Sistema Operacional YATOS para Redes de Sensores sem Fio

Vinícius C. de Almeida[1], Luiz F. M. Vieira[1], Breno A. D. Vitorino[1], Marcos A. M. Vieira[1], José A. Nacif[1] , Antônio O. Fernandes[1], Diógenes C. da Silva[2], Claudionor N. Coelho Jr[1]

[1]Departamento de Ciência da Computação – Universidade Federal de Minas Gerais
Av. Antônio Carlos, 6627 – 31270-010, Belo Horizonte/MG

[2]Departamento de Engenharia Elétrica – Universidade Federal de Minas Gerais
Av. Antônio Carlos, 6627 – 31270-010, Belo Horizonte/MG

{makish, vitorino, lfvieira, mmvieira, jnacif, otavio, coelho}@dcc.ufmg.br

diogenes@cpdee.ufmg.br

Abstract. *This paper describes YATOS , an operating system developed for sensor nodes of wireless sensor networks (WSN). A sensor node is a computing device with limited communication, sensing and processing capability. YATOS maps events and tasks, it is event-driven and it provides an application programming interface (API). We present the related works in the area that helped designing the new system, which is also described in this paper.*

Resumo. *Este artigo descreve o YATOS , um sistema operacional desenvolvido especificamente para nós sensores de redes de sensores sem fio (RSSF). Um nó sensor é um pequeno dispositivo computacional com capacidades limitadas de comunicação, sensoriamento e processamento. O YATOS atende aos requisitos impostos pelas RSSFs, mapeia eventos em tarefas, é dirigido por eventos e fornece uma interface de programação de aplicativos (API). Apresentamos os trabalhos relacionados na área que ajudaram a projetar o novo sistema, o qual também é descrito neste artigo.*

1. Introdução

Redes de sensores [10] são formadas por milhares de pequenos dispositivos, aqui denominados *nós*, dotados de capacidade de armazenamento, processamento, comunicação e sensoriamento. Esses nós têm fortes restrições quanto à memória, capacidade de processamento e principalmente energia, sendo desejável que possuam mecanismos de autoconfiguração e adaptação devido a problemas como falha de comunicação e perda de nós. Nessa rede, cada nó pode ser equipado com diferentes sensores, dada a natureza dos aplicativos executados nele, tais como: temperatura, pressão, etc.

Nós sensores podem ser usados para monitoração contínua, detecção de eventos, localização e controle local de atuadores. As áreas de aplicação das RSSFs são proeminentes e se destacam a área militar, meio-ambiente, saúde, automação residencial, monitoração de estruturas e aplicações comerciais.

RSSFs possuem recursos limitados, sendo energia o mais importante desses. Cada nó sensor possui uma fonte de energia, que em geral é uma bateria com capacidade limitada. É praticamente inviável recarregar manualmente todas as baterias, uma vez que RSSFs são compostas por milhares de nós sensores e, além disso, estes podem estar em locais inacessíveis. Dessa forma, o foco de projeto em RSSFs, do hardware aos protocolos de redes, é o uso eficiente de energia.

Devido às características dos nós e da própria rede de sensores, algumas questões devem ser consideradas no projeto de um sistema operacional:

- dadas suas limitações de memória, os nós não podem armazenar todos os aplicativos em sua memória local, sendo desejável que possuam um mecanismo para que novos aplicativos sejam transferidas de um nó para outro;
- deve-se fornecer também algum suporte à execução concorrente de aplicativos em um nó;
- a quantidade de nós em uma dada rede de sensores pode ser muito grande, tornando difícil a configuração manual de cada nó individual;

Devido às limitações quanto ao consumo de energia e capacidade computacional limitada dos nós, torna-se indispensável o uso eficiente de seus recursos. Dessa forma, um sistema operacional projetado especificamente para atender a esses requisitos pode oferecer maior tolerância a falhas, bem como economia de energia e facilidade de desenvolvimento de aplicações. Ele deve ser pequeno para caber na memória, consumir pouca energia, executar sem bloqueio e ser voltado para sistemas distribuídos. Este trabalho apresenta como contribuição o primeiro sistema operacional brasileiro voltado especificamente para RSSFs.

Este artigo está organizado da seguinte forma. A seção 2 descreve brevemente rede de sensores sem fio. A seção 3 apresenta os trabalhos relacionados, a seção 4 mostra as características do YATOS e a seção 5 os trabalhos futuros e conclusão.

2. Redes de Sensores Sem Fio

Redes de Sensores Sem Fio são redes *ad-hoc* formadas por nós sensores e por pelo menos um ponto de comunicação denominado estação-base. O objetivo destas redes é coletar informação. Usualmente, não possuem infra-estrutura pré-estabelecida, como ocorre com redes de celulares ou redes locais sem fio.

2.1. Aplicações

Redes de Sensores Sem Fio tem o potencial de serem aplicadas em várias áreas. RSSFs permitem monitorar uma grande variedade de condições ambientais que incluem: temperatura, umidade, movimentos veiculares, condições de iluminação, pressão, nível de ruído, presença ou ausência de certos tipos de objetos, nível de estresse mecânico em objetos atachados, características correntes como velocidade, direção e tamanho de um objeto.

Nós sensores podem ser usados para monitoração contínua, detecção de eventos, localização e controle local de atuadores. As áreas de aplicação das RSSFs são proeminentes e se destacam a área militar, meio-ambiente, saúde, automação residencial, monitoração de estruturas e aplicações comerciais.

Podemos citar algumas aplicações de meio-ambiente de redes de sensores. Elas incluem o rastreamento do movimento dos pássaros, pequenos animais, e insetos; monitoração de condições ambientais que afetam colheitas e plantio; irrigação; detecção de componentes químicos ou biológicos; agricultura de precisão; monitoração da água, solo e atmosfera; detecção de incêndio em florestas; pesquisa meteorológica ou geofísica; detecção de inundação; mapeamento da bio-complexidade ambiental e estudo da poluição.

3. Trabalhos relacionados

Para atender às necessidades de dispositivos com severas restrições quanto ao consumo de energia, espaço em memória e poder computacional, já foram propostos e/ou desenvolvidos diferentes SOs, que utilizam diferentes abordagens ao problema. Sistemas baseados em Linux, tal como μClinux, não se mostraram adequados para RSSFs, pois são voltados para sistemas embutidos com maior capacidade computacional (i.e. memória, bateria, processador). Os sistemas descritos a seguir atendem algumas das demandas específicas de RSSFs.

3.1. EYES-PEEROS

EYES [5] é um projeto desenvolvido na Universidade de Twente - Holanda. Seu funcionamento baseia-se nos seguintes princípios: operação dirigida por eventos, divisão da execução em tarefas e separação de funcionalidades correlatas em unidades distintas.

Uma primeira iniciativa de implementar o SO do projeto EYES é o sistema *PEEROS* (Preemptive Eyes Real Time Operating System) [9]. Seus elementos principais são descritos abaixo:

- Escalonamento de tarefas: sistema dirigido por eventos, com política de escalonamento preemptiva.
- Sistema de mensagens: comunicação entre os componentes do SO e entre os nós sensores.
- Gerente de módulos: armazena módulos executáveis, não usados, na memória externa (EEPROM) e os carrega no nó sensor sob demanda.
- "Shell" de comandos: conectado à interface serial do nó, permite a um usuário digitar comandos e receber respostas do nó sensor (quando este estiver ligado à porta serial de um computador pessoal).

3.2. SensorWare

Desenvolvido na Universidade da Califórnia, *SensorWare* [1] é um arcabouço para RSSFs. Suas características principais são: mobilidade de código através de scripts móveis e separação de funcionalidades correlatas em unidades distintas.

Scripts móveis são entidades do SensorWare que podem se deslocar de um nó para outro numa RSSF. Executam até um certo ponto num nó e, ao transferir-se para outro, continuam a execução a partir de pontos de entrada bem definidos. A maioria dos comandos e funções são agrupadas em APIs distintas. Elas possuem funcionalidades que fornecem acesso a um recurso ou serviço do nó sensor.

3.3. Bertha

Bertha OS [8] é um sistema operacional para a arquitetura de hardware *Pushpin* [2](redes de sensores com fio e com nós idênticos). Desenvolvido no MIT Media Lab, seu modelo de programação é baseado em um sistema de agentes móveis [7] muito simples, com três componentes: fragmentos de processos (PFrags), um sistema de quadro de boletins (BBS) e um vigia de vizinhança (NW).

Fragmentos de processos, ou *PFrags*, são entidades de programação autônomas e móveis, escritas em C, que podem interagir com seu ambiente local, podendo migrar de um nó para outro. Para isso, eles possuem código executável e estado persistente e podem se comunicar através do BBS do nó. Um sistema de quadro de boletins, ou *BBS*, está disponível para os PFrags em cada nó da rede. Ele é usado para permitir a comunicação entre PFrags, que postam mensagens nele. Sinopses do BBS dos nós vizinhos são espelhados no vigia de vizinhança, ou *NW*, do nó local. Quando um PFrag posta alguma mensagem

em um BBS, aquele pode escolher que uma sinopse daquela postagem esteja disponível no NW de todos os nós vizinhos. Infelizmente, PFrags não são flexivéis, limitando o tamanho de código dos programas.

3.4. TinyOS

Desenvolvido na UC Berkeley e ativamente utilizado por uma grande comunidade de usuários, *TinyOS* [6] é um escalonador de eventos que provê execução concorrente para redes de sensores embutidos, com recursos de hardware escassos usando a arquitetura Motes [3]. As características relevantes do TinyOS são sua arquitetura e seu modelo de concorrência, os quais são brevemente descritos a seguir.

3.4.1. Arquitetura baseada em componentes

Todo aplicativo possui pelo menos um arquivo de configuração e um de implementação. O primeiro especifica o conjunto de componentes do aplicativo e como eles se invocam. No segundo estão listadas as interfaces fornecidas e as usadas por um componente.

Um aplicativo usa um ou mais componentes, sendo possível reutilizar alguns componentes mais simples para criar outros mais elaborados. Isso cria uma hierarquia de camadas, na qual componentes em níveis altos na hierarquia originam comandos para componentes em níveis baixos. Estes, por sua vez, sinalizam eventos para aqueles.

3.4.2. Modelo de concorrência baseado em eventos

Concorrência é obtida através do uso de eventos e tarefas, usando escalonamento em dois níveis. No nível de mais baixa prioridade estão as tarefas e no de mais alta estão os eventos.

Tarefas de um componente são atômicas entre si, executando até seu término, mas podem ser preemptadas por eventos externos. Podem executar comandos de componentes em níveis baixos na hierarquia, sinalizar eventos para componentes em níveis altos e agendar a execução de outras tarefas em um componente. *Eventos* são generalizações de tratadores de interrupção, propagando processamento para cima na hierarquia (através da sinalização de outros eventos) ou para baixo (por meio da execução de comandos). São executados quando sinalizados, preemptando a execução de uma tarefa ou outro evento.

4. Características do YATOS

Esta seção descreve o sistema operacional desenvolvido, o hardware utilizado, bem como suas principais estruturas e funcionalidades.

Na seção 3, pôde ser visto que existem diferentes sistemas operacionais que podem ser aplicáveis ao paradigma de RSSFs. No entanto, há algumas características que não são encontradas em nenhum deles ao mesmo tempo:

Ser dirigido por eventos: Um sistema dirigido por eventos, pode entrar no modo de menor consumo de energia sempre que possível. Além disso, elimina a espera ocupada. A tarefa esperando por algum recurso, ao invés de realizar espera ocupada, pode “dormir” e ser acordada por um evento que notifica a liberação do recurso. Conforme [11], um sistema dirigido por eventos chega a economizar até 12 vezes mais energia que um sistema tradicional e propósito geral.

Ocupar pouca memória de um nó sensor: O hardware de um nó sensor possui memória limitada. Neste espaço, além do código do sistema operacional, tem o código das aplicações e os dados coletados. Por isso, o espaço ocupado pelo sistema operacional deve ser o menor possível.

Consumir pouca energia: Energia é um recurso precioso em uma Rede de Sensores Sem Fio e seu consumo deve ser eficiente. O sistema operacional deve, sempre que possível, minimizar o consumo de energia dos componentes. Por exemplo, quando apenas uma computação estiver sendo realizada, o rádio pode ser colocado no modo de mais baixo consumo de energia de modo que não interfira na computação sendo realizada. A eficiência de uma rede de sensores depende da aplicação desta. O sistema operacional deve prover os mecanismos, como permitir o processador mudar de modo de operação de energia, para que a aplicação possa gerenciar a rede eficientemente.

Oferecer multi-tarefa baseada em prioridade: Pode ser necessário que mais de uma tarefa execute em um mesmo dado intervalo de tempo. Ainda mais, elas podem ter prioridades diferentes e/ou terem que ser executadas em uma ordem específica (neste caso, a elas são atribuídas prioridades que forcem a execução na ordem definida).

O sistema operacional deve ser modular: Um sistema modular facilita a manutenção e o desenvolvimento de novos componentes. Além disso, componentes não necessários podem ser excluídos.

O sistema operacional deve ser fácil de usar: A facilidade de uso é um requisito importante. De nada adianta um sistema operacional que ninguém consegue usar ou integrar com a sua aplicação. Além disso, o tempo gasto para aprender a usá-lo não deve atrapalhar o tempo de desenvolvimento da aplicação. Nem mesmo o tempo necessário para aprender uma nova linguagem de programação e paradigma de programação. Por esta razão, YATOS foi desenvolvido em *C*, uma linguagem de programação bem difundida entre programadores e desenvolvedores de sistemas embutidos. Para os casos em que eficiência era requerida e o código não estaria visível externamente, o desenvolvimento foi feito em linguagem de máquina.

4.1. Hardware

A arquitetura de sistema de um nó sensor genérico é formada por 4 componentes principais: suprimento de energia, comunicação, unidade de processamento e sensores. O bloco de suprimento de energia consiste, geralmente, de uma bateria, tendo como objetivo suprir o nó com a energia necessária. O bloco de comunicação é constituído de canal de comunicação sem fio, comumente um rádio de curto alcance. A unidade de processamento é formada pelos componentes de memória e um processador. A memória é usada para armazenar dados e aplicativos, enquanto o microcontrolador executa os aplicativos, além de usar ADCs para converter sinais analógicos para digitais, advindos do bloco sensor. Um esquema dessa arquitetura pode ser visto na figura 1.

O YATOS foi inicialmente desenvolvido para o nó sensor BEAN, do projeto *SensorNet* [15, 14], mostrado na figura 2. Ele usará um rádio CC1000 [4] e um microcontrolador TI MSP430F149 [12]. O rádio dispõe de freqüência programável (300 - 1000 MHz) com granularidade de 250 Hz, baixo consumo de corrente (Rx: 7,4 mA) e baixa tensão (2,1 - 3,6 V).

O microcontrolador escolhido usa uma arquitetura RISC de 16 bits, dispondo de 2 KB de RAM para armazenar dados e 60 KB + 256 B de memória Flash para armazenar código e constantes. Possui 6 modos de operação: 1 modo ativo (*AM*) e 5 modos de baixo consumo de energia (*LPM0*, *LPM1*, *LPM2*, *LPM3* e *LPM4*). Os modos de baixo consumo

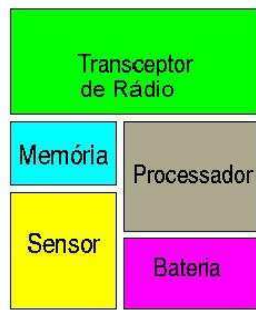

Figura 1: Componentes de um nó sensor.

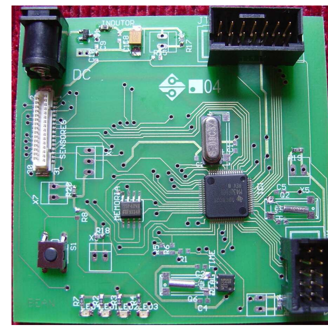

Figura 2: Foto do BEAN.

desligam a unidade central de processamento (CPU) e certas partes do microcontrolador. Através da escolha do modo de operação, pode-se configurar o microcontrolador para consumir apenas a energia necessária para seu funcionamento. Por exemplo: quando houver tarefas para executar no escalonador, o sistema se encontrará ativo; mas quando não houver mais tarefas para executar num dado momento, o microcontrolador entra em modo de economia de energia, apenas aguardando a ocorrência de algum evento. Como pode ser observado na figura 3, o modo mais econômico é o *LPM4*, mas este não é usado por desligar alguns periféricos utilizados pelo sistema. Para saber mais sobre os modos de operação, vide [13].

Outras características relevantes do microcontrolador:

- temporizadores programáveis;
- conversores ADC de 12 bits, que possibilitam converter sinais do ambiente para sinais digitais;
- comparadores que podem comparar valores lidos com um certo limiar e avisar quando este for ultrapassado.

Por fim, cada nó sensor possui um chip identificador, que armazena o número do nó na rede de sensores.

4.2. Princípios de funcionamento

O YATOS utiliza o termo *tarefa* para descrever um trecho de código executável. Normalmente, haverá muitas tarefas para serem executadas, sendo necessário que haja um *escalonador de tarefas* para decidir a ordem de execução. Foi escolhida a política de escalonamento cooperativa, na qual as tarefas, quando de posse do processador, decidem quando liberá-lo para a execução de outras. No caso específico do YATOS , isso ocorre quando a tarefa encerra sua execução em resposta a um dado evento. O escalonamento cooperativo foi escolhido devido às suas seguintes vantagens em relação ao modo preemptivo:

- uma tarefa nunca é interrompida inesperadamente;

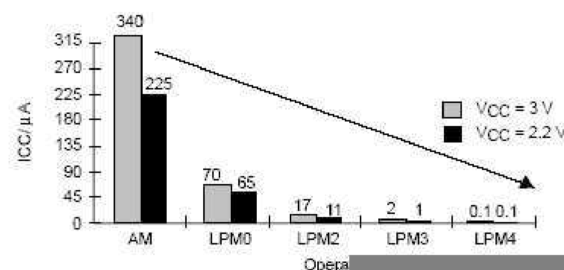

Figura 3: Consumo de corrente nos diferentes modos de operação.

- implementação mais simples, pois não há salvamento de contexto, economizando-se energia, memória e processamento.

O YATOS é dirigido por eventos. Isso significa que as tarefas criadas pelo programador vão executar em resposta a eventos do sistema e este estará em modo de operação ativo. Quando o escalonador de tarefas estiver vazio, o sistema entrará no modo de economia de energia.

4.3. Eventos

Eventos são interrupções que, quando tratadas por um manipulador, colocarão as tarefas que aguardam sua ocorrência no escalonador de tarefas, para serem executadas em ordem decrescente de prioridade. Os eventos válidos do YATOS são eventos que identificam a recepção de dados via rádio, notificação de término de conversão da leitura do sensor ligado em ADC, notificação do sensor de ultrapassagem de limiar e temporizadores. Cabe salientar que o YATOS é um SO de requisitos de tempo real fracos (*soft real-time requirements*).

4.4. Períodos de eventos

Eventos no YATOS podem ser classificados, de acordo com a freqüência com que são executados, em 3 tipos:

Aperiódicos: ocorrem sem o uso de temporizadores, através do disparo de interrupções pelo hardware.

Periódicos: ocorrem em intervalos de tempo definidos, sendo necessário o uso de temporização.

“Tiro único”: são eventos programados para ocorrer uma única vez, sendo posteriormente removidos da lista de eventos do sistema. Também necessitam de temporização.

4.5. Lista de tarefas

A lista de tarefas é um tipo abstrato de dados (TAD) responsável por armazenar todas as tarefas declaradas pelo programador. Através dela podem ser criadas novas tarefas, efetuar buscas por uma tarefa, alterar seus dados, e por fim destruir tarefas.

4.6. Lista de eventos

A lista de eventos é um TAD que contém listas de tarefas para cada evento válido do sistema. Quando são atribuídos eventos à uma tarefa, esta é copiada para a lista de tarefas do evento correspondente. Após ocorrer um certo evento, todos as tarefas na lista correspondente são então copiadas para o escalonador de tarefas.

4.7. Escalonador de tarefas

O escalonador de tarefas é um TAD implementado com uma fila de prioridades. As tarefas são executadas em ordem decrescente de prioridade, uma de cada vez, até seu fim (escalonamento cooperativo), sendo então removidas do escalonador. Associado a cada evento, existe um conjunto de tarefas que podem ser postas na fila do escalonador quando o evento ocorre, confome mostra a figura 4.

4.8. Rotinas para o programador

Foram desenvolvidas diferentes APIs de serviços para uso do microcontrolador e seus recursos, além de seus periféricos. A API *Tarefas* é responsável pela criação de tarefas, sua postagem e atribuição de eventos a elas. *Rádio* permite o envio e a recepção de dados

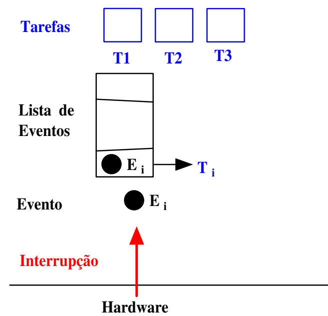

Figura 4: Lista de tarefas associadas a eventos.

através do rádio, configurar seu alcance e modo de operação. A API *Sensor* permite a obtenção de leituras dos sensores disponíveis no nó. *Memória* possibilita o apagamento, leitura e escrita da memória Flash. Por fim, a API *Microcontrolador* provê os seguintes serviços:

- iniciar componentes de hardware do nó e o sistema operacional;
- configurar seu modo de operação;
- iniciar e fechar seções críticas (regiões onde as interrupções são desabilitadas);
- configurar temporizadores;
- informar o identificador de cada nó sensor.

4.9. Exemplo de uso

O código 1 exibe um exemplo de aplicação em C utilizando o YATOS . A inicialização do hardware, declaração de eventos, declaração de tarefas, associação de eventos a tarefas e inicialização do SO é feita utilizando a API desenvolvida. Nesse código exemplo são criadas duas tarefas: a primeira executará apenas quando chamada pela segunda tarefa; a segunda executará em resposta a recepção de mensagens via rádio ou a eventos de temporização.

4.10. Estado atual de implementação

O YATOS foi implementado em C, linguagem difundida entre os desenvolvedores de sistemas embutidos. O espaço ocupado em memória da versão atual do YATOS é mostrado na tabela 1, tendo sido calculado sem considerar otimizações que o compilador pode efetuar.

Memória de código	3.144 bytes
Memória de dados	1.984 bytes
Memória de constantes	530 bytes

Tabela 1: Tabela com consumo de memória do YATOS .

5. Conclusões

Este artigo apresenta como maior contribuição a especificação de requisitos e o desenvolvimento de um sistema operacional especifico para RSSF. Diferentes SOs voltados para RSSFs foram analisados e é desenvolvido o YATOS , usando um escalonador de tarefas com política de escalonamento cooperativa. No paradigma de RSSFs, os componentes integrantes do nó e suas respectivas restrições computacionais foram responsáveis por moldar as escolhas de implementação das diferentes funcionalidades.

```c
#include "SO.h"    // APIs e rotinas do SO.
void rotina1(evento_t evento){
  /* Código da rotina (nao mostrado) */
  return;
}
void rotina2(evento_t evento){
  /* Código da rotina (nao mostrado) */
  switch(evento){
    case evRADIO:      // dado recebido via rádio.
      /* Código (nao mostrado) */
      break;
    case evTEMPORIZADOR0:  // evento de temporização.
      /* Código (nao mostrado) */
      if(condicao)
        Tarefa_Posta("RotinaA");  // Postagem de tarefa.
      break;
  }
  return;
}
void main(){
  /* Declaracao de variáveis.*/
  id_t id1, id2;
  // Inicia dispositivos de hardware.
  Microcontrolador_IniciaHardware();

  /* Declaracao de tarefas.*/
  id1 = Tarefa_Declara(&rotina1, 28, "RotinaA");
  id2 = Tarefa_Declara(&rotina2, 1, "");  // tarefa anônima.

  /* Atribuicao de eventos.*/
    // aperiodica.
  Tarefa_AtribuiEventos(id2, evRADIO, tpdNULO, pdNULO);
    // periodica: 200 ms.
  Tarefa_AtribuiEventos(id2, evTEMPORIZADOR0, tpdPERIODICO, 200);

  Microcontrolador_IniciaSO(); // Inicia o sistema operacional.
}
```

Código 1: Exemplo de aplicação usando o YATOS .

A área de pesquisa em RSSFs é uma área muito nova, sendo amplamente pesquisada por instituições de pesquisa no mundo todo. No caso desse projeto, muito trabalho ainda pode ser feito para extendê-lo e melhorá-lo. Dentre essas melhorias, podem-se citar as seguintes:

- Otimizar a alocação de memória no YATOS , minimizando o uso de alocação dinâmica, a fim de usar mais memória Flash e menos memória RAM para armazenar estruturas do YATOS .
- Recurso *shell*: usando a interface JTAG do nó conectada a um computador pessoal, um programador poderia digitar comandos e receber respostas a esses comandos a partir do nó, tais como: tarefa em execução, nós reconhecidos em sua vizinhança, etc.
- Mobilidade de código: possibilidade de envio/recepção de código para reprogramação via rádio dos nós.

6. Agradecimentos

Este trabalho foi parcialmente apoiado pelo CNPq, processos 55.2111/2002-3, 18.0381/2003-2, 18.0380/2003-6 e 830107/2002-9.

Referências

[1] A. Boulis and M. B. Srivastava. A framework for efficient and programmable sensor networks. In *Proceedings of OPENARCH 2002*, Junho 2002.

[2] M. Broxton, J. Lifton, D. Seetharam, and J. Paradiso. Pushpin computing system overview: a platform for distributed, embedded, ubiquitous sensor networks. In *Pervasive Computing 2002*, Agosto 2002.

[3] Alberto Cerpa, Jeremy Elson, Michael Hamilton, Jerry Zhao, Deborah Estrin, and Lewis Girod. Habitat monitoring: application driver for wireless communications technology. In *Workshop on Data communication in Latin America and the Caribbean*, pages 20–41. ACM Press, 2001.

[4] Chipcon. Smartrf cc1000 preliminary datasheet (rev. 2.1), *http://www.chipcon.com/files/CC1000_Data_Sheet_2_1.pdf*, 2002.

[5] S. Dulman and P. Havinga. Operating system fundamental for the eyes distributed sensor network. In *Progress 2002, Utrecht - Netherlands*, Outubro 2002.

[6] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David E. Culler, and Kristofer S. J. Pister. System architecture directions for networked sensors. In *Architectural Support for Programming Languages and Operating Systems*, pages 93–104, 2000.

[7] J. Kahn, R. Katz, and K. Pister. Next century challenges: Mobile networking for smart dust. In *ACM MobiCom'99*, Agosto 1999.

[8] J. Lifton, D. Seetharam, M. Seltzer, and J. Paradiso. Bertha: The os for pushpin computers. *Technical Report*, Maio 2002.

[9] Job Mulder. Peeros - preemptive eyes real-time operating system. Technical report, University of Twente, Abril 2003.

[10] L. B. Ruiz, L. H. A. Correia, L. F. Vieira, D. F. Macedo, E. F. Nakamura, C. M. S. Figueiredo, M. A. M. Vieira, E. H. Bechelane, D. Câmara, A. A. F. Loureiro, J. M. S. Nogueira, L. B. Ruiz, D. C. da Silva Jr., and A. O. Fernandes. Arquitetura para redes de sensores sem fio. In *Minicursos do XXII Simpósio Brasileiro de Redes de Computadores, Gramado/RS*, Maio 2004.

[11] Roy Sutton Suet-Fei Li and Jan Rabaey. Low power operating system for heterogeneous wireless communication systems. In *Workshop on Compilers and Operating Systems for Low Power 2001*, Setembro 2001.

[12] Texas_Instruments. Mixed signal microcontroller (rev. e), *http://www-s.ti.com/sc/ds/msp430f149.pdf*, 2003.

[13] Texas_Instruments. Msp430x1xx family user's guide, *http://www-s.ti.com/sc/psheets/slau049d/slau049d.pdf*, 2004.

[14] Luiz Filipe Menezes Vieira. YATOS e WISDOM: Plataforma de Software para Redes de Sensores sem Fio. Master's thesis, DCC-UFMG, Belo Horizonte-MG, Brasil, Abril 2004.

[15] Marcos Augusto Menezes Vieira. BEAN: Uma Plataforma Computacional para Redes de Sensores sem Fio. Master's thesis, DCC-UFMG, Belo Horizonte-MG, Brasil, Abril 2004.