa faz parte das abordagens do tipo *Melhor Esforço*, pois não oferece uma previsibilidade determinística para todas as tarefas do sistema. No entanto, somente as partes obrigatórias formam um subconjunto que deve ser analisado do ponto de vista das abordagens com garantias em tempo de projeto. Dessa forma, a divisão proposta pela computação imprecisa une as técnicas da computação crítica de tempo real (parte obrigatória) com as técnicas de "melhor esforço" (parte opcional). Neste contexto, este trabalho utiliza o algoritmo EDF (descrito na seção 3.2) para oferecer a previsibilidade determinística necessária para as partes obrigatórias.

A Computação Imprecisa permite também a existência de tarefas que são somente obrigatórias ou somente opcionais no sistema, desde que os prazos das tarefas obrigatórias sejam respeitados. Isso pelo fato de nem todas as tarefas de uma aplicação serem divisíveis em duas partes distintas.

3.1.2 Modelagem das Tarefas Imprecisas

As tarefas imprecisas que são divisíveis em partes obrigatórias e partes opcionais podem ser modeladas de três formas:

- Funções monotônicas - são funções que melhoram a qualidade do resultado durante o tempo que permanecem em execução e na pior das hipóteses não o alteram. O término dessas funções pode ocorrer em qualquer momento da execução sem ocasionar problemas de integridade do resultado, assim sendo, o escalonador pode decidir em qualquer instante finalizar a execução. Nesse caso de modelagem, a parte obrigatória retorna a solução com o mínimo de QoS necessária para a continuidade da aplicação, e a parte opcional faz refinamentos sucessivos nessa solução.

- Funções de melhoramento - são aquelas que fazem o melhoramento das entradas e geram saídas com o grau de precisão igual ou superior às entradas. Porém, as funções de melhoramento exigem que sejam executadas integralmente, sendo dispensável a execução dessas funções pela metade. Dessa forma, o escalonador deve decidir se irá executar ou não as funções de melhoramento antes do início da execução, ou seja, no máximo após o término da execução da parte obrigatória.

- Múltiplas versões - nessa técnica de modelagem, existem duas versões fixas da tarefa. A primeira versão retorna resultados precisos, parte obrigatória mais a parte opcional, mas esta versão pode ocupar mais tempo de execução do que o alocado ou o tempo é desconhecido. A segunda versão retorna resultados imprecisos, parte obrigatória, mas o tempo é conhecido e é garantida a execução dentro do prazo proposto. Na modelagem por versões, o escalonador deve escolher se irá executar a parte opcional antes mesmo de começar a executar a obrigatória, ou seja, ele opta uma das versões que executará.

A figura 3.2 apresenta o momento em que o escalonador pode descartar a execução de uma parte opcional para as três modelagens possíveis.

Com base em uma dessas três modelagens, é possível implementar as tarefas imprecisas de uma aplicação. Entretanto, as modelagens que exigem que as tarefas

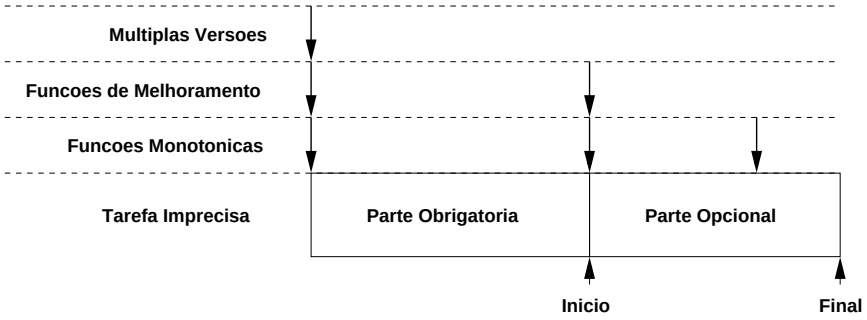

Figura 3.2: Momento em que o escalonador pode descartar uma parte opcional

opcionais sejam totalmente executadas ou então descartadas – restrição 0/1 – possuem problemas sérios de projeto. Esses problemas de projeto são evidenciados quando são identificadas a sobrecarga e a possível perda de *prazos* durante a execução de uma tarefa opcional com restrição 0/1. Nesse caso, uma escolha deve ser realizada com relação à tarefa opcional em execução: descartar a tarefa ou então permitir a perda de *prazos*. Entretanto, nenhuma dessas escolhas é simples e fácil de ser projetada, uma vez que:

- O simples descarte da tarefa opcional em execução necessita que os resultados e as variáveis alterados devam ser cancelados e os valores originais reparados. Entretanto, os valores originais podem ter sido perdidos durante a execução.

- A permissão da perda de *prazos* também é problemática, pois, em sistemas críticos, a perda dos parâmetros temporais não é uma opção considerável.

Dessa forma, a modelagem de tarefas com restrições 0/1 é complexa e os estudos em torno dela são teóricos e, por muitas vezes, não aplicáveis para a realidade, mas apenas em simulações.

3.1.3 Usos da Computação Imprecisa

As possibilidades de usos da Computação Imprecisa são diversas e em diferentes áreas. Uma possibilidade de uso dessa técnica é no processamento de imagens, em que as partes obrigatórias resultariam em uma imagem com uma qualidade de serviço

mínima aceitável, enquanto as partes opcionais melhorariam e refinariam esse resultado. Nesse caso, se houvesse uma sobrecarga no sistema, os refinamentos na imagem seriam deixados de lado sem prejudicar a aplicação.

Outra possibilidade de uso da Computação Imprecisa é a aquisição de dados que consome recursos para ser realizada (tempo, energia). A parte obrigatória poderia realizar uma única aquisição dos dados e a parte opcional implementaria a aquisição múltipla para realizar médias e calcular estatísticas de mudanças dos valores. Em sistemas embarcados, um exemplo clássico da necessidade de múltiplas leituras de um mesmo dado é a leitura da temperatura por meio de um conversor analógico digital (ADC). Como esse componente é impreciso, múltiplas leituras podem/devem ser realizadas para um valor real ser obtido. Todavia, essas leituras devem ser feitas com cautela, pois cada leitura consome um determinado tempo e energia para ser realizada. Neste caso, implementam-se as múltiplas leituras como uma parte opcional que será executada somente se existirem recursos para tal.

Além dessas possibilidades, existem inúmeras outras, como a implementação de algoritmos tipo “a qualquer tempo”. Nesses algoritmos, os resultados podem ser obtidos a qualquer momento da execução, entretanto, em geral quanto mais tempo for executado o algoritmo melhor será a qualidade e precisão do resultado. Exemplos desses algoritmos são as pesquisas heurísticas, como os algoritmos de melhor caminho, caminho mais curto, percorrimento em grafos.

As implementações de algoritmos de “controle e conforto” são outras possibilidades para usos da computação imprecisa. Nesses algoritmos, o “controle” é implementado pelas partes obrigatórias com prazos estipulados. O “conforto” é implementado pelas partes opcionais que apenas refinam os resultados e/ou fazem novos cálculos com os resultados obrigatórios.

Os algoritmos de “controle e conforto” são freqüentemente utilizados em sistemas embarcados. Exemplo de uso desses algoritmos é o monitoramento da temperatura de uma caldeira que pode explodir caso ultrapasse uma certa temperatura. Nesse exemplo, o “controle” verifica em períodos específicos a temperatura da caldeira, com objetivo de detectar se essa temperatura ultrapassou um certo valor. Caso ultrapasse, o

"controle" deve imediatamente acionar recursos para diminuir a temperatura ou notificar outros mecanismos que o façam. Portanto, a tarefa do "controle" é essencial ao sistema e os prazos devem ser respeitados. O "conforto" pode realizar cálculos adicionais nos dados das temperaturas obtidas, como a média das temperaturas analisadas em um determinado período, a contagem do número de vezes que a temperatura chegou a um certo nível e outros cálculos. Entretanto, em caso de sobrecarga no sistema, os cálculos realizados pelo "conforto" podem ser descartados pelo maior grau de importância dos trabalhos realizados pelo "controle".

3.2 *Earliest Deadline First*

Neste trabalho foi escolhido o algoritmo EDF para realizar a previsibilidade determinística exigida pelas partes obrigatórias. EDF é um mecanismo de escalonamento de tempo real baseado em prioridades dinâmicas [LL73]. EDF distribui maiores prioridades para as tarefas com *prazos* mais curtos. Em tempo de projeto, um teste de escalonabilidade avalia a possibilidade de alguma tarefa perder o seu respectivo *prazo*. Em tempo de execução, um escalonador preemptivo escolhe a tarefa em estado *Pronto* de mais alta prioridade.

A equação 3.2 apresenta um teste de escalonabilidade exato para o algoritmo EDF. O sistema de tempo real considerado contém um conjunto de n tarefas periódicas e independentes, $\mathcal{T} = \{\tau_0, \tau_1, ..., \tau_{n-1}\}$. Cada τ_i é caracterizado por três parâmetros, (P_i, D_i, C_i), onde, P_i é o período em que a tarefa i é escalonada, D_i é o *prazo* máximo de conclusão e C_i é o tempo de execução da tarefa i no pior caso (incluído tempos de espera pela inversão de prioridades). Para esse teste, é suposto que $\forall\tau_i$, $D_i \leq P_i$. A fração de processamento utilizada por uma tarefa i, U_i, é representada pela equação (3.1).

$$U_i = \frac{C_i}{D_i} \tag{3.1}$$

A capacidade de um processador é definida como 1, ou seja, 100%. Um sistema com ω processadores possui capacidade ω. Dessa forma, para as tarefas serem

escalonáveis no algoritmo EDF, o somatório das frações de processamento de todas as tarefas deve ser menor ou igual à capacidade dos processadores, ou seja,

$$\sum_{i=1}^{n} \left(\frac{C_i}{D_i} \right) \leq \omega \tag{3.2}$$

onde $\omega = 1$ para um sistema com apenas um processador. Caso $\sum_{i=1}^{n} U_i > \omega$, o processador estará sobrecarregado, e as tarefas não são escalonáveis nesse algoritmo.

Capítulo 4

Escalonamento com Requisitos de Tempo e de Energia

A qualidade de serviço pode ser definida como o conjunto de características (qualitativas e quantitativas) suportadas por um sistema para atingir uma determinada funcionalidade especificada pelo usuário. Assim, QoS é o mecanismo que controla o grau de satisfação do usuário com relação a um determinado serviço utilizado.

A utilização da qualidade de serviço em um sistema ocorre com o estabelecimento de parâmetros pelo usuário, como a porcentagem de perdas, de atrasos, da vazão de um canal de dados, etc. O sistema mapeia os parâmetros estabelecidos entre os componentes necessários e verifica a possibilidade de esses parâmetros serem atendidos. Durante a execução, o sistema monitora os recursos alocados para assegurar que os parâmetros de QoS continuem sendo atendidos. Uma renegociação dos recursos alocados é realizada caso haja uma alteração no comportamento do sistema que impeça os parâmetros de serem atendidos. Com isso, oferecer QoS para um sistema significa que os parâmetros estabelecidos pelas aplicações serão atendidos durante a execução e dentro de limites fornecidos. A qualidade de serviço pode ser utilizada para atender aos requisitos das aplicações em diferentes métricas da computação, como para o processamento de tarefas (CPU), para a comunicação de dados (rede), para o uso do disco e outros.

Com a demanda por dispositivos alimentados por baterias, argumenta-

mos que não é suficiente apenas garantir métricas tradicionais de QoS se a carga da bateria termina antes do término das tarefas. Nesse contexto, surge a necessidade de o tempo de disponibilidade do sistema ser mais um parâmetro de QoS a ser tratado. Com base nisso, a energia é um serviço que não pode continuar de fora com relação às métricas de QoS, sendo assim, neste trabalho utilizamos qualidade de serviço em termos de energia para garantir o tempo de disponibilidade do sistema. Para tal, a seção 4.1 apresenta o algoritmo do nosso escalonador que objetiva atender aos requisitos temporais e energéticos da aplicação. As seções 4.2 e 4.3 descrevem, respectivamente, os testes em tempo de projeto e em tempo de execução que analisam a escalonabilidade de um determinado conjunto de tarefas em termos dos *prazos* e do tempo de duração da bateria.

4.1 Algoritmo de Escalonamento

O nosso escalonador, fundamentado no algoritmo EDF e modelado com base na Computação Imprecisa, garante a execução das partes obrigatórias com os seus respectivos *prazos* atendidos, independentemente do nível de energia do sistema. Contudo, a execução das partes opcionais não é garantida. Nesta proposta, as partes opcionais são executadas somente se os *prazos* das partes obrigatórias e o tempo de duração da bateria desejado são sustentados. A figura 4.1 representa as tarefas que atendem ao parâmetro de energia (tempo de duração do sistema) e as que atendem ao parâmetro do tempo (*prazo* das partes obrigatórias). A intersecção dessas representações indica as tarefas que podem ser executadas e que serão atendidas em relação aos dois parâmetros desejados (energia e tempo). As tarefas fora dessa intersecção não são escalonáveis neste algoritmo.

O objetivo deste escalonador não é apenas economizar a energia consumida no sistema, pois, caso contrário, a técnica seria simplesmente nunca executar as partes opcionais. Com base nisso, o objetivo principal é atender ao tempo especificado pela aplicação com a execução dentro dos *prazos* das partes obrigatórias e com a execução do máximo possível das partes opcionais, ou seja, otimizar a utilidade da aplicação.

A figura 4.2 apresenta o algoritmo do escalonador proposto neste tra-

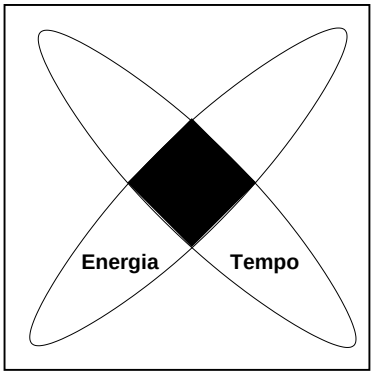

Figura 4.1: Intersecção entre a energia e o tempo.

balho, no qual as partes obrigatórias e opcionais são tratadas como tarefas em termos de escalonamento. π é o intervalo entre medições da carga da bateria que pode ser especificado pelo programador da aplicação e que deve levar em consideração que cada medição também consome energia para ser realizada.

O intervalo entre medições irá depender do estado de energia da bateria constatado na última medição. Caso a última medição verifique a existência de energia suficiente e que ultrapasse um determinado *limite*, o valor do intervalo será maior, pois o sistema não necessita que sejam realizadas medições freqüentes. Entretanto, se a última medição observar que a energia existente não é suficiente para atender ao tempo de duração especificado, as medições devem ser mais freqüentes, pois tarefas opcionais estão sendo descartadas.

4.2 Testes de Escalonabilidade em Tempo de Projeto

Como o escalonador proposto é baseado no algoritmo EDF, é possível adaptar o clássico teste de escalonabilidade do EDF para levar em consideração no cálculo a energia e a divisão das tarefas em partes obrigatórias e partes opcionais. Supondo que o sistema de tempo real considerado possua n tarefas periódicas e independentes, $\tau = \{\tau_0, \tau_1, ..., \tau_{n-1}\}$. P_i, D_i e C_i são parâmetros de cada τ_i, onde P_i é o período no qual a tarefa i é escalonada, D_i é o máximo *prazo* relativo de τ_i, e C_i é o tempo de execução no

```
1  for cada tarefa que entra no estado PRONTO do
2      Defini a nova prioridade;
3      Seleciona na fila a tarefa de mais alta prioridade;
4      for cada π unidades de tempo do          /* π é configurável */
5          Lê o monitor de energia;
6          Verifica se existe energia suficiente para atender ao tempo especificado pela
           aplicação;
7          Ajusta o valor de π;
8      end
9      if tarefa é hard real-time then
10         Executa a tarefa selecionada;
11     else                                  /* tarefa é best effort */
12         if existe energia suficiente para atender ao tempo de duração requerido
           then
13             Executa a tarefa selecionada;
14         else                          /* energia não é suficiente */
15             Executa o gerente de energia oportunista;
16         end
17     end
18 end
```

Figura 4.2: Algoritmo de escalonamento

pior caso de τ_i. Assumimos que $\forall\tau_i$, $D_i \leq P_i$. A fração de processamento, U_i, de τ_i é representada por $U_i = \frac{C_i}{D_i}$.

No modelo da computação imprecisa, cada τ_i é dividida em parte obrigatória e parte opcional com tempos de execuções nos piores casos, respectivamente, de μ_i e θ_i. Com isso, o tempo total de execução de τ_i no pior caso é $C_i = \mu_i + \theta_i$. Para garantir que nenhum *prazo* das partes obrigatórias seja perdido, a equação (4.1) deve ser

respeitada.

$$\sum_{i=1}^{n} \left(\frac{\mu_i}{D_i} \right) + \sigma \leq \omega \tag{4.1}$$

Onde $\omega = 1$ para um sistema com um único processador e σ representa o somatório das interferências em cada tarefa (nos piores casos), que inclui: tempo gasto no sistema operacional, nas trocas de contexto, no próprio algoritmo de escalonamento. A equação (4.1) deve ser atendida para as tarefas serem escalonáveis em relação aos *prazos* das partes obrigatórias, caso contrário ($\sum_{i=1}^{n} \left(\frac{\mu_i}{D_i} \right) + \sigma > \omega$), o processador estará sobrecarregado.

Com a inclusão do tempo de execução da parte opcional na equação (4.1), podemos determinar se as tarefas serão executadas integralmente. Entretanto, é importante observar que a equação (4.2) não é um requisito fundamental no nosso algoritmo e somente será relevante quando a equação (4.1) é válida, caso contrário, as tarefas já não seriam escalonáveis.

$$\sum_{i=1}^{n} \left(\frac{\mu_i + \theta_i}{D_i} \right) + \sigma \leq \omega \tag{4.2}$$

As partes obrigatórias e as partes opcionais são escalonáveis em relação aos seus *prazos* quando a equação (4.2) for respeitada. Caso contrário, uma certa fração de tempo χ das partes opcionais é descartada, apresentada pela equação (4.3).

$$\chi = \frac{\sum_{i=1}^{n} \left(\frac{\mu_i + \theta_i}{D_i} \right) + \sigma - \omega}{\sum_{i=1}^{n} \left(\frac{\theta_i}{D_i} \right)} \tag{4.3}$$

O objetivo em relação à energia pode ser alcançado seguindo o mesmo raciocínio lógico utilizado até o presente momento, mas levando em consideração o consumo de energia das tarefas. O consumo de energia de τ_i no pior caso, E_i, é dado pela soma dos consumos de energia da parte obrigatória e da parte opcional nos piores casos, respectivamente, $E_{\mu i}$ e $E_{\theta i}$, ($E_i = E_{\mu i} + E_{\theta i}$). Supomos que, semelhantemente aos tempos de execuções nos piores casos, os consumos de energia nos piores casos são previamente conhecidos pelo programador da aplicação. Esses valores podem ser obtidos traçando os perfis ou por meio de outras técnicas. O número máximo possível de

execuções, η_i, de τ_i no tempo requerido pela aplicação, T_t, é dado pela divisão entre o tempo requerido e o intervalo de execução de τ_i, ou seja, $\eta_i = \frac{T_t}{P_i}$. T_t é dado pelo programador da aplicação baseado na capacidade da bateria. Com o intuito de atender, no mínimo, às partes obrigatórias das tarefas, temos a equação (4.4) que indica se o conjunto das tarefas será escalonável em relação à energia.

$$\sum_{i=1}^{n} \left(\frac{E_{\mu i} \times \eta_i}{E_t} \right) + \epsilon \leq 1 \tag{4.4}$$

Onde E_t é a energia total do sistema (especificação da bateria), ou seja, a capacidade da bateria, ϵ representa o somatório das interferências (em cada tarefa) do consumo de energia dos serviços do sistema operacional (nos piores casos), por exemplo, de alarmes, trocas de contexto e pelo próprio escalonador. A capacidade do sistema em relação à energia é definida como 1, ou seja, 100%. Substituindo o número máximo de execuções possíveis η_i de τ_i na equação (4.4) temos a equação (4.5).

$$\sum_{i=1}^{n} \left(\frac{E_{\mu i} \times T_t}{P_i \times E_t} \right) + \epsilon \leq 1 \tag{4.5}$$

As tarefas são escalonáveis em relação à energia no nosso algoritmo se a equação (4.5) for respeitada. Caso contrário ($\sum_{i=1}^{n} \left(\frac{E_{\mu i} \times T_t}{P_i \times E_t} \right) + \epsilon > 1$), o sistema não atenderá ao tempo de duração requerido pela aplicação para esse conjunto de tarefas.

A inclusão da energia consumida pelas partes opcionais no pior caso na equação (4.5) permite que verifiquemos se as tarefas serão executadas integralmente. Como discutido anteriormente, isso não é um requisito obrigatório, e a equação (4.6) só deve ser calculada se a equação (4.5) é respeitada, ou seja, as partes obrigatórias são atendidas.

$$\sum_{i=1}^{n} \left(\frac{(E_{\mu i} + E_{\theta i}) \times T_t}{P_i \times E_t} \right) + \epsilon \leq 1 \tag{4.6}$$

Todas as partes obrigatórias e opcionais das tarefas são executadas em relação à energia do sistema se a equação (4.6) é respeitada. Caso contrário, uma determinada fração de tempo γ das partes opcionais não será executada, pois o sistema não

atenderia ao tempo de duração desejado pela aplicação. A equação (4.7) fornece a fração de tempo das partes opcionais descartadas em relação à energia.

$$\gamma = \frac{\sum_{i=1}^{n} \left(\frac{(E_{\mu i} + E_{\theta i}) \times T_t}{P_i \times E_t} \right) + \epsilon - 1}{\sum_{i=1}^{n} \left(\frac{E_{\theta i} \times T_t}{P_i \times E_t} \right)} \tag{4.7}$$

Com o objetivo de atender aos requisitos temporais e energéticos, é necessário respeitar todas os *prazos* das partes obrigatórias e o tempo de duração da bateria especificado pela aplicação. A equação (4.8) é utilizada para determinar se um certo conjunto de tarefas é escalonável no nosso algoritmo.

$$\left[\sum_{i=1}^{n} \left(\frac{\mu_i}{D_i} \right) + \sigma \leq \omega \right] \wedge \left[\sum_{i=1}^{n} \left(\frac{E_{\mu i} \times T_t}{P_i \times E_t} \right) + \epsilon \leq 1 \right] \tag{4.8}$$

As partes obrigatórias das tarefas têm as execuções garantidas no nosso escalonador em relação aos seus *prazos* e ao parâmetro de energia se a equação (4.8) é respeitada. A fração de tempo máxima λ possível de tarefas opcionais perdidas em relação ao tempo e à energia pode ser obtida por intermédio da equação (4.9).

$$\lambda = \max(\chi, \gamma) \tag{4.9}$$

4.3 Teste de Escalonabilidade em Tempo de Execução

Com objetivo de prover qualidade de serviço em termos de energia e fazer melhor uso dos recursos com a execução das partes opcionais, é necessário verificar periodicamente se o tempo de duração do sistema requerido pela aplicação pode ser atendido em tempo de execução. Para isso, o tempo de duração do sistema, $T_{t\kappa}$, e a energia do sistema (carga da bateria), $E_{t\kappa}$, no instante κ são recalculados. A equação (4.10) pode ser calculada com os novos valores com o intuito de verificar se $T_{t\kappa}$ pode ser atendido no instante κ. Essa equação utiliza a taxa de descarga do período anterior para verificar se $T_{t\kappa}$ pode ser alcançado com os níveis atuais de utilização de energia.

$$\left[\left(\frac{E_{t\kappa}}{E_{t\kappa-1} - E_{t\kappa}} \right) \times (T_{t\kappa-1} - T_{t\kappa}) \right] \geq T_{t\kappa} \tag{4.10}$$

Se a equação (4.10) é respeitada, existe energia suficiente para atender ao $T_{t\kappa}$. Com isso, todas as partes obrigatórias são executadas e as partes opcionais são escalonadas. Caso contrário, algumas partes opcionais serão descartadas, diminuindo os níveis de QoS, e o escalonador invoca o gerente de energia oportunista usando o tempo em que as partes opcionais estariam em execução. Dessa forma, o escalonador aproveita o tempo ocioso criado no sistema para economizar energia. Se $T_{t\kappa+\iota}$ em um instante futuro $\kappa + \iota$ é atendido, as partes opcionais retornam a serem escalonadas, aumentando os níveis de QoS.

Capítulo 5

Implementação

Este capítulo apresenta e discute o protótipo desenvolvido com o intuito de testar a abordagem de escalonamento proposta, usando o *Embedded Parallel Operating System* (EPOS) [Frö01, MJWF06]. A seção 5.1 descreve com maiores detalhes o sistema EPOS com a infra-estrutura de gerência de energia disponível. A seção 5.2 apresenta detalhes da implementação do escalonador proposto neste trabalho. A seção 5.3 realiza uma descrição do gerente de energia implementado para aproveitar o tempo ocioso criado com os descartes das partes opcionais. A seção 5.4 apresenta um estudo de caso com relação ao escalonador implementado nesta pesquisa.

5.1 *Embedded Parallel Operating System*

O sistema EPOS serviu para provar os conceitos da metodologia de projeto de sistemas orientados à aplicação (Application-Oriented System Design - AOSD) proposta por Fröehlich [Frö01]. AOSD une diferentes técnicas de engenharia de *software* e de programação que permitem a geração de sistemas específicos para determinadas aplicações.

O EPOS é um *framework* baseado em componentes hierarquicamente organizados para a geração de sistemas específicos a uma determinada aplicação embarcada. Ferramentas de análise permitem componentes serem automaticamente sele-

cionados para atender aos requisitos dessas aplicações específicas. Pela definição, uma instância do sistema agrega todo o suporte necessário para aplicação e nada mais.

Por intermédio de um conjunto de componentes de alto nível chamados de Abstrações, o EPOS implementa funcionalidades abstratas de forma independente de plataforma, como, **Thread**, **Alarm**, variáveis de condições (**Mutex, Semaphore**), entre outros. As Abstrações utilizam componentes chamados de Mediadores de *Hardware* [PF04] que exportam as funcionalidades dos dispositivos de *hardware* por meio de uma interface uniforme, abstraindo as dependências de *hardware* para as Abstrações. O EPOS utiliza aspectos, que isolam as características não-funcionais do sistema como depuração, compartilhamento, e ele usa também características configuráveis, que permitem a configurabilidade do sistema. Com a utilização dessas características e técnicas associadas à meta-programação estática e programação orientada a aspectos, o EPOS permite aos programadores desenvolverem aplicações independentes de plataformas em um sistema configurável e adaptativo. O EPOS suporta diferentes arquiteturas como IA32, PowerPC, Sparc, MIPS, H8 e AVR [MJWF06].

No EPOS, todo componente do sistema implementa uma interface uniforme para a gerência de energia dirigida pela aplicação [HWF06], como descrito anteriormente na proposta de Hoeller. Essa infra-estrutura permite que as aplicações interajam com o sistema para implementar uma gerência de energia apropriada sem comprometer a portabilidade e sem ocasionar sobrecustos excessivos para a aplicação. Além disso, o sistema disponibiliza um monitor da carga da bateria e uma infra-estrutura que permite o escalonamento de tempo real.

5.1.1 Monitor da Carga da Bateria

O EPOS disponibiliza um monitor da carga da bateria [GF07] que contribui para alcançarmos os objetivos deste trabalho. O monitor do EPOS é baseado na observação da tensão das baterias, pois estas têm suas tensões reduzidas conforme a utilização. Entretanto, alguns detalhes devem ser analisados com cautela: a tensão amostrada não é relacionada linearmente com a taxa de descarga da bateria, o sistema não tem

capacidade de converter toda a tensão fornecida em recurso utilizável e também existe uma tensão mínima em que o sistema opera.

A figura 5.1 apresenta o gráfico Tensão X Tempo, no qual pode ser observado que a taxa de diminuição da tensão é variável no tempo. Com base nisso, o monitor estabelece uma relação discreta entre a tensão obtida e a carga da bateria. A relação é dada pela divisão das tensões obtidas em 10 fatias de tempo, chamadas épocas, nas quais as tensões possuem diferentes variações, como apresentado no gráfico. Cada época corresponde a uma porcentagem da capacidade nominal da bateria utilizada.

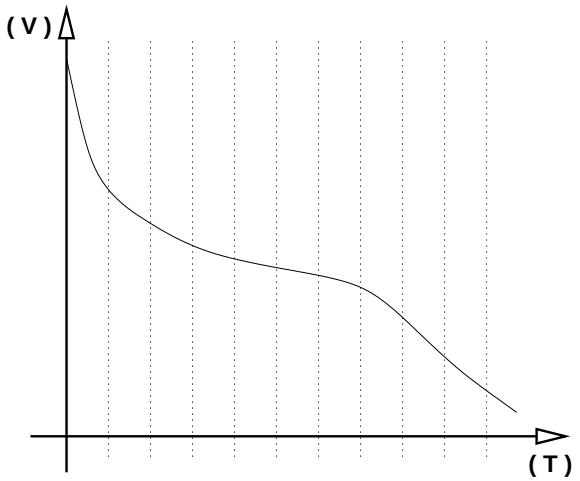

Figura 5.1: Gráfico da Tensão X Tempo

Apesar de ser possível, o monitor do EPOS não realiza um acompanhamento constante da tensão real da bateria, pois cada leitura consome energia e ocasiona um sobrecusto considerável para ser realizada. Para diminuir esses efeitos, o monitor utiliza uma estrutura com informações previamente conhecidas que permitem acompanhar o consumo de energia de uma forma aproximada. A estrutura contém informações a respeito das características específicas da bateria e da energia a ser consumida pelos dispositivos de *hardware* do sistema. Quando o sistema é ligado, o monitor verifica a carga da bateria por meio da tensão e, durante a execução, esse valor é atualizado com as informações das energias consumidas pelos periféricos do sistema obtidas na estrutura.

5.1.2 Escalonamento de Tempo Real

A infra-estrutura de escalonamento de tempo real no EPOS é formada por três componentes centrais: **Thread**, **Criterion** e **Scheduler**. A principal diferença entre a modelagem de tempo real no EPOS e em outros sistemas operacionais de tempo real é relacionada às filas de escalonamento. Essas são ordenadas conforme o objeto **Criterion**, o qual deve ser substituído com as necessidades das aplicações específicas.

A figura 5.2 apresenta o diagrama de classe dos componentes **Criterion** do EPOS. O suporte de tempo real é disponibilizado pela classe **Real-Time**, o qual herda a prioridade e as informações de escalonamento da tarefa. Algoritmos de escalonamento de tarefas periódicas como **RM** (*Rate Monotonic*) e **EDF** (*Earliest Deadline First*) herdam informações de período do componente **Periodic**.

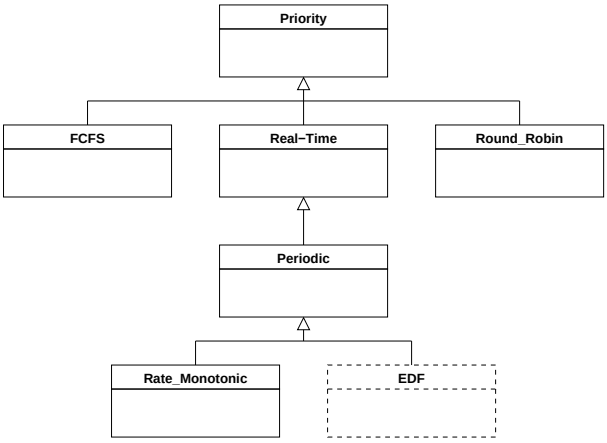

Figura 5.2: Diagrama de classe dos componentes **Criterion**

A separação da política de escalonamento e escalonador permite que o mesmo escalonador seja utilizado com diferentes políticas. Nos casos em que a política requer tratamentos de escalonamento específicos, um novo escalonador pode ser implementado estendendo escalonadores existentes.

5.2 Escalonador Proposto

As características e as funcionalidades do EPOS apresentadas são vantajosas para a implementação desta proposta, além de este sistema ter sido desenvolvido no mesmo grupo de pesquisa. Fundamentado nisso, estendemos o EPOS para suportar o nosso algoritmo de escalonamento com tarefas imprecisas e execuções condicionais aos parâmetros desejados (tempo e energia).

Em um primeiro momento, implementamos o algoritmo de escalonamento EDF no EPOS para garantir os *prazos* das partes obrigatórias das tarefas. Como apresentado, a prioridade do algoritmo EDF é dinâmica, distribuindo maiores prioridades para as tarefas com prazos mais curtos. A partir disso, a prioridade deve ser recalculada a cada ativação da tarefa, que no caso do EDF a prioridade é o *prazo* absoluto (D_{absi} de i) que pode ser obtido pela equação 5.1.

$$D_{absi} = \rho_i + P_i \times \alpha_i + D_i \tag{5.1}$$

Onde, ρ_i é a fase da tarefa i – instante no qual essa tarefa iniciou a execução. α_i é o número de ativações da tarefa i.

Com a utilização da equação 5.1, o valor do *prazo* absoluto e o contador de ativações aumentam a cada período. Em algum momento, as variáveis utilizadas para armazenar esses valores ultrapassarão os limites suportados e a estabilidade do sistema dependerá dos tamanhos das variáveis utilizadas que podem mudar entre arquiteturas. Com isso, o sistema em algum momento irá travar.

Uma técnica implementada neste trabalho que não utiliza o *prazo* absoluto da tarefa para o algoritmo EDF é pela utilização de uma fila relativa ordenada pelo *prazo* relativo da tarefa – nesse caso a prioridade é o *prazo* relativo. A tarefa é inserida na fila com o seu *prazo* e, antes de cada inserção, o valor do *prazo* (*prazo* igual à prioridade) da tarefa na posição de mais alta prioridade (“tarefa na cabeça da fila”) é atualizado de acordo com o tempo decorrido. Na remoção de uma tarefa da fila, o valor do seu *prazo* é ajustado para o original. Dessa forma, obtemos as prioridades dinâmicas do algoritmo EDF.

A figura 5.3 apresenta um exemplo de inserção de três tarefas em uma fila relativa ordenada pelos *prazos*. Com essa técnica, a implementação do algoritmo EDF foi viável e não prejudicou a estabilidade do sistema.

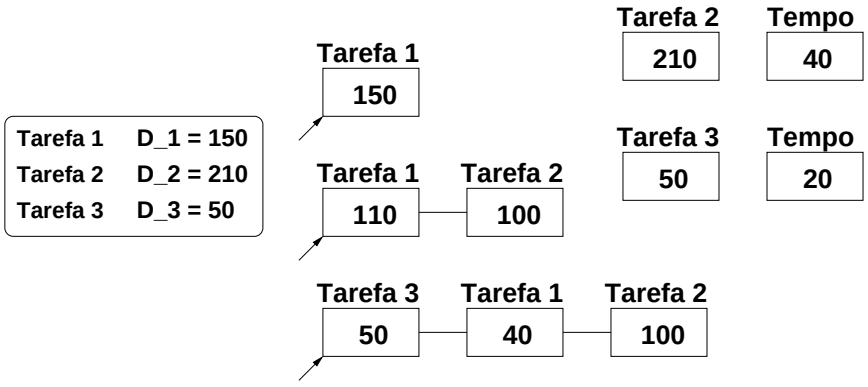

Figura 5.3: Inserção em uma fila relativa ordenada pelos *prazos* das tarefas

Em um segundo momento, foi decidida a modelagem a ser implementada com relação às tarefas imprecisas no EPOS. As tarefas imprecisas no EPOS foram modeladas com base nas funções monotônicas da Computação Imprecisa, pois essa modelagem não possui a restrição 0/1. Nessa modelagem, as funções melhoram a qualidade do resultado durante o tempo que permanecem executando e, na pior das hipóteses, não alteram o resultado, ou seja, a parte obrigatória retorna a solução com o mínimo de QoS necessária para a continuidade da aplicação e a parte opcional faz refinamentos sucessivos nessa solução. O término dessas funções pode ocorrer em qualquer momento da execução sem ocasionar problemas com relação à integridade do resultado. Com isso, o escalonador pode decidir em qualquer instante finalizar a execução da parte opcional. A aplicação pode tratar a integridade dos resultados por intermédio de diferentes métodos, como, por exemplo, o uso de bits de controle ou mesmo o uso de *timestamps* da última atualização dos dados.

A figura 5.4 apresenta as transições de estados da parte obrigatória e da parte opcional. O estado ESPERANDO indica as tarefas que esperam por um novo período. O estado PRONTO indica as tarefas que estão prontas para execução e aguardam o escalonamento. A tarefa no estado EXECUTANDO é aquela que está utilizando o

processador em um determinado momento. Na figura é possível observar que a diferença entre as transições de estados da obrigatória para a opcional é do estado PRONTO para o estado ESPERANDO. Essa diferença indica que a parte opcional pode ser cancelada antes mesmo de ser executada, pois ela estaria no estado PRONTO e não no estado EXECUTANDO. Além disso, a parte opcional pode ser cancelada durante a execução, ou seja, do estado EXECUTANDO para o estado ESPERANDO, mesmo que não tenha terminado a sua execução. A parte obrigatória, por sua vez, só pode ir para o estado ESPERANDO depois de passar pelo estado EXECUTANDO e finalizar a execução no seu período.

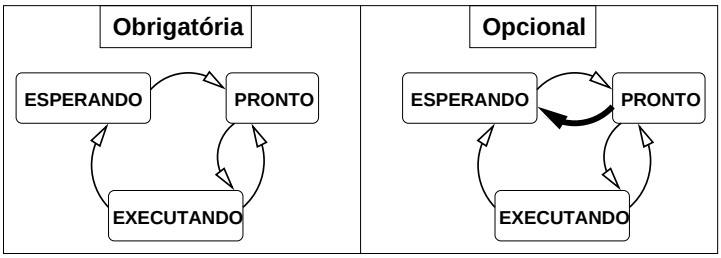

Figura 5.4: Transições de estados da parte obrigatória e da parte opcional

Com base na modelagem definida, um fator importante a ser analisado é como tratar as partes obrigatórias e opcionais das tarefas imprecisas. No EPOS foi possível implementar as partes obrigatórias e as partes opcionais das tarefas imprecisas por meio de fluxos de execução (*threads*). Com isso, cada tarefa imprecisa possui duas *threads*, uma contendo o fluxo de execução da parte obrigatória e outra com o fluxo de execução da parte opcional. O sistema implementado cria essas duas *threads* de uma forma transparente para o programador da aplicação. No momento da criação de uma tarefa imprecisa, o programador especifica apenas dois ponteiros para funções: um para a parte obrigatória e outro para a parte opcional, com os seus respectivos parâmetros.

A figura 5.5 apresenta o diagrama de classe das tarefas imprecisas no EPOS. A classe **Imprecise_Thread** herda informações da classe **Periodic_Thread** responsável pela execução da parte obrigatória. Além disso, a classe **Imprecise_Thread** é composta por um elemento da classe **Aperiodic_Thread** responsável pela parte opcional quando requisitada.

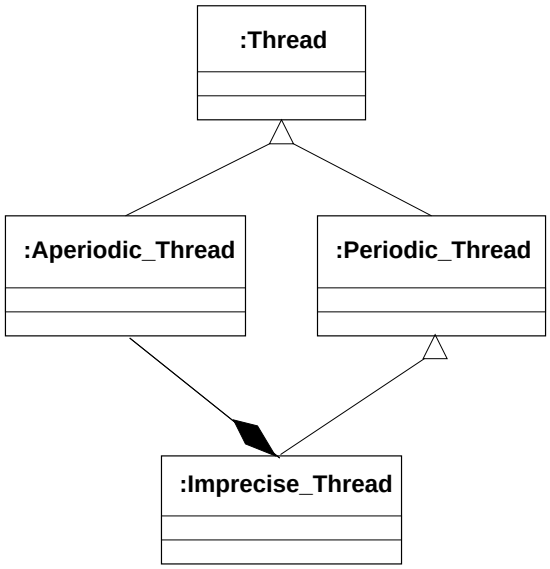

Figura 5.5: Diagrama de Classe das Tarefas Imprecisas

A requisição da parte opcional (entra no estado PRONTO) é realizada no final da execução da sua respectiva parte obrigatória (no método **wait_next** da parte obrigatória). No início de um novo período da tarefa (início de um período da parte obrigatória), a execução da sua parte opcional é suspensa caso não tenha sido terminada (volta para o estado ESPERANDO) e o seu contexto (informações relativas a execução, pilha, registradores) é reiniciado. A figura 5.6 ilustra o método **wait_next** da parte obrigatória. A figura 5.7 apresenta o diagrama de seqüência desse método.

Durante a execução, o escalonador escolhe a *thread* de mais alta prioridade de acordo com o nosso algoritmo baseado no EDF. As partes opcionais possuem prioridades inferiores às partes obrigatórias, sendo, apenas, escalonadas quando não existirem partes obrigatórias no estado PRONTO e, ainda, se houver energia suficiente para atender ao tempo de duração da bateria requerido. Essas características permitem que não ocorram perdas dos *prazos* das partes obrigatórias. A figura 5.8 apresenta o diagrama de seqüência do escalonamento das tarefas do sistema proposto.

A cada π unidades de tempo, o tempo de duração requerido, $T_{t\kappa}$, e a carga restante da bateria, $E_{t\kappa}$, no instante κ, são atualizados pelo tempo decorrido no sistema e pelo monitor de energia do EPOS, respectivamente. Nesses períodos de tempo,

```
class Imprecise_Thread: public Periodic_Thread
{
    //  ...

    static  void wait_next()
    {
        //  ...
        self()->_optional_thread->resume();

        Periodic_Thread::wait_next ();

        self()->_optional_thread->suspend();
        self()->_optional_thread->restore_context();
    }

    //  ...
};
```

Figura 5.6: Código relativo ao método **wait_next** da parte obrigatória

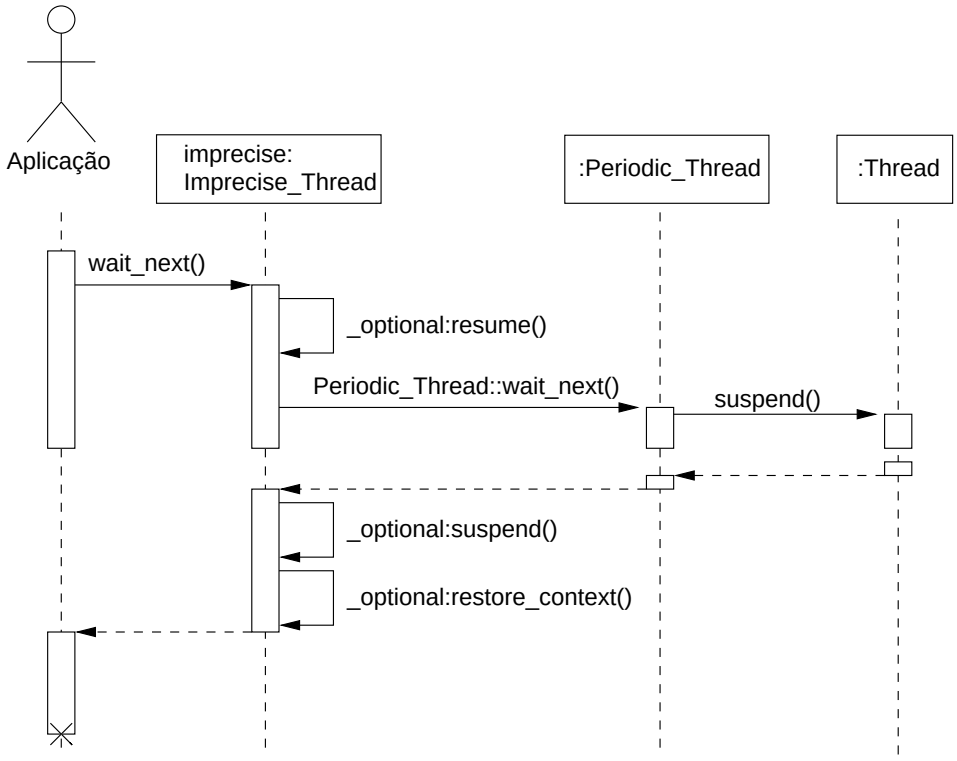

Figura 5.7: Diagrama de seqüência do método **wait_next** da parte obrigatória

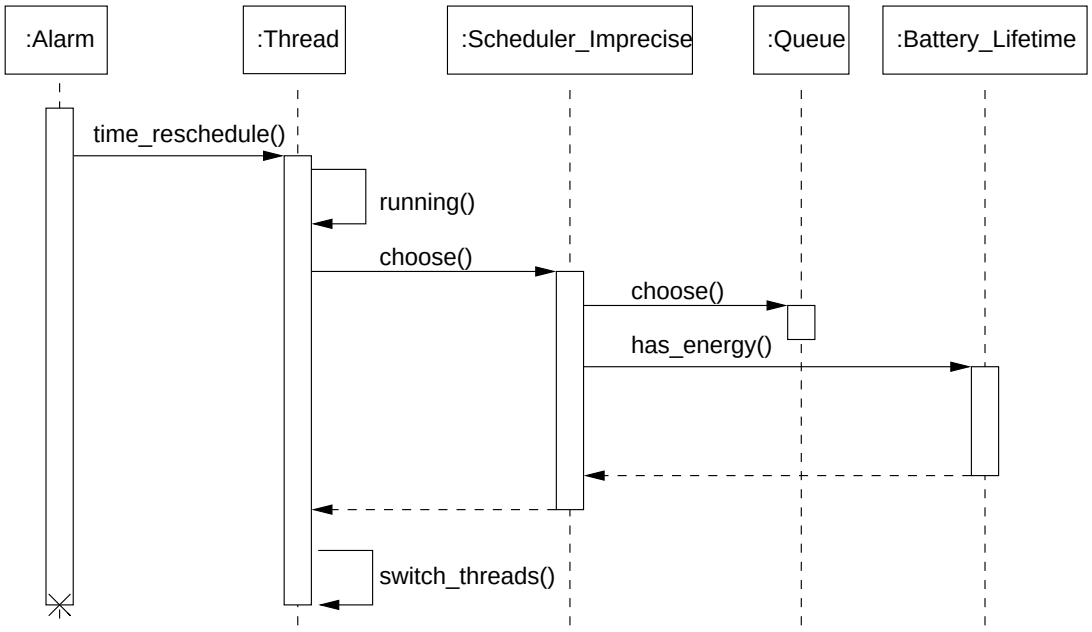

Figura 5.8: Diagrama de seqüência do escalonamento de tarefas

o escalonador realimenta a equação (4.10) com as variáveis atualizadas para determinar se o sistema é capaz de manter a carga de trabalho sem que a bateria termine antes de $T_{t\kappa}$ seja alcançado. O valor de π dependerá do resultado da última análise de energia como mencionado anteriormente.

Todas as partes obrigatórias são executadas e todas as partes opcionais são escalonadas se a equação (4.10) é atendida. Caso contrário, as partes opcionais serão descartadas e o escalonador invoca o gerente de energia que está no modo passivo, aproveitando o tempo ocioso criado para economizar energia. Nesse caso, além da economia de energia por não executar as partes opcionais, o gerente reduz o consumo global do sistema com o uso de técnicas de gerência de energia. O gerente de energia implementado neste trabalho será descrito com maiores detalhes na próxima seção. No instante $\kappa + \iota$ em que o escalonador identifica que $T_{t\kappa+\iota}$ pode ser alcançado, o sistema volta a permitir o escalonamento das partes opcionais.

5.3 Gerente de Energia

Com o objetivo de aproveitar os tempos ociosos criados pelos descartes das partes opcionais, este trabalho estendeu a infra-estrutura de gerência de energia disponível no EPOS para o desenvolvimento de um gerente global à aplicação para sistemas embarcados. Esse gerente permite que, no momento da geração do sistema, o programador configure qual será o algoritmo de gerência de energia a ser utilizado, por meio de heurísticas "replugáveis". Essa configurabilidade é importante, pois o algoritmo de gerência é um dos principais fatores de influência no equilíbrio entre economia de energia e sobrecustos para a aplicação. Dessa forma, a configuração do algoritmo de gerência permite que o programador da aplicação decida qual o algoritmo de gerência do consumo de energia seu sistema suporta, sem que ocasione interferências significativas para a aplicação.

A seção 5.3.1 descreve um algoritmo de gerência de energia implementado no gerente de energia, enquanto, a seção 5.3.2 apresenta os modos de operação do gerente.

5.3.1 Algoritmo do Gerente de Energia

Como descrito, sistemas embarcados não podem arcar com os custos de algoritmos complexos de gerência de energia. Uma estratégia simples para a gerência de energia dinâmica faz uso de um escalonador no sistema operacional. No momento em que o escalonador não tiver tarefas a serem escalonadas, o processador principal do sistema é colocado no modo *standby*. Porém, essa técnica é muito limitada, pois não considera componentes periféricos, os quais em sistemas embarcados usualmente consomem mais energia do que o micro-controlador principal. Para contemplar esses dispositivos, um gerente de energia é requerido. Esse gerente verifica a utilização de cada componente e, se um certo componente está inativo por um determinado período de tempo, ele altera o modo de operação desse componente para um estado mais econômico com relação à energia.

Uma técnica para detectar a ociosidade de componentes utiliza "conta-

dores" de uso em *hardware* para cada componente. Entretanto, as plataformas de sistemas embarcados tipicamente não possuem essa técnica em *hardware*, e uma infra-estrutura baseada em *software* deve ser usada. Este trabalho implementou um algoritmo de gerência de energia baseado em "contadores" de acesso que objetiva interferir o mínimo possível na execução da aplicação.

A figura 5.9 é um diagrama de tempo da UML que representa o algoritmo de gerência de energia implementado. Os estágios do algoritmo são apresentados desde os testes em busca de um tempo ocioso do componente, a verificação de ociosidade e até troca do modo de operação do dispositivo que, nesse caso, é a migração do modo ligado para o modo desligado do componente UART. Os estados e valores da *thread* da aplicação, da *thread* do gerente de energia e do componente UART são representados nessa figura. A UART foi utilizada como um exemplo de componente a ser monitorado pelo gerente.

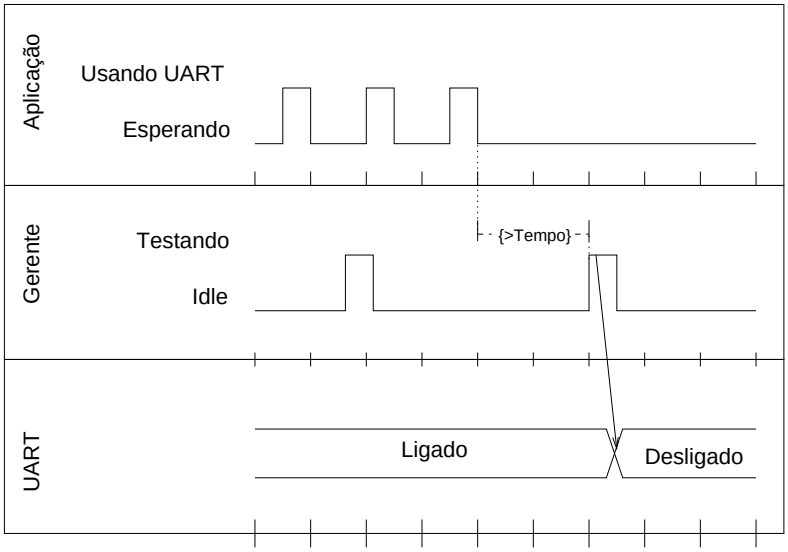

Figura 5.9: Algoritmo de gerência do consumo de energia

Em termos gerais, o funcionamento do algoritmo ocorre da seguinte forma:

- A data (*timestamp*) de utilização de cada componente monitorado é atualizada a cada acesso a esse componente ("contadores do acesso").

- O gerente de energia é executado em intervalos de tempo configurado pelo programador.

- No momento da execução do gerente, é obtida a data corrente (*timestamp*). Um cálculo dessa data menos a data armazenada do último acesso ao componente é realizado.

- O modo de operação do componente é alterado caso o resultado obtido seja maior do que um valor pré-definido (*limite*), pois esse valor indica ao gerente que o componente está ocioso.

5.3.2 Modos de Operação do Gerente

Neste trabalho, o gerente de energia foi implementado como um aspecto para permitir que ele seja isolado do sistema. Isso possibilitou o desenvolvimento de diferentes características de modos de operação que são configurados pelo programador da aplicação na geração do sistema:

- O programador da aplicação pode escolher se o gerente será habilitado para realizar a gerência de energia ou não. Com a ativação do gerente, a aplicação não necessita utilizar interfaces de gerência de energia.

- Configurar apenas os componentes que a aplicação deseja que o gerente de energia monitore, com objetivo de atender aos requisitos de cada aplicação específica. Com isso, a gerência de energia de componentes indesejados pela aplicação não é adicionada no sistema final, apesar de ter sido implementada.

- Escolher se o gerente será ativo ou passivo no sistema. Na gerência ativa, o gerente executa ativamente em intervalos de tempo configurado pelo programador. No modo passivo, outro componente deve invocar a execução do gerente.

5.4 Estudo de Caso

Neste estudo de caso, utilizamos os módulos de sensoriamento Mica Mote2 [HHKK04]. O Mica2 é um módulo de sensoriamento alimentado por baterias para redes de sensores sem fio desenvolvido pela empresa CrossBow. Esse dispositivo possui um microcontrolador AVR, ATMEGA128, produzido pela ATMEL. O AVR é um microcontrolador RISC (*Reduced Instruction Set Computer*) de 8 bits baseado na arquitetura de *Harvard*. Além disso, o Mica2 é composto por um *transceiver* de rádio Chipcon CC1000 e um conjunto de sensores como, por exemplo, luminosidade, temperatura, aceleração.

A aquisição de dados de sensores é um exemplo de uma aplicação que pode se beneficiar da Computação Imprecisa. A média de múltiplas leituras de uma fonte de dados não confiável pode aumentar a precisão comparada a uma simples leitura da mesma fonte. Entretanto, leituras múltiplas podem ocasionar um excessivo tempo de execução e consumo de energia. O escalonador impreciso pode escolher quando realizar múltiplas leituras para aumentar a precisão e quando realizar uma simples leitura para economizar recursos do sistema.

Modelamos uma aplicação de aquisição de dados de sensores que periodicamente lê dados de um sensor de temperatura e envia esses dados para a rede. A figura 5.10 esboça um diagrama de seqüência que ilustra o comportamento dessa aplicação. Seguindo nosso modelo da Computação Imprecisa, a parte obrigatória da nossa tarefa de sensoriamento realiza uma única leitura. Caso existam tempo e energia suficientes disponíveis no sistema, a parte opcional realiza uma média de múltiplas leituras. Finalmente, uma tarefa de comunicação envia os dados lidos, ou da única leitura obrigatória, ou da média das múltiplas leituras.

Essa aplicação foi implementada no EPOS utilizando o nosso modelo da Computação Imprecisa. A figura 5.11 ilustra as APIs do sistema usadas pela aplicação. A função **main** cria um sensor de temperatura e um link de comunicação [WHdOF06, WdOF07], posteriormente, cria uma *thread* imprecisa (**Imprecise_Thread**) e uma *thread* periódica (**Periodic_Thread**), as quais representam as tarefas de sensoriamento e de comunicação, respectivamente. A parte obrigatória da tarefa de aquisição de dados realiza

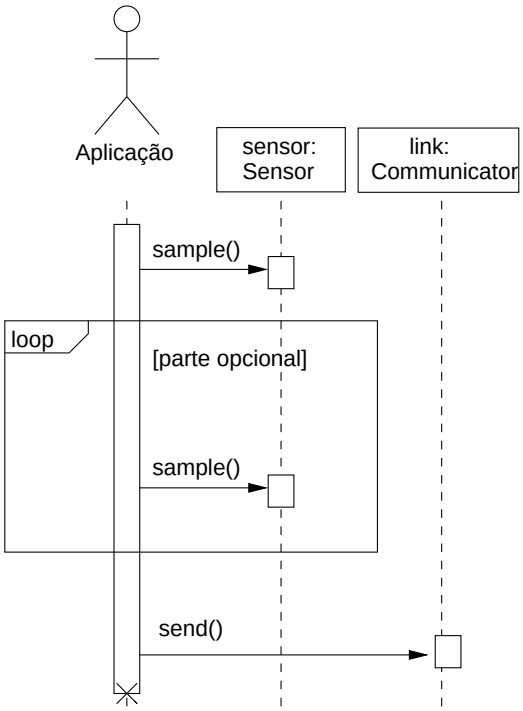

Figura 5.10: Diagrama de seqüência da aplicação de aquisição de dados

uma única leitura do sensor e aguarda pelo seu próximo período de execução. Quando executada, a parte opcional realiza múltiplas leituras (neste caso, 10 leituras se possível) fazendo a média de todas as leituras anteriores. Quando finalizada, a parte opcional libera também o processador e aguarda pelo próximo período. Finalmente, a tarefa de comunicação envia os dados coletados, os quais foram atualizados por último pela parte obrigatória ou pela parte opcional.

A *thread* imprecisa (**Imprecise_Thread**) recebe suas duas funções, obrigatória e opcional, como parâmetro, enquanto a *thread* periódica (**Periodic_Thread**) recebe apenas uma função. Ambas as *threads* recebem um **Criterion** como parâmetro, o qual especifica o *prazo*, o período, o número de execuções e a fase de cada *thread*. O *prazo* de cada *thread* é estabelecido para que a ordem de execução da tarefa de sensoriamento (parte obrigatória e parte opcional) e da tarefa de comunicação seja preservada. Todas as *threads* executam por um infinito número de vezes. A tabela 5.1 apresenta os valores do **Criterion** (critérios de escalonamento) para ambas as *threads*.

```
void sensor_main() {
    // ...
    sensor = new Temperature_Sensor();
    link = new Communicator(SENSOR_SINK);
    Imprecise_Thread sensing(&mandatory_sensor,
        &optional_sensor, Criterion(150e3,170e3,INFINITE,0));
    Periodic_Thread communication(&send_data,
        Criterion(20e3,170e3,INFINITE,150e3), SUSPENDED);
}

int mandatory_sensor() {
    while(1) {
        // ...
        temperature = sensor->sample();
        Imprecise_Thread::wait_next();
    }
}

int optional_sensor() {
    while(1) {
        // ...
        for (int i = 1; i <= MAX_SAMPLES; i++){
            sample = sensor->sample();
            temperature =
                (temperature * i + sample) / (i+1);
        }
        Optional_Thread::wait_next();
    }
}

int send_data() {
    while(1) {
        // ...
        link->write(temperature);
        Periodic_Thread::wait_next();
    }
}
```

Figura 5.11: APIs do sistema usadas pela aplicação de aquisição de dados

Tabela 5.1: Critérios de escalonamento.

Thread	Fase (ms)	*prazo* (ms)	Período (ms)	N. Vezes
Sensoriamento	0	150	170	∞
Comunicação	150	20	170	∞

A figura 5.12 apresenta o diagrama de escalonamento para essa aplicação. Os *prazos* estabelecidos para ambas as *threads* são ilustrados nessa figura.

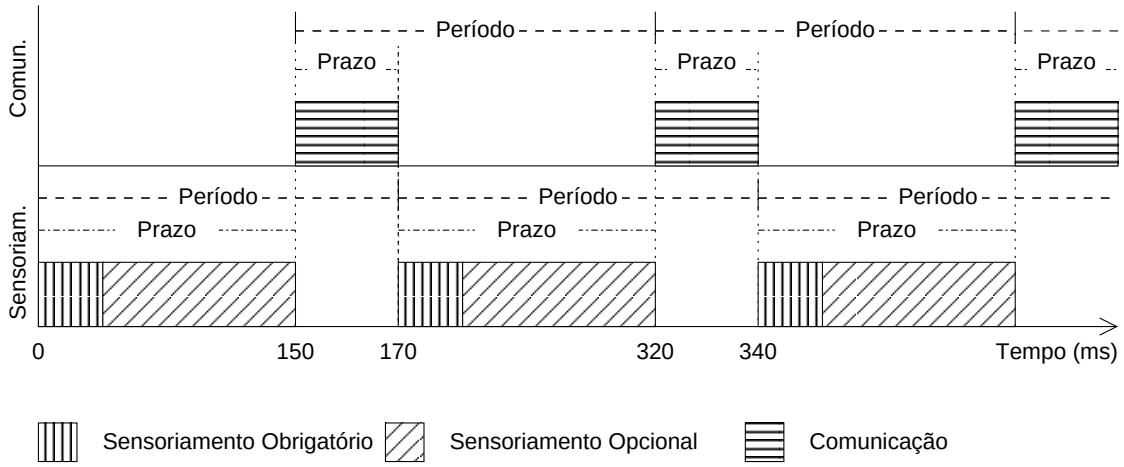

Figura 5.12: Diagrama de escalonamento

Com o objetivo de verificar a escalonabilidade em tempo de projeto, traçamos o perfil da aplicação com relação ao tempo e à energia. A energia medimos por meio da multiplicação do tempo pela potência. A potência obtivemos pela leitura de dois canais de um osciloscópio digital. O primeiro canal lia a tensão da bateria e o segundo lia a corrente elétrica. Um resistor do tipo *shunt* de 1 ohm foi utilizado no sistema para o monitoramento da corrente elétrica. A tabela 5.2 apresenta o tempo de execução e o consumo de energia nos piores casos para a execução das tarefas da nossa aplicação e o sobrecusto total do sistema para um período de execução. Esses valores representam a média de 9000 leituras.

Nesse exemplo, o nosso orçamento com relação à energia foi a carga

Tabela 5.2: Tempo de execução e consumo de energia nos piores casos.

Parte	Tempo (ms)	Energia (J)
Obrigatória	11,683	0,0004254
Opcional	116,831	0,0042543
sobrecusto do sistema	0,138	0,0098289

provida por duas baterias normais AA com a capacidade total de 58320 J (5400 mAh em 3 V). O sistema foi configurado com um tempo de duração requerido de 11 dias, ou 950400000 ms. Nesse caso, é importante observar que, para melhor ilustrar o consumo de energia (exaustão da carga da bateria), nossa aplicação intencionalmente utiliza o sistema com carga total. Muitas aplicações tipicamente executariam com pouca carga (*lower duty-cycle*) e, com isso, executariam por um período de tempo mais longo. Para verificar a escalonabilidade da aplicação com relação ao tempo, aplicamos os valores na equação (4.1):

$$\begin{aligned} \sum_{i=1}^{n} \left(\frac{\mu_i}{D_i} \right) + \sigma &\leq \omega \\ \left(\frac{11,683 \text{ ms}}{150 \text{ ms}} \right) + \frac{0,138 \text{ ms}}{150 \text{ ms}} &\leq 1 \\ 0,078 &\leq 1 \end{aligned} \tag{5.2}$$

Se (5.2) é verdadeira, as partes obrigatórias são escalonáveis com relação ao tempo. A utilização da equação (4.2), apresentada em (5.3), ilustra que ambas as partes são escalonáveis no que se refere ao tempo.

$$\begin{aligned} \sum_{i=1}^{n} \left(\frac{\mu_i + \theta_i}{D_i} \right) + \sigma &\leq \omega \\ \left(\frac{(11,683 + 116,831) \text{ ms}}{150 \text{ ms}} \right) + \frac{0,138 \text{ ms}}{150 \text{ ms}} + \frac{0,138 \text{ ms}}{150 \text{ ms}} &\leq 1 \\ 0,8586 &\leq 1 \end{aligned} \tag{5.3}$$

Com o objetivo de verificar a escalonabilidade da aplicação com relação à energia, aplicamos os valores da aplicação na equação (4.5):

$$
\begin{aligned}
\sum_{i=1}^{n}\left(\frac{E_{\mu i}\times T_t}{P_i\times E_t}\right)+\epsilon &\leq 1\\
\left(\frac{425,4\times 10^{-6}\mathrm{J}\times 950,4\times 10^{6}\mathrm{ms}}{170\ \mathrm{ms}\times 58320\ \mathrm{J}}\right)+&\\
\left(\frac{9828,9\times 10^{-6}\mathrm{J}\times 950,4\times 10^{6}\mathrm{ms}}{170\ \mathrm{ms}\times 58320\ \mathrm{J}}\right) &\leq 1\\
\left(\frac{425,4\times 950,4}{170\times 58320}\right)+\left(\frac{9828,9\times 950,4}{170\times 58320}\right) &\leq 1\\
0,0407791+0,9422039 &\leq 1\\
0,9829830 &\leq 1
\end{aligned}
\tag{5.4}
$$

Se (5.4) é verdadeira, as tarefas são escalonáveis com relação à energia. Contudo, a verificação da equação (4.6) mostra que algumas partes opcionais serão descartadas devido às restrições de energia, como pode ser observado em (5.5):

$$
\begin{aligned}
\sum_{i=1}^{n}\left(\frac{(E_{\mu i}+E_{\theta i})\times T_t}{P_i\times E_t}\right)+\epsilon &\leq 1\\
\left(\frac{(425,4+4254,3)\times 10^{-6}\mathrm{J}\times 950,4\times 10^{6}\mathrm{ms}}{170\ \mathrm{ms}\times 58320\ \mathrm{J}}\right)+&\\
\left(\frac{9828,9\times 10^{-6}\ \mathrm{J}\times 950,4\times 10^{6}\mathrm{ms}}{170\ \mathrm{ms}\times 58320\ \mathrm{J}}\right)&\\
\left(\frac{4679,7\times 950,4}{170\times 58320}\right)+\left(\frac{9828,9\times 950,4}{170\times 58320}\right)&\\
0,4485987+0,9422039&\\
1,3908026 &\not\leq 1
\end{aligned}
\tag{5.5}
$$

Como a verificação da equação (4.6) falha com a nossa aplicação, apresentada em (5.5), o escalonador decidirá em tempo de execução quando executar as partes opcionais. Em cada reescalonamento, o escalonador verifica se a próxima tarefa é uma tarefa de tempo real (tarefa obrigatória) por meio da sua prioridade. Se o teste é verdadeiro, ela recebe o processador. Caso contrário, a tarefa é opcional e recebe o processador somente se existe energia suficiente para atender ao tempo de duração do sistema desejado. Se ela é opcional e a energia não é suficiente, o escalonador invoca o gerente de ener-

gia oportunista, que está em modo passivo, para colocar o processador e os componentes periféricos em modos econômicos de energia.

A figura 5.13 apresenta a potência utilizada pela nossa aplicação ao longo do tempo. Na curva com os valores de potências mais altos, o sistema é configurado para executar todas as partes das tarefas (parte obrigatória e opcional). Na curva com os valores de potências mais baixos, o sistema é configurado para executar somente as partes obrigatórias. A curva entre as anteriores representa o uso do nosso modelo da Computação Imprecisa, no qual as tarefas opcionais somente são executadas se o sistema está dentro de seu orçamento de energia, isto é, atende ao tempo de duração da bateria requerido.

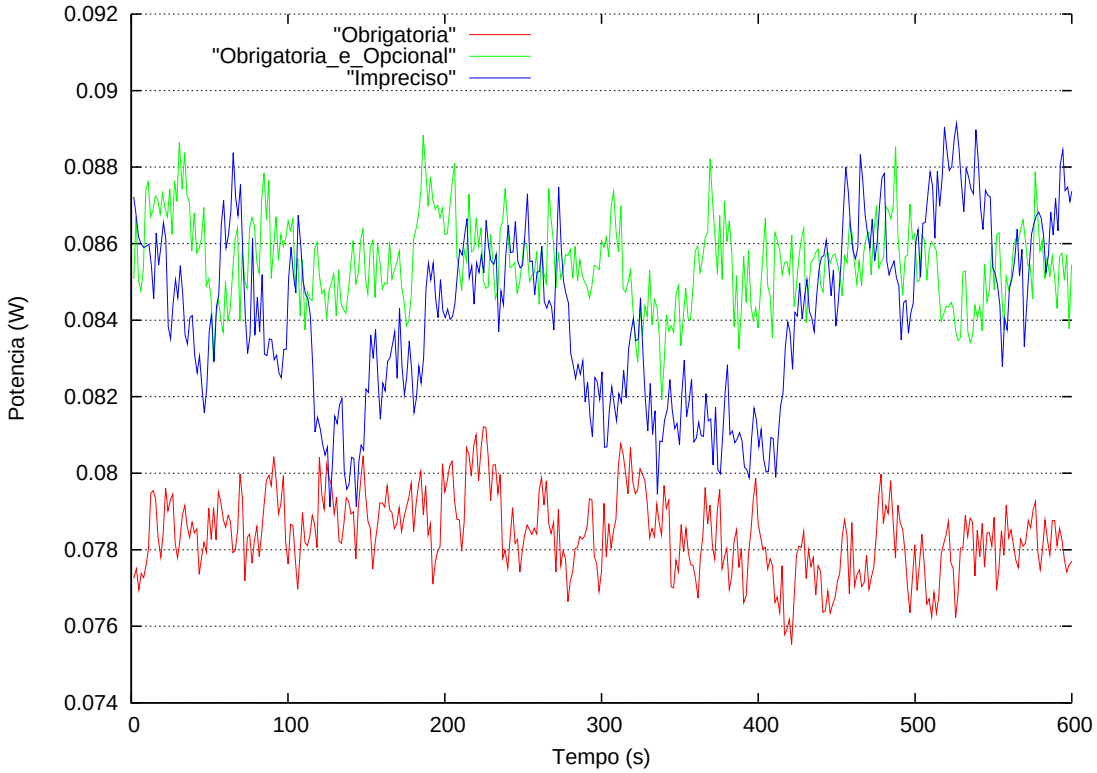

Figura 5.13: Potência ao longo do tempo

A figura 5.14 apresenta diferentes níveis de QoS ao longo do tempo para o nosso modelo da Computação Imprecisa. Nesse gráfico, o nível de QoS é definido como o número de leituras do sensor realizadas em um dado período pela parte opcional, fração executada. Quando existe energia suficiente no sistema, as partes opcionais são escalonadas, aumentando a qualidade de serviço. Quando os níveis de energia são mais

baixos do que o esperado, as partes opcionais são descartadas. Os descartes das partes opcionais podem ocorrer durante a execução, resultando em níveis intermediários de QoS, como, por exemplo, quando somente duas leituras do sensor foram realizadas.

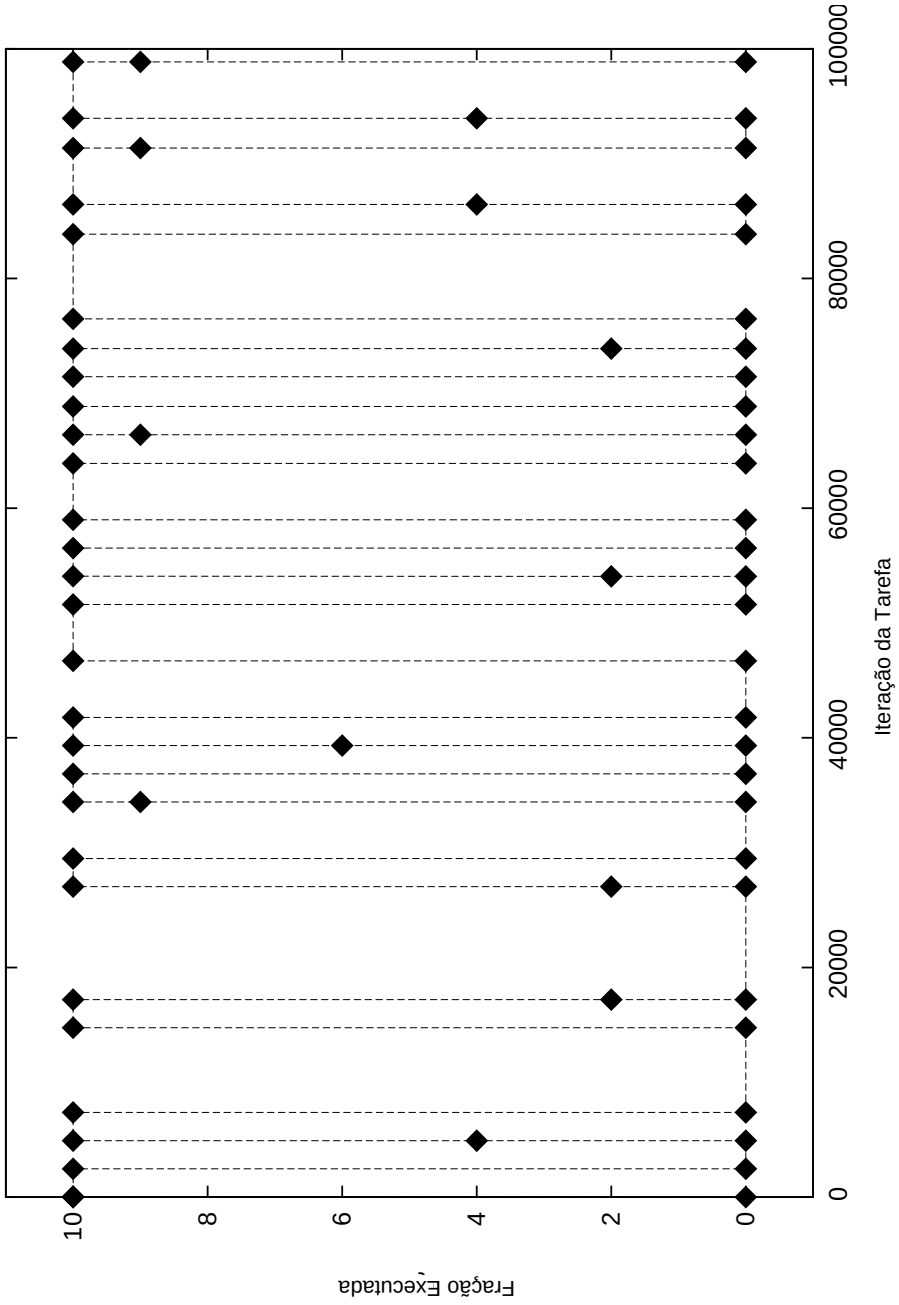

Figura 5.14: Níveis de QoS ao longo do tempo

Capítulo 6

Considerações Finais

Este trabalho está inserido no contexto dos sistemas computacionais embarcados. Segundo pesquisas realizadas por Tennenhouse, mais de 98% dos processadores fabricados no mundo estão em sistemas embarcados. Normalmente, esses sistemas apresentam limitações de recursos em termos da capacidade de processamento e de memória e, em muitos casos, são alimentados por baterias com uma limitada carga de energia. Considerando-se esses fatores, é importante serem eles capazes de gerenciar seus próprios consumos de energia sem prejudicar a execução.

As técnicas de gerência de energia permitem que certos componentes do sistema sejam ligados ou desligados dinamicamente a fim de permitir que o sistema possa economizar energia quando esses componentes não são utilizados. Além disso, técnicas como *Dynamic Voltage Scaling* (DVS) e hibernação de recursos foram introduzidas para permitir que alguns componentes operem com diferentes níveis de energia ao longo do tempo.

Essas técnicas são essenciais para a modelagem de sistemas embarcados capazes de gerenciar seus próprios consumos de energia; entretanto, elas, freqüentemente, ocasionam sobrecusto e/ou latência para o sistema. Esses fatores não podem ser ignorados em sistemas que buscam garantir as restrições de tempo real e/ou de QoS. Por outro lado, frisamos não ser suficiente garantir as restrições de tempo real e/ou de QoS se, fazendo isso, o sistema consome o orçamento de energia disponível (descarrega a bateria) e não é

capaz de completar suas tarefas importantes.

Neste trabalho, utilizamos o orçamento de energia da aplicação, isto é, o tempo de duração da bateria esperado, como um parâmetro de QoS, que denominamos QoS em termos de energia. Nosso objetivo não foi apenas economizar energia, mas aumentar a utilidade da aplicação, garantindo o tempo de duração da bateria desejado e, ainda, preservando os *prazos* das tarefas *hard real-time*. Para isso, o controle de QoS foi inspirado na Computação Imprecisa que divide cada tarefa em parte obrigatória e em parte opcional. Com o monitoramento do consumo de energia, nosso escalonador é capaz de detectar quando os níveis de energia não são suficientes para atender ao tempo de duração especificado. Caso seja constatada a falta de energia, o escalonador realiza diminuições dos níveis de QoS pela prevenção da execução de partes opcionais com o objetivo de reduzir o consumo de energia.

Disponibilizamos equações para a análise em tempo de projeto da escalonabilidade do conjunto das tarefas em termos dos *prazos* e do orçamento de energia. Em tempo de execução, nosso escalonador preemptivo verifica periodicamente os níveis de energia para assegurar que as partes opcionais somente executem quando há energia suficiente para que a carga da bateria dure pelo tempo especificado pela aplicação. Esse controle dos níveis de QoS cria períodos ociosos no sistema, nos quais o escalonador invoca um gerente de energia oportunista, também desenvolvido neste trabalho, para reduzir o consumo de energia de certos componentes. Com isso, além de reduzir a demanda por processamento e, assim, o consumo de energia, este trabalho utiliza técnicas de gerência de energia para atender aos parâmetros desejados.

Um protótipo foi implementado com o intuito de validar a abordagem de escalonamento proposta neste trabalho, usando-se o EPOS. O algoritmo EDF foi implementado para garantir os *prazos* das partes obrigatórias das tarefas, e uma modelagem das tarefas imprecisas foi selecionada e implementada no EPOS. Com o objetivo de aproveitar os tempos ociosos criados com os descartes das partes opcionais, estendemos a infra-estrutura de gerência de energia disponível no EPOS para o desenvolvimento de um gerente global à aplicação. Com isso, fornecemos suporte ao nosso algoritmo de escalonamento com tarefas imprecisas e execuções condicionais aos parâmetros desejados (tempo

e energia). Como demonstrado pelo nosso estudo de caso, esse automático e adaptativo controle dos níveis de QoS permite que as aplicações alcancem seus compromissos entre qualidade de serviço e consumo de energia.

Como sugestão para trabalho futuro, este sistema poderia também ser utilizado para a gerência de QoS em uma rede de sensores. Por exemplo, em uma aplicação que usa “freshness of information” como principal parâmetro de QoS, nosso sistema seria utilizado para dinamicamente atualizar as informações dos sensores de acordo com a energia disponível. Com isso, cada nodo decidiria, baseado em seus recursos de tempo e de energia disponíveis, se atualiza a leitura do sensor ou utiliza a leitura anterior. Nesse exemplo, a qualidade de serviço das aplicações das redes de sensores seria automaticamente adaptada de acordo com os recursos disponíveis no sistema. Outra sugestão, poderia ser a implementação de algoritmos mais complexos para o gerente com objetivo de economizar mais energia nos períodos ociosos.

Referências Bibliográficas

[ANF04] Manish Anand, Edmund B. Nightingale, and Jason Flinn. Ghosts in the machine: Interfaces for better power management. In *Proceedings of the Second International Conference on Mobile Systems, Applications, and Services (MOBISYS'04)*, June 2004.

[BBM98] Luca Benini, Alessandro Bogliolo, and Giovanni De Micheli. Dynamic power management of electronic systems. In *ICCAD '98: Proceedings of the 1998 IEEE/ACM international conference on Computer-aided design*, pages 696–702, New York, NY, USA, 1998. ACM Press.

[BR03] Bishop Brock and Karthick Rajamani. Dynamic power management for embedded systems. In *Proceedings of the IEEE International SOC Conference*. IEEE, IEEE, sep 2003.

[CC96] Intel Corporation and Microsoft Corporation. *Advanced Power Management (APM) BIOS Interface Specification*, 1.2 edition, Feb 1996.

[CCC+04] Hewlett-Packard Corporation, Intel Corporation, Microsoft Corporation, Phoenix Technologies Ltd., and Toshiba Corporation. *Advanced Configuration and Power Interface Specification*, 3.0 edition, Sep 2004.

[CCLH04] Deming Chen, Jason Cong, Fei Li, and Lei He. Low-power technology mapping for FPGA architectures with dual supply voltages. In *FPGA '04: Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays*, pages 109–117, New York, NY, USA, 2004. ACM Press.

[DKB95] Fred Douglis, Padmanabhan Krishnan, and Brian Bershad. Adaptive disk spin-down policies for mobile computers. In *Proc. 2nd USENIX Symp. on Mobile and Location-Independent Computing*, 1995.

[dO97] Rômulo Silva de Oliveira. Computação imprecisa. *RITA*, 4(1):87–106, 1997.

[EGC04] Jo C. Ebergen, Jonathan Gainsley, and Paul Cunningham. Transistor sizing: How to control the speed and energy consumption of a circuit. In *ASYNC*, pages 51–61. IEEE Computer Society, 2004.

[Frö01] Antônio Augusto Fröhlich. *Application-Oriented Operating Systems*. GMD - Forschungszentrum Informationstechnik, Sankt Augustin, 2001.

[FS99a] Jason Flinn and M. Satyanarayanan. Energy-aware adaptation for mobile applications. In *SOSP '99: Proceedings of the seventeenth ACM symposium on Operating systems principles*, pages 48–63, New York, NY, USA, 1999. ACM Press.

[FS99b] Jason Flinn and M. Satyanarayanan. Powerscope: A tool for profiling the energy usage of mobile applications. *wmcsa*, 00:2, 1999.

[GF07] Raphael Tucunduva Gonçalves and Antônio Augusto Fröhlich. Monitoramento da Carga de Baterias em Sistemas Embarcados. Trabalho de diplomação (Bacharelado em Ciência da Computação), Federal University of Santa Catarina, Florianópolis, 2007.

[HHKK04] Jason Hill, Mike Horton, Ralph Kling, and Lakshman Krishnamurthy. The platforms enabling wireless sensor networks. *Commun. ACM*, 47(6):41–46, 2004.

[HLS96] David P. Helmbold, Darrell D. E. Long, and Bruce Sherrod. A dynamic disk spin-down technique for mobile computing. In *MobiCom '96: Proceedings of the 2nd annual international conference on Mobile computing and networking*, pages 130–142, New York, NY, USA, 1996. ACM Press.

[HUN06] Fumiko Harada, Toshimitsu Ushio, and Yukikazu Nakamoto. Power-aware resource allocation with fair qos guarantee. In *RTCSA '06: Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 287–293, Washington, DC, USA, 2006. IEEE Computer Society.

[HWF06] Arliones Stevert Jr. Hoeller, Lucas Francisco Wanner, and Antônio Augusto Fröhlich. A Hierarchical Approach For Power Management on Mobile Embedded Systems. In *5th IFIP Working Conference on Distributed and Parallel Embedded Systems*, pages 265–274, Braga, Portugal, October 2006.

[JWdO+06] Arliones Stevert Hoeller Junior, Lucas Francisco Wanner, Augusto Born de Oliveira, Roger Immich, and Antônio Augusto Fröhlich. Gerenciamento de Energia em Sistemas de Sensoriamento Remoto. In *Third Brazilian Workshop on Operating System*, pages 46–58, Campo Grande, Brazil, July 2006.

[JWL+03] W.-B. Jone, J. S. Wang, Hsueh-I Lu, I. P. Hsu, and J.-Y. Chen. Design theory and implementation for low-power segmented bus systems. *ACM Trans. Des. Autom. Electron. Syst.*, 8(1):38–54, 2003.

[KGMS97] J. Kin, M. Gupta, and W. H. Mangione-Smith. The filter cache: An energy efficient memory structure. In *Proceedings of the 30th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-97)*, pages 184–193, Los Alamitos, December 1–3 1997. IEEE Computer Society.

[KGS04] Eren Kursun, Soheil Ghiasi, and Majid Sarrafzadeh. Transistor level budgeting for power optimization. In *ISQED*, pages 116–121. IEEE Computer Society, 2004.

[KK98] Robin Kravets and P. Krishnan. Power management techniques for mobile communication. In *MobiCom '98: Proceedings of the 4th annual ACM/I-*

EEE international conference on Mobile computing and networking, pages 157–168, New York, NY, USA, 1998. ACM Press.

[KKJ01] B. Konh, S. Kim, and Y. Jun. Conditional-capture flip-flop for statistical power reduction. *IEEE Journal of Solid-State Circuits*, 36(8):1263–1271, 2001.

[KLM+97] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-Oriented Programming. In *Proceedings of the European Conference on Object-oriented Programming'97*, volume 1241 of *Lecture Notes in Computer Science*, pages 220–242, Jyväskylä, Finland, June 1997. Springer.

[LFZE00] Alvin R. Lebeck, Xiaobo Fan, Heng Zeng, and Carla Ellis. Power aware page allocation. In *ASPLOS-IX: Proceedings of the ninth international conference on Architectural support for programming languages and operating systems*, pages 105–116, New York, NY, USA, 2000. ACM Press.

[LL73] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, 1973.

[LNL87a] K. J. Lin, S. Natarajan, and J. W. S. Liu. Imprecise results: Utilizing partial computations in real-time systems. In *Proc. IEEE Real-Time System Symp.*, 1987.

[LNL87b] K. J. Lin, S. Natarajan, and J. W. S. Liu. Scheduling real-time, periodic jobs using imprecise results. In *Proc. IEEE Real-Time System Symp.*, 1987.

[LSL+94] Jane W.S. Liu, Wei-Kuan Shih, Kwei-Jay Lin, Riccardo Bettati, and Jen-Yao Chung. Imprecise computations. *Proceedings of the IEEE*, 82(1):83–94, Jan 1994.

[LSPS05] Daniel Lohmann, Wolfgang Schröder-Preikschat, and Olaf Spinczyk. Functional and non-functional properties in a family of embedded operating sys-

tems. In *Proceedings of the Tenth IEEE International Workshop on Object-oriented Real-time Dependable Systems*, Sedona, USA, Feb 2005. IEEE-Press.

[LWG05] Olaf Landsiedel, Klaus Wehrle, and Stefan Gotz. Accurate prediction of power consumption in sensor networks. In *EmNetS-II: Proceedings of The Second IEEE Workshop on Embedded Networked Sensors*. IEEE, May 2005.

[MJWF06] Hugo Marcondes, Arliones Stevert Hoeller Junior, Lucas Francisco Wanner, and Antônio Augusto Fröhlich. Operating Systems Portability: 8 bits and beyond. In *11th IEEE International Conference on Emerging Technologies and Factory Automation*, pages 124–130, Prague, Czech Republic, September 2006.

[NQ05] Linwei Niu and Gang Quan. A hybrid static/dynamic dvs scheduling for real-time systems with (m, k)-guarantee. *rtss*, 0:356–365, 2005.

[PF04] Fauze Valério Polpeta and Antônio Augusto Fröhlich. Hardware Mediators: a Portability Artifact for Component-Based Systems. In *International Conference on Embedded and Ubiquitous Computing*, volume 3207 of *Lecture Notes in Computer Science*, pages 271–280, Aizu, Japan, August 2004. Springer.

[PLS01] Johan Pouwelse, Koen Langendoen, and Henk Sips. Dynamic voltage scaling on a low-power microprocessor. In *MobiCom '01: Proceedings of the 7th annual international conference on Mobile computing and networking*, pages 251–259, New York, NY, USA, 2001. ACM Press.

[PS01] Padmanabhan Pillai and Kang G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 89–102, New York, NY, USA, 2001. ACM Press.

[SHrC+04] Victor Shnayder, Mark Hempstead, Bor rong Chen, Geoff Werner Allen, and Matt Welsh. Simulating the power consumption of large-scale sensor network applications. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 188–200, New York, NY, USA, 2004. ACM Press.

[SK97] Mark Stemm and Randy H. Katz. Measuring and reducing energy consumption of network interfaces in hand-held devices. *IEICE Transactions on Communications*, E80-B(8):1125–31, 1997.

[SL04] Claudio Scordino and Giuseppe Lipari. Using resource reservation techniques for power-aware scheduling. In *EMSOFT '04: Proceedings of the 4th ACM international conference on Embedded software*, pages 16–25, New York, NY, USA, 2004. ACM Press.

[SN02] Roger R. Schmidt and Budy D. Notohardjono. High-end server low-temperature cooling. *IBM Journal of Research and Development*, 46(6):739–752, 2002.

[Ten00] David Tennenhouse. Proactive computing. *Commun. ACM*, 43(5):43–50, 2000.

[VF05] Vasanth Venkatachalam and Michael Franz. Power reduction techniques for microprocessor systems. *ACM Comput. Surv.*, 37(3):195–237, 2005.

[WBB02] Andreas Weissel, Bjórn Beutel, and Frank Bellosa. Cooperative I/O: A novel I/O semantics for energy-aware applications. In *OSDI*, 2002.

[WdOF07] Lucas Francisco Wanner, Augusto Born de Oliveira, and Antônio Augusto Fröhlich. Configurable Medium Access Control for Wireless Sensor Networks. In *Embedded System Design: Topics, Techniques and Trends*, IFIP int. Federation for Information Processing, pages 401–410. Springer, 2007.

[WHdOF06] Lucas Francisco Wanner, Arliones Stevert Hoeller Junior, Augusto Born de Oliveira, and Antônio Augusto Fröhlich. Operating System Support for Data Acquisition in Wireless Sensor Networks. In *11th IEEE ETFA*, pages 582–585, Prague, Czech Republic, September 2006.

[WJF07] Geovani Ricardo Wiedenhoft, Arliones Stevert Hoeller Junior, and Antônio Augusto Fröhlich. A Power Manager for Deeply Embedded Systems. In *12th IEEE International Conference on Emerging Technologies and Factory Automation*, pages 748–751, Patras, Greece, September 2007.

[WWDS94] Mark Weiser, Brent Welch, Alan J. Demers, and Scott Shenker. Scheduling for reduced CPU energy. In *OSDI*, pages 13–23, 1994.

[YN03] Wanghong Yuan and Klara Nahrstedt. Energy-efficient soft real-time CPU scheduling for mobile multimedia systems. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 149–163, New York, NY, USA, 2003. ACM Press.

[Yua04] Wanghong Yuan. *Grace-OS: An energy-efficient mobile multimedia operating system*. PhD thesis, University of Illinois at Urbana-Champaign, 2004.

[ZELV02] Heng Zeng, Carla S. Ellis, Alvin R. Lebeck, and Amin Vahdat. Ecosystem: managing energy as a first class operating system resource. In *ASPLOS-X: Proceedings of the 10th international conference on Architectural support for programming languages and operating systems*, pages 123–132, New York, NY, USA, 2002. ACM Press.