# Energia como um Parâmetro para Qualidade de Serviço em Sistemas Embarcados

**Geovani R. Wiedenhoft, Arliones Hoeller Jr. and Antônio A. Fröhlich**
Universidade Federal de Santa Catarina
Laboratório de Integração Software e Hardware
PO Box 476 – 88049-900 – Florianópolis, SC, Brasil
{grw,arliones,guto}@lisha.ufsc.br

**Abstract**

Besides the usual limitations in terms of memory and processing capabilities, embedded systems are often powered by batteries, making energy consumption another important restriction. This work explores energy as a parameter for Quality of Service (QoS) of embedded systems. We argue that it is not enough to guarantee other QoS metrics (e.g., processing, communication, memory) if doing so the system runs out of battery and is unable to complete its computations. In this context, we developed a scheduler for imprecise tasks that, knowing battery's life-time and tasks' energy consumption, prevents tasks from executing its optional parts to ensure that tasks finish. A prototype was developed for EPOS (Embedded Parallel Operating System) using the system's power management infrastructure.

**Keywords:** Energy Consumption, Embedded Systems, Quality of Service, Imprecise Computation.

**Resumo**

Além das usuais limitações em termos das capacidade de processamento e memória, os sistemas embarcados são freqüentemente alimentados por baterias, tornando o consumo de energia outra importante restrição. Este trabalho explora a energia como parâmetro para qualidade de serviço (QoS) de sistemas embarcados. Nós argumentamos que não é suficiente garantir outras métricas de QoS, como processamento, comunicação e memória, se a bateria termina e o sistema não é capaz de terminar seus cálculos. Neste contexto, nós desenvolvemos um escalonador para tarefas imprecisas que, conhecendo o tempo de vida da bateria e o consumo de energia das tarefas, impede das tarefas executarem suas partes opcionais para assegurar que tarefas terminem. Um protótipo foi desenvolvido para o EPOS (*Embedded Parallel Operating System*) usando a infra-estrutura de gerência de energia.

**Palavras chave:** Consumo de Energia, Sistemas Embarcados, Qualidade de Serviço, Computação Imprecisa.

## 1 Introdução

Sistemas embarcados são plataformas computacionais dedicadas a realizar um determinado conjunto de tarefas, como monitorar e/ou controlar os ambientes nos quais estão inseridos. Muitos desses sistemas apresentam rigorosas limitações em termos das capacidades de processamento e memória. Alguns deles, devido a natureza móvel de suas aplicações, são também alimentados por baterias, e isso faz com que a energia seja outro importante recurso no qual deve ser gerenciado. Considerando todas essas limitações, torna-se importante que os sistemas embarcados móveis sejam capazes de gerenciar o consumo de energia sem comprometer a execução do sistema.

O *Hardware* dos sistemas embarcados provê um vasto conjunto de mecanismos para gerenciar o consumo de energia. Esses mecanismos incluem técnicas como DVS (Dynamic Voltage Scaling), hibernação de recursos, conjuntos de distintos modos de operação, etc. Alguns projetos desenvolvidos pela academia exploram a integração dessas técnicas em trabalhos que garantem qualidade de serviço (QoS) em aplicações enquanto minimizando o consumo de energia. Esses trabalhos, entretanto, focam suas gerências do consumo de energia nas métricas tradicionais de QoS, como para o processamento, memória e comunicação. Neste trabalho, nós argumentamos que não é suficiente garantir outras métricas de QoS se, fazendo isso, a bateria termina antes das tarefas completarem suas execuções.

Neste trabalho nós propomos usar a energia como parâmetro de QoS, no qual garantir a métrica de QoS em energia é considerada mais importante do que garantir outras métricas. A abordagem proposta espera o desenvolvedor definir o período em que o sistema embarcado deve estar operacional. Pelo monitoramento do tempo de vida da bateria, o escalonador é capaz de diminuir métricas de QoS em termos de processamento, memória ou comunicação para reduzir o consumo de energia e, assim, aumentar o tempo de vida do sistema.

Em nosso protótipo, o controle de QoS das aplicações foi inspirada na computação imprecisa [4–6]. As tarefas da computação imprecisa são divididas em duas partes, uma implementado o fluxo de execução obrigatório, e outra implementando um fluxo opcional. A parte obrigatória é a principal da tarefa, e deve sempre ser executada. A parte opcional, entretanto, executa ações não críticas, e pode ser impedida de executar caso o sistema esteja com falta de recursos.

As técnicas da computação imprecisa foram adaptadas para alterar o foco do escalonamento de *deadlines* das tarefas para o controle do consumo de energia. Esta versão adaptada do ambiente da computação imprecisa foi implementada no EPOS [7], um sistema operacional embarcado baseado em componentes. EPOS provê uma infraestrutura que permite aplicações interagirem com o sistema para implementar uma apropriada gerência do consumo de energia para sistemas embarcados [3]. O escalonador impede a execução das partes opcionais quando o nível de energia está abaixo de um limite pré-definido, e assim, reduzindo a demanda por processamento do sistema. Essa redução cria mais períodos ociosos no sistema, nos quais o escalonador pode usar a infraestrutura mencionada acima para trocar os modos de operação, parando ou diminuindo o consumo de energia pelos componentes do sistema durante os períodos ociosos.

Este artigo é estruturado como segue. Seção 2 apresenta os trabalhos relacionados. Seção 3 descreve o escalonador que foi desenvolvido neste trabalho. Seção 4 abrange a implementação do ambiente da computação imprecisa proposta para energia em sistemas embarcados. Seção 5 finaliza resumindo o artigo e apresentando as futuras direções para este trabalho.

## 2 Trabalhos Relacionados

GRACE-OS foi apresentado pelo Yuan [9] como um sistema operacional eficiente em termos de energia para aplicações móveis de multimídia. Esse sistema usa técnicas de adaptações multi-camadas para garantir QoS em sistema com *software* e *hardware* adaptativos. GRACE-OS utiliza escalonamento de tempo real e DVS para dinamicamente gerenciar o consumo de energia. Ele foi implementado sobre o sistema operacional LINUX. GRUB-PA [8] é de certa forma semelhante ao GRACE-OS. A principal diferença é o tipo de tarefas que cada sistema utiliza. Enquanto que GRACE-OS provê suporte somente para tarefas *soft real-time*, GRUB-PA também suporta tarefas *hard real-time*.

Outros projetos exploram um balanceamento entre QoS das aplicações e consumo de energia através de adaptações nas aplicações. Um sistema operacional que utiliza esse tipo de gerência é ODYSSEY [1]. ODYSSEY implementa um *framework* para aplicações *web* e multimídia. O sistema monitora os usos de recursos pelas aplicações e alerta as aplicações quando os recursos começam a ficar escassos, esperando que as aplicações fiquem em um nível baixo de QoS até que os recursos estejam disponíveis novamente.

Outro sistema operacional que suporta aplicações adaptativas é o ECOSystem [10]. Esse sistema determina custos que as aplicações devem pagar para utilizar recursos do sistema (como por exemplo, acesso a memória, rede, disco). As aplicações pagam para acessar recursos através de uma moeda, chamada *currentcy*. O sistema distribui *currentcies* periodicamente para as tarefas de acordo com um equação que define uma velocidade de descarga que a bateria pode assumir para forçar o sistema a durar um período de tempo definido. Isso com que as aplicações adaptem as execuções baseadas nos seus *currentcy*.

## 3 QoS em Termos de Energia

Este trabalho desenvolveu uma abordagem para garantir que o tempo de vida da bateria de sistemas embarcados possa durar o tempo suficiente para assegurar a execução de, no mínimo, as tarefas especificadas previamente pelo usuário como essenciais. Quando o sistema detecta que a bateria não durará o suficiente para satisfazer o tempo de vida esperado, ele começa a diminuição controlada dos níveis de QoS com objetivos de economizar energia. O controle de QoS proposto utiliza técnicas extraídas dos mecanismos da computação imprecisa [6].

A computação imprecisa é uma técnica de escalonamento, originalmente proposta para satisfazer os requisitos temporais das tarefas de tempo real através da diminuição controlada dos níveis de QoS. Tarefas imprecisas são divididas em duas partes: uma parte obrigatória e outra opcional. Tarefas imprecisas perdem na qualidade do resultado não executando as partes opcionais com objetivos de garantir que nenhum *deadline* de execução seja perdido. Com essa divisão a computação imprecisa une a computação de tempo real e as técnicas de melhor esforço para, respectivamente, a parte obrigatória e a parte opcional.

A parte obrigatória das tarefas imprecisas geram resultados imprecisos que refletem o mínimo de QoS para garantir que esses resultados sejam úteis. Esses resultados imprecisos tem suas qualidades elevadas quando as partes opcionais são executadas, gerando os resultados precisos. A execução da parte obrigatória é garantido sem levar em consideração o nível de energia do sistema. A execução da parte opcional, entretanto, não é garantido. Nesta proposta, as partes opcionais são executadas somente se o tempo de vida da bateria desejado é sustentado.

Existem diversas possibilidade de aplicações da computação imprecisa. Por exemplo, é possível utilizar as tarefas imprecisas para implementar o processamento de imagens. Neste exemplo, as partes obrigatórias devem gerar uma imagem com uma qualidade aceitável, enquanto as partes opcionais devem aumentar a qualidade dessa imagem. Outro conjunto de aplicações pode ser os algoritmos a qualquer tempo. Esses algoritmos normalmente implementam métodos iterativos que refinam os resultados depois de cada iteração. Nesse caso, quanto mais tempo o algoritmo é executado, melhor é a qualidade do resultado. Nesse conjunto de algoritmos incluem os métodos numéricos, cálculos de raízes de polinômios, aproximações numéricas, e entre outros. Outro conjunto de aplicações são as aplicações de controle&conforto.

Para utilizar a computação imprecisa para prover QoS em termos de energia, algumas adaptações foram realizadas. Essas modificações são descritas na seção 3.2. A seção a seguir descreve como as estimativas do consumo de energia podem ser realizadas para tornar essa abordagem possível.

### 3.1 Estimativas do Consumo de Energia

Com objetivos de prover QoS em termos de energia é necessário verificar se o tempo de vida da bateria requerido pela aplicação pode ser sustentado. Para fazer isso, o tempo de vida desejado é comparado com a estimativa do tempo de vida da bateria. A estimativa do tempo de vida da bateria é obtida através do cálculo do nível de energia da bateria e estimativas do consumo de energia necessárias para as tarefas futuras. As plataformas dos sistemas embarcados, normalmente, provê mecanismos para acessar o nível da bateria através de um circuito analógico que pode ser conectado a um conversor analógico-digital ($ADC$). Para estimar o consumo de energia futuro dos conjuntos de tarefas é, entretanto, mais complexo. Este trabalho explora dois caminhos para realizar essa estimativa:

- Através da análise da velocidade de descarga. Um monitor pode ler o nível da bateria periodicamente e estimar o tempo de vida da bateria considerando que a velocidade de descarga é constante. Uma vez que nenhuma velocidade de descarga nem a carga de trabalho das tarefas são constantes, essa aproximação pode resultar em erros na estimativa. Embora esse método apresente uma baixa precisão, ele é uma solução com um baixo sobrecusto para o problema.

- Através do monitoramento do comportamento do sistema. Um modelo de custo pode ser organizado para representar os custos em termos de energia que cada atividade do sistema possui (como por exemplo, quanta energia é consumida para realizar uma conversão com o ADC ou enviar um *byte* através do rádio). O consumo de energia das tarefas pode, então, ser obtida com uma análise antes da execução, fazendo uma estimativa da energia consumida no futuro com maior precisão. A implementação desse modelo pode, entretanto, tornar-se uma tarefa complexa, pela diversidade de variáveis que podem afetar na energia em um sistema eletrônico (como por exemplo, as variações nos valores da tensão e corrente).

### 3.2 Adaptações nas Técnicas da Computação Imprecisa

O principal objetivo nas adaptações realizadas nas técnicas da computação imprecisa é trocar o foco dos mecanismos de escalonamento com restrições temporais para restrições com relação a energia. Nesse contexto, uma adaptação realizada é alterar o escalonador para, ao invés de garantir *deadlines* individuais das tarefas, garantir o tempo de vida da bateria especificado pelo usuário. Nesta abordagem, os *deadlines* individuais das tarefas e os requisitos temporais deixam de constituir uma restrição no sistema, depois dessa adaptação.

A outra adaptação realizada é a alteração das estimativas de tempos de execuções das tarefas para as estimativas de consumo de energia. Essa estimativa é usada pelo escalonador para verificar se o tempo de vida da bateria desejado será sustentado e, eventualmente, impedir que tarefas opcionais executem. Conhecendo essa estimativa, o escalonador pode, também, impedir a execução de partes opcionais específicas.

## 4 Implementação

Um protótipo foi desenvolvido para testar a abordagem proposta usando o EPOS (Embedded Parallel Operating System) [2]. EPOS é um *framework* com componentes hierarquicamente organizados para a geração de sistemas específicos a uma determinada aplicação embarcada. O EPOS analisa o conjunto das aplicações dedicadas que ele deve suportar para a geração do sistema, e então, configura o sistema de acordo. Nós estendemos o EPOS para suportar tarefas imprecisas e modificamos seu escalonador para tratar apropriadamente a execução condicional das partes opcionais de cada tarefa. Para o objetivo deste trabalho, o EPOS, também, disponibiliza a abstração ao acesso as informações da bateria, como o nível da bateria e estimativas do tempo de vida do sistema. Nesta versão atual, a estimativa do tempo de vida do sistema é realizada somente pela análise da velocidade de descarga do nível da bateria.

Nós estudamos três caminhos para a implementação das tarefas imprecisas no EPOS. Todos os caminhos estudados esperam que o programador especifique no momento da criação de uma *thread* dois ponteiros para funções: um para a parte obrigatória e outro para a parte opcional. A diferença está na maneira em que o sistema operacional trata essas partes. A primeira alternativa que estudamos é a criação de duas *threads*, um fluxo de execução que trata a parte obrigatória e outro fluxo que trata a parte opcional. A criação dessas *threads* são realizadas pelo sistema, sendo então, transparente para o programador da aplicação. As desvantagens identificadas nessa implementação são os sobrecustos pelo aumento do número de *threads* no sistema e o conjunto de problemas complexos com relação a coordenação dessas *threads* que são necessários para assegurar a correta seqüencia de execução para as partes obrigatórias e opcionais das tarefas.

Outra alternativa é ajustar a pilha da tarefa para ter ambos os ponteiros das partes (obrigatória e opcional) a serem executadas seqüencialmente no momento em que a tarefas executa. Para realizar esse ajuste, o endereço da função de retorno na pilha deve ser trocado pelo endereço com o código para a chamada da parte opcional ou somente finaliza a execução da tarefa. Essa abordagem apresenta mais desvantagens do que a anterior: a parte opcional "sempre" deve executar depois da parte obrigatória e somente será utilizável para tarefas periódicas. Tarefas não periódicas que implementam, por exemplo, um *loop* infinito não saem do fluxo de execução obrigatório nessa abordagem, impedindo as partes opcionais de serem executadas.

Finalmente, a implementação adotada neste trabalho disponibiliza ao programador da aplicação um caminho para especificar na implementação da parte obrigatória o ponto no qual a parte opcional deve ser executada. Nessa abordagem, o programador pode chamar a parte opcional antes, depois ou no meio da parte obrigatória. A figura 1 apresenta uma aplicação de controle&conforto para exemplificar o uso da infra-estrutura proposta para as tarefas imprecisas. O programador da aplicação separa as tarefas em duas partes: um parte obrigatória (`control`) e outra parte opcional (`comfort`). Uma tarefa imprecisa é criada pela criação de uma *thread* passando os dois ponteiros para funções: `control` e `comfort`. Na parte obrigatória, o programador da aplicação especifica o ponto no qual a parte opcional será executada

através da chamada ao método `optional`. O programador da aplicação necessita definir o tempo de vida do sistema que é esperado, passando um *timestamp* para o sistema através do método `system_lifetime`. Essa implementação, também, permite o programador da aplicação mudar o ponteiro da função opcional durante a execução da parte obrigatória, habilitando a tarefa alterar sua parte opcional se desejado. Isso é realizado pelo método `optional(FunctionPointer)`. A figura 2 apresenta um diagrama de classe em UML para a implementação das tarefas imprecisas no EPOS.

```
void control() {
    //...
    Imprecise_Thread::self()->optional();
    //...
}
```

```
void comfort() {
    //...
}
```

```
void main() {
    Imprecise_Thread::system_lifetime(TIMESTAMP);
    //...
    Imprecise_Thread thread(&control,&comfort);
    //...
}
```

Figura 1: Exemplo de utilização de tarefas imprecisas.

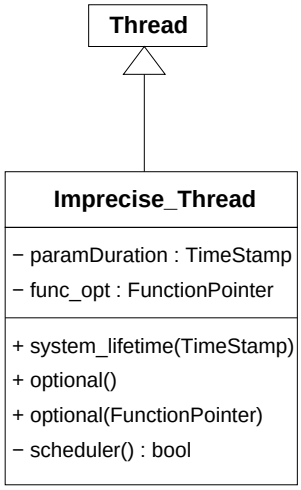


Figura 2: Diagrama de classe em UML para as tarefas imprecisas no EPOS.

A cada chamada do método `optional` é realizado um teste para verificar se o sistema é capaz de manter a carga de trabalho sem que a bateria termine antes do tempo de vida do sistema seja alcançado. No momento em que o escalonador identifica que o tempo de vida exigido não pode ser alcançado com a carga de trabalho corrente, certas partes opcionais são impedidas de executarem reduzindo o consumo de energia. Caso posteriormente o escalonador identifique que o tempo de vida do sistema pode ser alcançado novamente, o sistema permite a execução das partes opcionais. A figura 3 apresenta um diagrama de seqüencia em UML que ilustra como o sistema trata a execução das partes opcionais.

## 5 Conclusão

Este trabalho propôs uma abordagem que explora a energia como um parâmetro de QoS em sistemas embarcados móveis, para atender o tempo de vida da bateria definido pelo programador da aplicação. Essa

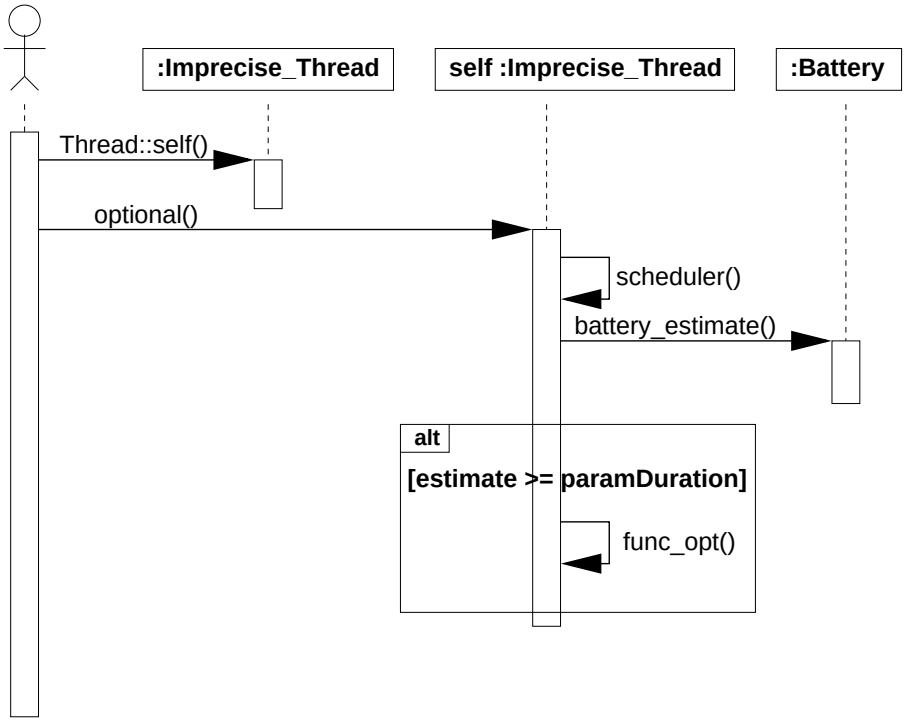


Figura 3: Diagrama de seqüencia em UML para a execução de partes opcionais.

abordagem foi inspirada pelos conceitos de tarefas imprecisas, que são tarefas que podem ser divididas em parte obrigatória e parte opcional. O escalonador proposto analisa a descarga do nível da bateria para verificar se o tempo de vida definido pelo programador pode ser atendido ou não. Caso o escalonador identifique que o tempo de vida esperado não pode ser alcançado, certas partes opcionais são impedidas de executarem, diminuindo os níveis da qualidade de serviço. Um protótipo foi implementado no EPOS que permitiu gerenciar a energia consumida nos períodos ociosos em que as partes opcionais foram impedidas de executarem, consumindo menos energia e aumentando o tempo de vida da bateria. A implementação é funcional e trabalhos futuros caminharão em direção a uma abordagem que atenda tanto os requisitos em termos de energia quanto os requisitos temporais das tarefas.

## Referências

[1] FLINN, J., AND SATYANARAYANAN, M. Energy-aware adaptation for mobile applications. In *SOSP '99: Proceedings of the seventeenth ACM symposium on Operating systems principles* (New York, NY, USA, 1999), ACM Press, pp. 48–63.

[2] FRÖHLICH, A. A. *Application-Oriented Operating Systems*. GMD - Forschungszentrum Informationstechnik, Sankt Augustin, 2001.

[3] HOELLER, A. S. J., WANNER, L. F., AND FRÖHLICH, A. A. A Hierarchical Approach For Power Management on Mobile Embedded Systems. In *5th IFIP Working Conference on Distributed and Parallel Embedded Systems* (Braga, Portugal, Oct. 2006), pp. 265–274.

[4] LIN, K. J., NATARAJAN, S., AND LIU, J. W. S. Imprecise results: Utilizing partial computations in real-time systems. In *Proc. IEEE Real-Time System Symp.* (1987).

[5] LIN, K. J., NATARAJAN, S., AND LIU, J. W. S. Scheduling real-time, periodic jobs using imprecise results. In *Proc. IEEE Real-Time System Symp.* (1987).

[6] LIU, J. W., SHIH, W.-K., LIN, K.-J., BETTATI, R., AND CHUNG, J.-Y. Imprecise computations. *Proceedings of the IEEE 82*, 1 (Jan 1994), 83–94.

[7] Marcondes, H., Junior, A. S. H., Wanner, L. F., and Fröhlich, A. A. Operating Systems Portability: 8 bits and beyond. In *11th IEEE International Conference on Emerging Technologies and Factory Automation* (Prague, Czech Republic, Sept. 2006), pp. 124–130.

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

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

[10] Zeng, H., Ellis, C. S., Lebeck, A. R., and Vahdat, A. 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* (New York, NY, USA, 2002), ACM Press, pp. 123–132.