Gerenciamento do Consumo de Energia Dirigido pela Aplicação em Sistemas Profundamente Embarcados*

Arliones Stevert Hoeller Junior, Lucas Francisco Wanner
e Antônio Augusto Fröhlich
Laboratório de Integração Software/Hardware
Campus Universitário - UFSC
P.O.Box 476, 88040-900
Florianópolis, Brasil
{arliones,lucas,guto}@lisha.ufsc.br

Abstract

Deeply Embedded Systems are dedicated computational platforms. Usually, these platforms are simple and have its resources limited to those which are necessary to execute the specific applications for which it was designed. Very often non-functional requirements drive the design of such devices. Among these requirements, is energy consumption. It is very important to handle energy consumption of these devices in a non-restrictive and low-overhead way. Restrictions cannot avoid the use of the several low-power operating modes such devices often feature. However, the energy consumption management strategy cannot compromise large amounts of system resources (processing and memory). In this context, this paper proposes an API (Application Programming Interface) which allows applications for managing energy consumption of software and hardware components. This paper also presents an component message propagation mechanism. This mechanism allows applications to handle operating modes of subsystems and the whole system. A prototype was develop using a component-based operating system. It shows that energy consumption management was possible without the need for using costly techniques or strategies. A case study done over a sensing platform showed energy savings of almost 40% by only allowing applications to express when certain components are no long being used.

Keywords: Power management, embedded systems, mobile computing, embedded operating systems.

Resumo

Sistemas Profundamente Embarcados são plataformas computacionais dedicadas. Estas plataformas são normalmente simples, com recursos limitados aos necessários para executar as aplicações específicas para as quais foi projetado. Muitas vezes, requisitos não funcionais são fatores predominantes no processo de desenvolvimento destes dispositivos. Um destes requisitos normalmente é o consumo de energia. Assim sendo, é muito importante que se possa tratar o consumo de energia destes dispositivos de uma maneira não-restritiva e eficiente. Não se pode restringir o acesso da aplicação aos variados modos de baixo consumo que dispositivos embarcados oferecem. Contudo, é necessário que a estratégia de gerência de energia não comprometa grandes quantidades de recursos do sistema (processamento e memória). Neste escopo, este artigo propõe uma API (*Application Programming Interface*) que possibilita que aplicações gerenciem o consumo de energia de componentes de software e hardware. Além da API, uma estrutura de propagação de mensagens entre componentes do sistema é apresentada, o que permite à aplicação controlar os modos de operação não só de componentes individualmente, mas também de subsistemas e do sistema como um todo. Através de um protótipo desenvolvido utilizando um sistema operacional baseado em componentes, mostra-se que a gerência do consumo de energia em sistemas embarcados foi possível sem a necessidade do uso de técnicas ou estratégias custosas em termos de processamento ou memória. Um estudo de caso executado sobre uma plataforma de sensoriamento mostrou economias de até 40% apenas por permitir que aplicações expressem quando certos componentes não estão mais sendo usados.

Palavras chave: Gerenciamento do Consumo de Energia, Sistemas Embarcados, Computação Móvel, Sistemas Operacionais Embarcados.

*Este trabalho foi parcialmente apoiado pela FINEP (Financiadora de Estudos e Projetos), contrato no. 01.04.0903.00.

1 Introdução

Sistemas profundamente embarcados são plataformas computacionais utilizadas para monitorar e/ou controlar os ambientes nos quais estão inseridos. Estes ambientes podem ser máquinas, motores, dispositivos eletrônicos, ambientes físicos (e.g, módulos de sensoriamento em uma rede de sensores sem-fio monitorando um habitat), etc. Neste contexto, é muito importante que estes sistemas sejam *power-aware*, i.e., capazes de gerenciar seu consumo de energia, possibilitando a diminuição do consumo e o controle do aquecimento. Contudo, a maioria das metodologias, técnicas e padrões de software para gerenciamento de energia não se mostram viáveis em sistemas profundamente embarcados com recursos limitados. Isto ocorre porque aquelas estratégias foram concebidas focando sistemas de propósito geral, onde sobrecargas de processamento ou memória são geralmente insignificantes.

ACPI e APM são os padrões de gerência de energia mais utilizados pela indústria hoje. Embora muito usados em sistemas de propósito geral, eles impõem requisitos de recursos adicionais de hardware ou capacidade de processamento que podem impossibilitar seu uso em sistemas embarcados. Como componentes de sistemas embarcados geralmente apresentam vários modos de operação para baixo consumo de energia, o uso de tais padrões pode ainda se tornar muito restritivo. Em conjunto a estes padrões, outras técnicas foram desenvolvidas para tratar o consumo de energia de sistemas eletrônicos. A maioria deles são classificados como dinâmicos (*Dynamic Power Management* - DPM) [2]. Estas técnicas reúnem informação através da análise do comportamento do sistema, e usam esta informação para guiar as decisões acerca do gerenciamento de energia. Exemplos de técnicas DPM são técnicas de *Dynamic Voltage and Frequency Scaling* (DVFS), que, dinamicamente, alteram a fonte de tensão e/ou frequência do processador para diminuir o consumo [1, 9, 11].

Embora simples, dispositivos embarcados também permitem gerenciamento de energia provendo diferentes modos de operação e uma grande gama de características configuráveis do hardware. Por exemplo, um microcontrolador ATMega [4], da Atmel, oferece oito modos de operação diferentes e características configuráveis para quase todos seus componentes (ADC, USART, etc) que têm efeito direto no consumo de energia. Variações de tensão (operação em 3 V e 5 V) e na frequência do processador também são possíveis com o auxílio de um circuito externo. De fato, hardware para sistemas embarcados suportam gerenciamento de energia, mas os ambientes de software existentes (sistemas operacionais e bibliotecas para sistemas embarcados) não provêm suporte adequado para este fim.

A maioria dos sistemas operacionais para plataformas embarcadas são compostas por simples camadas de abstração de hardware (*Hardware Abstraction Layers* - HAL) e oferecem muito poucos (ou nenhum) componentes de alto-nível como escalonadores, sistemas de arquivo, pilhas de protocolos de comunicação, etc. Na maioria destes sistemas, espera-se que as próprias aplicações implementem operações de controle de consumo de energia acessando a HAL. O problema com este método é que ele compromete a portabilidade da aplicação e força o programador desta a adquirir informação detalhada sobre o hardware que está utilizando.

Neste artigo nós exploramos gerência de energia dirigida pela aplicação para permitir o controle do consumo de energia em sistemas profundamente embarcados sem implicar em sobrecarga excessiva. Nesta estratégia, a portabilidade do código da aplicação é garantida devido ao uso de *hardware mediators* [13] e da organização hierárquica pela qual os componentes de software e hardware foram organizados. Gerência de energia dirigida pela aplicação foi possível permitindo que as aplicações expressem quando certos componentes de software ou hardware não estão mais sendo utilizados. Isto é feito através de uma interface simples e uniforme presente em todos os componentes do sistema. Quando a aplicação desliga um componente, este componente permanece desligado até que seja novamente acessado. Quando isto acontece, o mecanismo de gerência de energia é responsável por ligar este componente e qualquer outro componente que seja necessário para garantir que o sistema opere corretamente. Este mecanismo foi implementado permitindo a propagação de mensagens através dos componentes do sistema. Um estudo de caso é apresentado para demonstrar o uso da técnica através de uma implementação real deste mecanismo utilizando um sistema operacional baseado em componentes desenvolvido especificamente para plataformas embarcadas, o EPOS [6].

Este artigo está organizado como segue. A seção 2 introduz a interface de gerência de energia para componentes de software e hardware. A seção 3 apresenta uma aplicação para exemplificar o uso desta interface. A seção 4 apresenta uma revisão de trabalhos correlatos. A seção 5 finaliza o artigo.

2 Interface de Gerenciamento do Consumo de Energia para Componentes de Software e Hardware

Políticas para gerenciamento do consumo de energia em sistemas operacionais como LINUX e WINDOWS analizam dinamicamente o comportamento do sistema para determinar quando um componente de hardware deve modificar seu modo de operação através de uma interface compatível com ACPI. Contudo, a maioria dos sistemas embarcados

não podem arcar com os custos destas estratégias dinâmicas. Além disso, considerando que um sistema profundamente embarcado é normalmente composto por uma única aplicação, o melhor lugar para determinar a estratégia de gerenciamento do consumo de energia é na própria aplicação.

Através da definição de uma interface uniforme para o gerenciamento do consumo de energia de componentes do sistema nós permitimos que ao programador da aplicação mudar o estado de operação de cada componente do sistema, tanto de software quanto de hardware. A interface é composta por dois métodos: um para verificar o estado de operação atual do componente (`power()`) e outro para modificá-lo (`power(estado_desejado)`). O mecanismo por trás desta interface faz uso da organização hierárquica de componentes de software e hardware para permitir que esta migração de estados seja feita de forma consistente.

Componentes de hardware de baixa potência que são utilizados em sistemas embarcados frequentemente apresentam um grande conjunto de modos de operação. Permitir o uso de todos os modos de operação disponíveis, embora aumente a configurabilidade do sistema, pode aumentar a complexidade das aplicações quando gerenciando o consumo de energia do sistema. Para resolver este problema, foi estabelecido um conjunto de definições de alto-nível para os estados de cada componente, tornando desnecessário que o programador da aplicação conheça detalhes acerca de cada um dos componentes de hardware que seu sistema possui. Como no ACPI [5], quatro modos universais foram definidos: `FULL`, `LIGHT`, `STANDBY` and `OFF`. Contudo, estes modos podem ser extendidos pelos componentes sempre que necessário. Quando o dispositivo está operando com toda sua capacidade, ele está no estado `FULL`. O estado `LIGHT` coloca o dispositivo em modos de operação onde ele continua oferecendo as mesmas funcionalidades, porém consumindo menos energia e, muito provavelmente, implicando em alguma degradação de sua performance. No estado `STANDBY` o dispositivo para de operar, entrando em um estado do qual possa voltar quando solicitado e continuar sua operação do ponto em que parou. Este estado provavelmente será um modo de *sleep*. Quando no estado `OFF` o dispositivo é desligado. É importante notar que sempre que um dispositivo é desligado o retorno dele a um estado operacional implica em uma reinicialização, podendo assim perder dados ou configurações previamente definidas.

Conforme as aplicações embarcadas crescem em complexidade, estas passam a fazer uso de um maior número de componentes de sistemas. Com isso, pode tornar-se impraticável para programadores de aplicação controlar o consumo de energia de cada componente individualmente. Para solucionar este problema foi permitido que as aplicações alterem também o estado de operação de subsistemas (e.g., subsistema de comunicação, processamento, sensoriamento, etc) ou do sistema como um todo.

Para exemplificar como um subsistema inteiro pode alterar seu modo de operação, é apresentado uma breve descrição do ambiente experimental, o sistema operacional EPOS [6], e seu subsistema de comunicação. O *Embedded Parallel Operating System* (EPOS) é um sistema operacional orientado a aplicação baseado em componentes. No EPOS, abstrações de sistema de alto nível, como `File`, `Thread`, `Scheduler` e `Communicator`, são exportados para aplicações através de uma interface de componente. Estes componentes interagem com os dispositivos eletrônicos que utilizam através de mediadores de hardware [13]. Devido à hierarquia pela qual os componentes são organizados no sistema, cada abstração ou mediador sabe o estado de seus recursos. O subsistema de comunicação é apresentado na Figura 1 e compreende, basicamente, quatro famílias de componentes: `Communicator`, `Channel`, `Network` e `NIC`. `NIC` é a família de mediadores de hardware que abstrai o dispositivo de comunicação para a família `Network`. A família `Network` é responsável por abstrair o tipo de rede (e.g., Ethernet, CAN, ATM, etc). `Channel` é responsável por realizar a comunicação entre processos (IPC) e usa a `Network` para construir um canal lógico de comunicação através do qual mensagens são trocadas. Finalmente, o `Communicator` é um ponto-final para comunicações, e inclui uma estrutura configurável para adaptação de protocolos.

Para garantir portabilidade do código da aplicação, é esperado que o programador desta use abstrações de mais alto nível do sistema, como os membros da família `Communicator` no subsistema de comunicação. Neste contexto, a estratégia de gerenciamento do consumo de energia precisa prover meios pelos quais o programador da aplicação possa, apenas alterando o estado de um `Communicator`, modificar o modo de operação de todos os componentes envolvidos na real implementação das funcionalidades desenvolvidas por aquele componente. Isto é feito propagando mensagens através da estrutura hierárquica de componentes. Por exemplo, uma implementação de um `Communicator` vai usar um `Channel` e, provavelmente, um componente de temporização chamado `Alarm` para tratar *time-outs* no protocolo de comunicação. Quando a aplicação executa um comando solicitando que o `Communicator` altere seu modo de operação para `STANDBY`, o `Communicator` finalizará todas as comunicações iniciadas, esvaziando seus *buffers* e aguardando por todas as confirmações de recebimento (sinais `ACK`) antes de propagar mensagens de `STANDBY` para os outros componentes que usa.

Ações de gerência do consumo de energia do sistema como um todo no EPOS são tradadas pelo componente `System`. `System` contém referências para todos os subsistemas em uso pela aplicação. Então, se uma aplicação

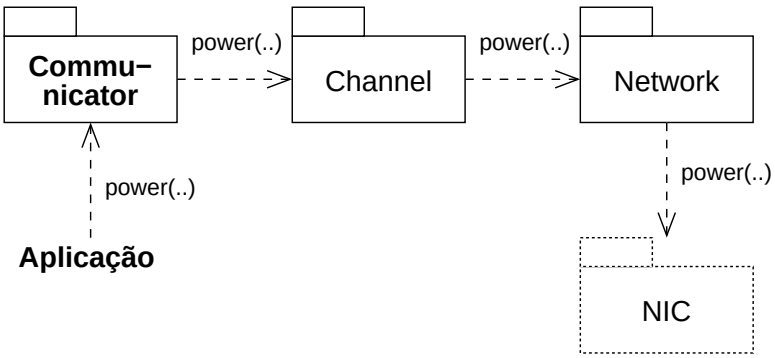

Figura 1: Subsistema de comunicação do EPOS.

deseja alterar o modo de operação do sistema inteiro, isto pode ser feito acessando a interface deste componente, que propagará este pedido para todos os subsistemas.

A Figura 2 ilustra como a interface de gerenciamento do consumo de energia pode ser acessada. Ela mostra alguns dos components instanciados para um sistema de sensoriamento. Nesta instância, o sistema é composto por três subsistemas: processamento (`CPU`), comunicação (`Communicator`), sensoriamento (`Sensor`). A figura ainda mostra o componente `System`. Cada componente tem sua própria interface, que pode ser chamada pela aplicação a qualquer momento, e um conjunto de níveis de consumo de energia específico. Se a aplicação deseja modificar o modo de operação de um subsistema específico, esta deve acessar diretamente seus componentes de mais alto nível hierárquico. Se a aplicação deseja modificar o modo de operação de todo o sistema, ela deve acessar o componente `System`, que propagará as mensagens através do sistema.

O principal desafio identificado no desenvolvimento de componentes *power-aware* foi a necessidade da migração consistente entre modos de operação de subsistemas inteiros em um ambiente onde as funcionalidades do subsistema estão dividos em vários componentes de software. Isto foi resolvido através do mecanismo de propagação de mensagens implementado, que garante que nenhum dado será perdido e nenhuma ação inacabada será interrompida. Permitindo que cada componente trate de suas responsabilidades (e.g., um `Communicador` esvaziando seus *buffers* e aguardando pelos sinais `ACK`) antes de propagar as mensagens de alteração do modo de operação para os componentes abaixo em sua hierarquia (e.g., para `Alarm` e `Channel`), é possível garantir troca de modo de operação consistente de um subsistema inteiro.

Nesta estratégia, é esperado que o programador da aplicação especifique, em seu código-fonte, quando certos componentes não estão sendo utilizados. Isto é feito executando os métodos "`power`" para cada componente, subsistema ou para o sistema. Para evitar que o programador tenha que, manualmente, acordar cada um destes componentes, o mecanismo de gerência garante que estes componentes retomem o seu estado anterior automaticamente quando acessados.

3 Estudo de Caso: Termômetro

Para demonstrar a usabilidade da interface definida, um termômetro foi implementado utilizando um protótipo com um termistor (resistor sensível a temperatura) de 10 kilo ohm conectado a um canal do conversor analógico-digital de um microcontrolador ATMega16, da Atmel [4]. A aplicação implementada para esta plataforma é apresentada na Figura 3. Esta aplicação utiliza quatro componentes do sistema: `System`, `Alarm`, `Thermometer` (membro da família de Senscientes [16]) e `UART`. A organização hierárquica do EPOS amarra, por exemplo, a abstração de sistema `Thermometer` ao conversor analógico-digital (`ADC`) do microcontrolador.

Quando a aplicação inicia, todos os componentes sendo utilizados são inicializados através de seus construtores, e um evento periódico é registrado no componente `Alarm`. O modo de operação de todo o sistema é então alterado para `STANDBY` através do método `power` do componente `System`. Quando isto acontece, o componente `System` coloca todos os componentes do sistema em `STANDBY`, com excessão do `Alarm`, para garantir que o sistema não perca sincronismo. O `Alarm` utiliza um temporizador (*timer*) para gerar interrupções a uma dada frequência. A cada interrupção de tempo, a CPU acorda e o componente `Alarm` trata todos os eventos registrados, executando os que atingiram seu período. Neste exemplo, a cada segundo os componentes `Thermometer` e `UART` são acordados automaticamente quando acessados e uma leitura de temperatura é enviada através da porta serial. Quando todos

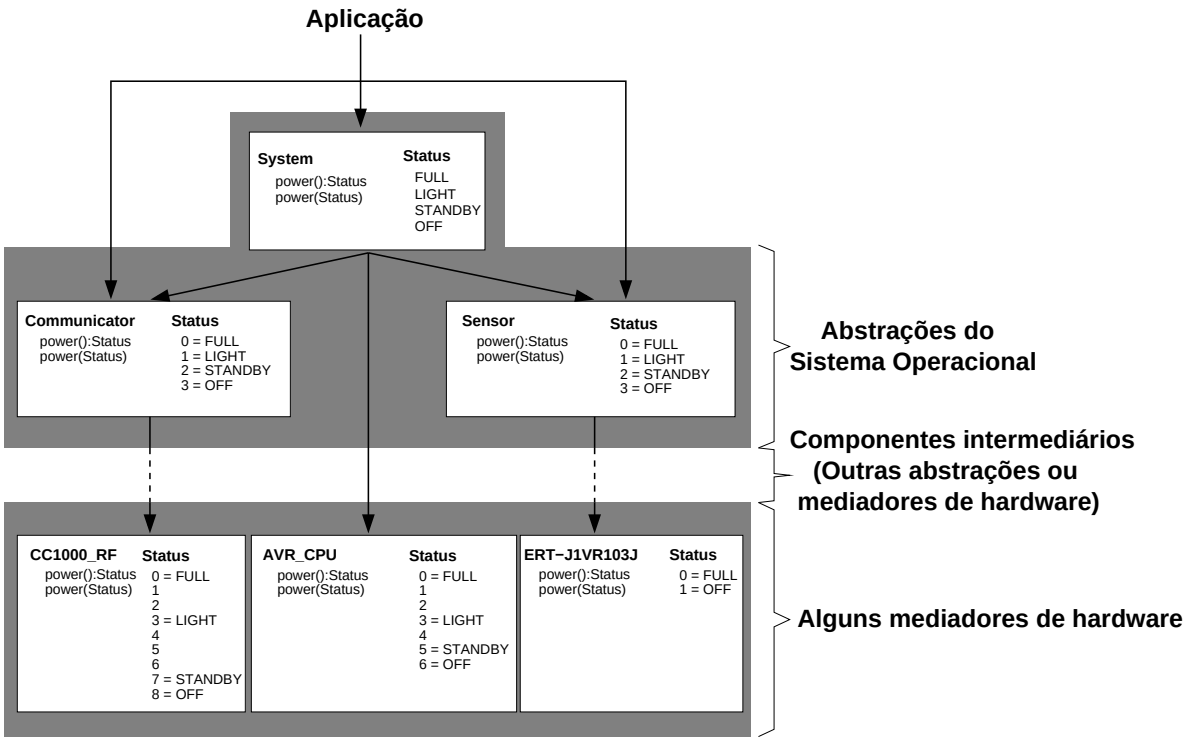

Figura 2: Acessando a interface de gerenciamento do consumo de energia.

os eventos registrados são tratados, a aplicação continua a execução normal chegando ao laço principal, que põe o componente `System` de volta em `STANDBY`.

Os gráficos apresentados na Figura 4 mostram medições do consumo de energia para esta aplicação sem e com gerência deste consumo pelo sistema. Ambos gráficos mostram o resultado de uma média de dez medições. Cada medição foi tomada por dez segundos. No gráfico (a) (sem gerência) é observado que o consumo do sistema oscila entre 2.5 e 4 Watts. No gráfico (b) (com gerência), a oscilação permanece entre 2 e 2.7 Watts. Calculando a integral destes gráficos, é possivel obter o consumo destas instâncias de sistema durante o tempo em que rodaram. Os resultados são 3.96 Joules para (a) e 2.45 Joules para (b), i.e., o sistema economizou 38.1% de energia sem comprometer a funcionalidade do sistema [1].

4 Trabalhos Correlatos

TINYOS e MANTIS são sistemas operacionais para plataformas embarcadas, mais especificamente, para plataformas de redes de sensores sem fio. Nestes sistemas, controle do consumo de energia foca principalmente na implementação de MACs de baixo consumo de energia [12, 15] e escalonadores para roteamento multi-salto [7, 14]. Isto faz sentido no contexto de redes de sensores sem fio porque uma parte muito significante do consumo de energia do sistema é devido ao mecanismo de comunicação. Embora estes sistemas mostrem resultados expressivos, eles constantemente focam no desenvolvimento de componentes que consumam o mínimo de energia possível, ao invés de permitir trocas entre consumo e performance, permitindo atender necessidades específicas de algumas aplicações. Outro incoveniente destes sistemas é o baixo nível de configuração oferecida pelos componentes de software e a falta de uma interface padronizada para acesso aos componentes do sistema.

SPEU (*System Properties Estimation with UML*) é uma ferramenta de otimização que leva em conta performance, tamanho de código do sistema e especifições de consumo de energia para gerar estimativas de sistemas otimizados para performance, tamanho de código ou energia. Estas informações são extraídas do modelo UML da aplicação embarcada. Este modelo precisa incluir diagramas de classe e de sequência. O protótipo usa um ambiente `Java` baseado no processador FEMTOJAVA [8]. Já que o SPEU apenas considera os diagramas UML, estas estimativas

[1] As leituras apresentam um valor relativamente alto porque o protótipo utilizado foi montado sobre um kit de desenvolvimento do fabricante do microcontrolador. Este kit possui vários dispositivos anexados ao circuito que geram uma base de ruído nas leituras

```
#include <system.h>
#include <sentient.h>
#include <uart.h>
#include <timepiece.h>

using namespace System;

System sys;
Thermometer therm;
UART uart;

void alarm_handler() {
    uart.put(therm.get());
}

int main() {
    Handler_Function handler(&alarm_handler);
    Alarm alarm(1000000, &handler);

    while(1) {
        sys.power(STANDBY);
    }
}
```

Figura 3: A aplicação Thermometer

mostram erros de até 85%, tornando-a útil apenas para comparar diferentes decisões de projeto. O ambiente ainda é pouco configurável, já que o processo de otimização é guiado por apenas uma variável, i.e., se a decisão do projetista da aplicação for por um sistema que ofereça o máximo de performance, o sistema nunca entrará em modos de baixo consumo de energia, mesmo que não esteja utilizando certos dispositivos. Isto certamente limita seu uso em situações reais.

CIAO (*CiAO is Aspect-Oriented*) é um projeto que foca o desenvolvimento de um sistema operacional de baixa granularidade para linhas de produção de sistemas embarcados. Este sistema abstrai propriedades não funcionais em aspectos, aumentando a configurabilidade do sistema. Uma destas propriedades não funcionais é consumo de energia. CIAO continua em um estágio inicial de desenvolvimento, mas se mostra relativamente diferente do método utilizado pelo EPOS. O EPOS também utiliza aspectos para abstrair algumas propriedades não funcionais mas, enquanto o CIAO utiliza orientação a aspectos como metodologia de projeto para o todo o sistema, o EPOS usa aspectos como uma ferramenta extra de desenvolvimento de software. No caso particular do controle do consumo de energia, não é desejável modelar isto como um aspecto porque, embora seja uma propriedade não funcional, muitas vezes decisões tomadas para diminuir o consumo de energia afetam o funcionamento do dispositivo, modificando o comportamento do sistema [10].

O IMPACCT (que significa *Integrated Management of Power-Aware Computing and Communication Technologies*) [3] é uma ferramenta de pré-escalonamento para explorar trocas entre consumo de energia e performance através de escalonamento e configuração arquitetural. A idéia por trás do IMPACCT é analisar a aplicação através de simulação para definir a maior gama de trocas entre consumo de energia e performance para cada componente durante a execução. Esta ferramenta inclui um escalonador *power-aware* para sistemas *hard real-time*. As ferramentas do IMAPCCT implementam um modo muito interessante de configurar escalonamento *hard real-time* focando consumo de energia em sistemas embarcados, porém está longe de provêr um ambiente onde a prototipação seja fácil e rápida.

5 Conclusão

Neste artigo foi apresentada uma estratégia para permitir gerenciamento de energia dirigido pela aplicação em sistemas profundamente embarcados. Para atingir este objetivo foi permitido que programadores de aplicação expressem os pontos em que certos componentes não estão sendo utilizados. Isto é feito através de uma interface de software simples e uniforme que permite a troca de modos de operação de componentes individualmente, agrupados em subsistemas ou do sistema como um todo, tornando todas as combinações de modos de operação possíveis. Explorando a estrutura hierárquica pela qual componentes de software e hardware são organizados em nosso sistema, gerenciamento eficiente

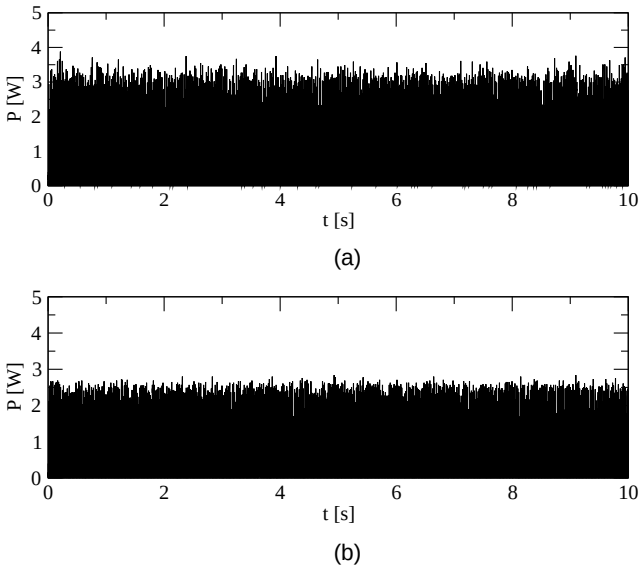

Figura 4: Consumo de energia para a aplicação Thermometer *sem (a)* e *com (b)* gerenciamento do consumo de energia.

de energia foi possível em sistemas profundamente embarcados sem lançar mão de técnicas ou estratégias custosas, gerando sistemas sem sobrecargas de processamento e memória desnecessárias.

Um estudo de caso utilizando um microcontrolador de 8 bits para monitorar temperatura em um ambiente fechado mostrou que quase 40% de energia foi economizada utilizando esta estratégia, o que implicou em apenas uma linha de código alterada na aplicação.

Referências

[1] Frank Bellosa, Andreas Weissel, Martin Waitz, and Simon Kellner. Event-driven energy accounting for dynamic thermal management. In *Proceedings of the Workshop on Compilers and Operating Systems for Low Power*, pages 04–1 – 04–10, New Orleans, USA, Sep 2003.

[2] 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.

[3] Pai H. Chou, Jinfeng Liu, Dexin Li, and Nader Bagherzadeh. Impacct: Methodology and tools for power-aware embedded systems. *DESIGN AUTOMATION FOR EMBEDDED SYSTEMS, Special Issue on Design Methodologies and Tools for Real-Time Embedded Systems*, 7(3):205–232, Oct 2002.

[4] Atmel Corporation. *ATMega16L Datasheet*. San Jose, CA, 2466j edition, Oct 2004.

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

[6] Antônio Augusto Fröhlich. *Application-Oriented Operating Systems*. Number 17 in GMD Research Series. GMD - Forschungszentrum Informationstechnik, Sankt Augustin, August 2001.

[7] Barbara Hohlt, Lance Doherty, and Eric Brewer. Flexible power scheduling for sensor networks. In *Proceedings of The Third International Symposium on Information Processing in Sensor Networks*, pages 205–214, Berkley, USA, Apr 2004. IEEE.

[8] S.A. Ito, L. Carro, and R.P. Jacobi. Making java work for microcontroller applications. *IEEE Design and Test of Computers*, 18(5):100–110, Sep-Oct 2001.

[9] Russ Joseph, David Brooks, and M. Martonosi. Live, runtime power measurements as a foundation for evaluating power/performance tradeoffs. In *Workshop on Complexity Effectice Design WCED, held in conjunction with ISCA-28*, June 2001.

[10] Daniel Lohmann, Wolfgang Schröder-Preikschat, and Olaf Spinczyk. Functional and non-functional properties in a family of embedded operating systems. In *Proceedings of the Tenth IEEE International Workshop on Object-oriented Real-time Dependable Systems*, Sedona, USA, Feb 2005. IEEEPress.

[11] 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.

[12] Joseph Polastre, Robert Szewczyk, Cory Sharp, and David Culler. The mote revolution: Low power wireless sensor network devices. In *Proceedings of Hot Chips 16: A Symposium on High Performance Chips*, aug 2004.

[13] Fauze Valerio Polpeta and Antônio Augusto Fröhlich. Portability in component-based embedded systems. In *Proceedings of the International Conference on Embedded and Ubiquitous Computing*, volume 3207 of *Lecture Notes in Computer Science*, pages 271–280. Springer, Aizu, Japan, sep 2004.

[14] Anmol Sheth and Richard Han. Adaptive power control and selective radio activation for low-power infrastructure-mode 802.11 lans. In *Proceedings of the 23rd International Conference on Distributed Computing Systems Workshops*, pages 797–802, Providence, USA, May 2003. IEEE.

[15] Anmol Sheth and Richard Han. Shush: A mac protocol for transmit power controlled wireless networks. Technical Report CU-CS-986-04, Department of Computer Science, University of Colorado, Boulder, dec 2004.

[16] Lucas Francisco Wanner, Arliones Stevert Hoeller Junior, Fauze Valerio Polpeta, and Antonio Augusto Frohlich. Operating system support for handling heterogeneity in wireless sensor networks. In *Proceedings of the 10th IEEE International Conference on Emerging Technologies and Factory Automation*, Catania, Italy, Sep 2005. IEEE.