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

Lucas Francisco Wanner

Um Ambiente de Suporte à Execução de Aplicações em Redes de Sensores sem Fios

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, dezembro de 2006

Um Ambiente de Suporte à Execução de Aplicações em Redes de Sensores sem Fios

Lucas Francisco Wanner

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

Prof. Dr. Raul Sidnei Wazlawick

Banca Examinadora

Prof. Dr. Antônio Augusto Medeiros Fröhlich
Orientador

Prof. Dr. Antonio Alfredo Ferreira Loureiro

Prof. Dr. Carlos Barros Montez

Prof. Dr. Luís Almeida

Prof. Dr. Mário Antonio Ribeiro Dantas

Para meus pais,

Jaime Ricardo Wanner

Filomena Rosa Hoff

Agradecimentos

Agradeço em primeiro lugar ao Professor Antônio Augusto Fröhlich, por apoio, dedicação e comprometimento que sempre superaram as minhas mais altas expectativas. O ambiente de pesquisa criado pelo Professor Fröhlich e meus colegas no LISHA foi fundamental para o sucesso deste trabalho. Muito obrigado especialmente aos colegas Arliones Hoeller Jr. e Augusto de Oliveira, pelas valiosas discussões técnicas e trabalho compartilhado em diversas áreas relacionadas a esta dissertação.

Muito obrigado também aos Professores Antonio Loureiro, Carlos Montez, Luís Almeida e Mário Dantas, que compuseram a banca deste trabalho de mestrado, e forneceram valiosas perspectivas e observações que influenciaram a revisão final desta dissertação.

Agradeço a minha mãe, Mena, e minha irmã, Joice, pelo seu amor e apoio constante. Finalmente, agradeço a minha amada Gabriella, por encher meus dias de alegria e felicidade.

Este trabalho contou com apoio material da CAPES (bolsa de mestrado), Fundo de Incentivo à Pesquisa/UFSC (equipamentos de redes de sensores sem fios), FINEP (equipamentos, recursos para viagem e bolsa de desenvolvimento através do projeto PDSCE), Atmel, Inc. (equipamentos), Pró-Reitoria de Pós-Graduação/UFSC e Diretoria do Centro Tecnológico/UFSC (recursos para viagem e participação em evento científico), e projeto Embedded WiSeNts/FP6 (recursos para participação em evento científico). Agradeço a todas instituições pela oportunidade.

Resumo

Em uma rede de sensores sem fios, diversos nodos sensores obtém dados do local onde se encontram e comunicam-se entre si, para gerar uma visão global de um objeto de estudo. A idéia de uma rede auto-gerenciada de dispositivos autônomos, de baixa potência, que colete dados de um ambiente e propague informações através de um enlace sem fios traz uma série de novos desafios e requisitos de suporte à execução de aplicações. Diversos projetos de pesquisa se propuseram a tratar o problema de suporte de sistema para redes de sensores sem fios. Entretanto, a maioria deles falha em tratar principalmente dois dos requisitos levantados neste trabalho: configuração transparente do canal de comunicação e abstração unificada e eficiente de hardware de sensoriamento.

Este trabalho apresenta o projeto e implementação de um ambiente de suporte à execução de aplicações em redes de sensores sem fios, baseado no sistema operacional EPOS, que inclui o projeto e implementação do protocolo de controle de acesso ao meio C-MAC (*Configurable MAC*) e um sistema de aquisição de dados de sensores. O projeto e implementação modular do protocolo C-MAC permitem que aplicações configurem o canal de comunicação de acordo com suas necessidades. O sistema de aquisição de dados de sensor desenvolvido é capaz de abstrair famílias de dispositivos sensores de maneira uniforme, sem ocasionar sobrecusto excessivo, e apresenta vantagens significativas com relação a outras soluções encontradas em outros sistemas operacionais para redes de sensores.

Abstract

In a wireless sensor network, several sensor nodes obtain data from the place where they are located and communicate among themselves in order to create a global vision of an object of study. The idea of a self-managed network of low-power, autonomous devices, that collects data from an environment and propagates information through a wireless link brings about several new challenges and requirements in application run-time support. Several research projects have aimed at solving the problem of system support for sensor networks. However, most of them fail in dealing with two requirements listed in this work: transparent configuration of the data communication channel, and efficient and unified sensor hardware abstraction.

This work presents the project and implementation of a run-time support environment for wireless sensor network applications based on the EPOS system. This environment includes the project and implementation of the C-MAC (*Configurable MAC*) medium access control protocol, which allows applications to configure the communication according to their needs, and a sensor data acquisition system, which abstracts families of sensing devices in an uniform fashion, without incurring excessive overhead, and presenting significant advantages in relation to the solutions found in other operating systems for sensor networks.

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

O dicionário Houaiss da língua portuguesa define *ubíquo* como algo que está ou existe ao mesmo tempo em toda parte; onipresente; que se difundiu extensamente; universal [Hou01]. Nos últimos sessenta anos os computadores passaram de máquinas de calcular gigantes a objetos ubíquos, presentes em todos os aspectos do dia-a-dia humano.

Dos primeiros *computadores programáveis* que surgiram na década de 1940 a partir da aplicação dos conceitos traçados por pioneiros como Alan Turing [Tur37] e Claude Shannon [Sha38] até a invenção do *microprocessador* na década de 1970, o *personal computer* na década de 1980, e a popularização das *redes de comunicação* e a Internet da década de 1990, os avanços na Ciência da Computação trouxeram o computador para o dia-a-dia de aproximadamente 600 milhões de pessoas em todo o mundo, de acordo com dados da *International Telecommunication Union* [ITU05].

A verdadeira ubiquidade dos computadores não está, entretanto, relacionada diretamente com os conceitos tradicionais de *computador pessoal* ou *mainframe*. Seiscentos milhões de pessoas representam menos de 10% da população mundial em 2006. Ao mesmo tempo, Tennenhouse [Ten00] mostra que apenas 2% dos mais de 8 bilhões de microprocessadores fabricados no ano 2000 foram utilizados em *computadores interativos*. Os 98% restantes equiparam semáforos de trânsito, freios de automóveis, robôs em plantas industriais, marcapassos, reprodutores de áudio digital, *video-games* e milhares de outros dispositivos de computação embarcada que integram em alguma escala a vida de todos os habitantes do planeta.

O constante avanço e miniaturização dos componentes eletrônicos está permitindo o surgimento de uma nova categoria de aplicações para o conceito fundamental de computador, na forma de micro-sensores sem fios de baixo consumo de energia. Estes micro-sensores são equipados com

um *módulo de sensores* analógicos ou digitais (e.g., sensor de temperatura, magnético, acústico), um *processador digital* capaz de tratar os sinais dos sensores, um *módulo de comunicação sem fios* (e.g., rádio de baixa potência) e um *módulo de energia* (e.g., bateria, célula solar). Cada sensor é capaz de obter uma *visão* do local em que se encontra e comunicar-se com outros módulos para gerar uma visão global de um ambiente qualquer.

A conexão destes dispositivos em *rede de sensores sem fios* traz fascinantes possibilidades de aplicação em virtualmente todos os campos de conhecimento humano. Uma rede de sensores pode, por exemplo, monitorar um habitat selvagem em uma floresta e fornecer dados ambientais sobre a fauna e flora locais. Outro conjunto de sensores pode ser utilizado no controle de um *ambiente inteligente*, onde as condições ambientais (e.g., temperatura, umidade, luminosidade) são alteradas dinamicamente de acordo com as preferências dos usuários e com as condições climáticas externas. Uma rede corporal de sensores pode fornecer dados biométricos sobre pacientes a médicos, auxiliando no diagnóstico de enfermidades. Muitos pesquisadores visualizam um futuro próximo povoado por circuitos eletrônicos microscópicos, alimentados por energia coletada do ambiente, formando uma rede de *poeira inteligente* [HSW$^+$00, KKP99].

A idéia de uma rede auto-gerenciada de dispositivos autônomos energizados por baterias de baixa capacidade ou por energia ambiente, que colete dados de um ambiente e propague informações através de um enlace sem fios traz uma série de novos desafios e requisitos tanto do ponto de vista de hardware quanto de suporte a execução de aplicações. Para serem instalados não intrusivamente e operar por longos períodos com uma fonte limitada de energia, os nodos devem ser pequenos e consumir pouca energia. Para suprir necessidades de sensoriamento de diferentes aplicações, mantendo uma mesma arquitetura básica, os nodos devem ter projeto modular, permitindo a conexão com sensores específicos para diferentes aplicações. Da mesma forma, o hardware de comunicação usado na rede deve permitir a mais ampla configuração possível do canal de transmissão de dados, para que diferentes aplicações possam se beneficiar de diferentes técnicas de modulação de dados e controle de acesso ao meio. Estes requisitos levaram à criação ou adaptação de uma série de tecnologias e protótipos e, conforme a complexidade destas tecnologias aumenta, torna-se crítica a necessidade de suporte de tempo de execução para mediar as capacidades do hardware e as necessidades de aplicações.

Os requisitos de sistema para redes de sensores sem fios são bastante amplos, e incluem a funcionalidade básica de um sistema operacional, serviços de gerência do consumo de energia, mecanismos para reprogramação, abstração de hardware de sensores heterogêneo e pilha de comunicações configurável. As restritas capacidades computacionais dos nodos de sensor fazem

ainda com que estes sistemas tenham que operar com recursos limitadas, e torna impossível o uso de sistemas operacionais tradicionais.

Diversos projetos de pesquisa [HDJH04, dAVV+04, BBD+02, DGV04, HSW+00, ABC+03, HKS+05] se propuseram a tratar o problema de suporte de sistema para redes de sensores sem fios. Entretanto, a maioria deles falha em tratar principalmente dois dos requisitos listados: abstração unificada e eficiente de hardware de sensoriamento e configuração transparente do canal de comunicação.

O sistema EPOS [Frö01, PFHD04, PF04] é um framework baseado em componentes para geração de suporte de execução a aplicações de computação dedicada. O projeto do sistema permite que programadores desenvolvam aplicações independentes de plataforma, e ferramentas de análise de aplicações permitem a geração de um sistema de suporte de tempo de execução que agregue todos os recursos que esta aplicação específica necessita, e nada mais. Por definição, uma instância do sistema utiliza somente os recursos necessários ao suporte da aplicação. Ao mesmo tempo, o repositório de componentes do sistema disponibiliza um grande conjunto de serviços tradicionais de sistema operacional através de interfaces independentes de plataforma. O sistema suporta diversas plataformas computacionais heterogêneas, como IA32, PowerPC, H8, Sparc e MIPS, entre outras [MHWF06, MHW+06]. No contexto deste trabalho, o EPOS ganhou também portes para plataformas de 8-bits AVR, tradicionalmente utilizadas em sistemas embarcados de baixa potência como redes de sensores sem fios. Trabalhos relacionados a esta dissertação adicionaram mecanismos para gerência do consumo de energia aos componentes do sistema [HWF06b, HWdO+06, HWF06a], tornando o EPOS um bom candidato para implementação de serviços de sistema operacional para redes de sensores sem fios.

1.1 Objetivos

O objetivo principal desta dissertação é desenvolver um ambiente de suporte à execução de aplicações em redes de sensores sem fios a partir do sistema operacional EPOS, fazendo uso da técnicas de engenharia de domínio denominada *Projeto de Sistemas Orientados a Aplicação* para modelar e implementar os componentes de software que se fizerem necessários, implementando-os sobre uma arquitetura de sistema embarcado que suporte redes de sensores sem fios e validando-os através de testes comparativos com outros sistemas existentes para redes de sensores sem fios. A fim de alcançar este objetivo principal, ficam estabelecidos os seguintes objetivos específicos:

- Estudar as tecnologias associadas às redes de sensores sem fios, incluindo hardware e protocolos de comunicação, tecnologias de sensoriamento e plataformas de hardware contemporâneas.
- Estudar e analisar os serviços presentes nos principais sistemas operacionais para redes de sensores sem fios contemporâneos.
- Modelar, implementar e testar serviços de sistema operacional para aquisição de dados de sensores e comunicação em redes de sensores sem fios.
- Incorporar os serviços implementados ao sistema EPOS, analisando os resultados obtidos.

1.2 Organização do Texto

O restante do texto desta dissertação está organizado da seguinte forma:

Capítulo 2 descreve os principais conceitos e tecnologias das redes de sensores sem fios.

Capítulo 3 descreve e analisa os principais sistemas operacionais para redes de sensores sem fios desenvolvidos na atualidade, e descreve o sistema EPOS, utilizado como base para a implementação dos serviços de redes de sensores desenvolvidos no contexto desta dissertação.

Capítulo 4 descreve e avalia os serviços de comunicação e sensoriamento modelados e implementados neste trabalho.

Capítulo 5 conclui a dissertação e apresenta os trabalhos em andamento e futuros.

Capítulo 2

Redes de Sensores Sem Fios

Sensores têm um papel fundamental em sistemas embarcados, desde aparelhos "inteligentes" até o controle e monitoramento de máquinas industriais. Um sensor eletrônico responde a estímulos como luz, pressão ou movimento e geram um sinal mensurável, que pode ser interpretado de maneira a extrair informação sobre um objeto de estudo. Avanços no projeto e tecnologias de hardware levaram a reduções em tamanho, consumo de energia e custo para dispositivos sensores. Tecnologias de Micro Sistemas Eletro-Mecânicos (MEMS – *Micro Electro Mechanical Systems*) integram elementos mecânicos, sensores e sistemas eletrônicos em um único substrato de silício, permitindo a atuação autônoma de dispositivos sensores.

Por diversos anos, aplicações industriais vêm se beneficiando de sistemas de sensores em redes, onde um conjunto de sensores é conectado por um barramento de campo (e.g., CAN, Profibus). A convergência recente de sensoriamento e tecnologias de comunicação sem fios de baixa potência permitiram o advento de Redes de Sensores sem Fios (RSSF). Em uma Rede de Sensores sem Fios, diversos *nodos sensores*, compostos por um conjunto de sensores analógicos e digitais, um micro-controlador, um transceptor sem fios e bateria, coordenam-se e trocam informações de maneira a prover uma visão global de um dado objeto de estudo. Cada nodo individual possui capacidade limitada, mas a comunicação e processamento cooperativo na rede permitem a obtenção de dados mais precisos. Com base na pesquisa e aplicações atuais [PK00, BBD+02], é possível definir que a arquitetura básica de hardware de um nodo de sensor, composta por um micro-controlador e um transceptor sem fios deve:

- Ter dimensões físicas reduzidas.

  Para poderem ser instalados de maneira não intrusiva, os nodos sensores devem ter dimensões reduzidas. Dado o constante avanço das técnicas de miniaturização de hardware, o

tamanho dos componentes eletrônicos utilizados nos nodos tende a diminuir constantemente. Entretanto, a miniaturização dos nodos sensores pode estar limitada ao tamanho da fonte de energia (seja na forma de baterias ou dispositivos para captura de energia ambiente).

- Ser capaz de operar por um longo tempo com quantidade limitada de energia.

  A necessidade de operação autônoma de um nodo sensor, e a capacidade limitada de energia disponíveis ao mesmo, fazem com que o baixo consumo de energia seja um fator determinante no projeto de hardware. Sendo assim, o projeto de um nodo sensor priorizará componentes de baixa potência e com suporte à gerencia do consumo de energia (e.g., microcontroladores, transceptores de baixa potência) em detrimento de componentes direcionados à alta capacidade de processamento, desempenho ou potência.

- Ter um projeto modular, permitindo a conexão com sensores específicos para diferentes aplicações.

  Os serviços de uma rede de sensores tendem a ser específicos, e utilizar somente o hardware necessário aos requisitos de cada aplicação [HSW+00]. Desta forma, é importante que o projeto seja modular, e permita a remoção e inclusão de sensores conforme as necessidades da aplicação.

- Permitir a mais ampla configuração possível do canal de transmissão de dados.

  O transceptor de dados sem fios é, em geral, o componente com maior consumo de energia em um nodo sensor [LH05]. Desta forma, é importante que este transceptor passe a maior parte do tempo desligado. Por outro lado, aplicações específicas terão padrões de comunicação específicos, e poderão se beneficiar de diferentes técnicas de modulação de dados e controle de acesso ao meio, que permitam o controle do consumo de energia sem comprometer a comunicação de dados. Desta forma, o transceptor deve permitir a maior configuração do canal de dados possível.

Baseado nestes requisitos, este capítulo descreve as principais tecnologias relacionadas com Redes de Sensores sem Fios, apresentando seus princípios de comunicação (seção 2.1), principais protocolos de controle de acesso ao meio (seção 2.3), relacionando as principais tecnologias de sensoriamento (seção 2.4) e plataformas de hardware contemporâneas (seção 2.5).

2.1 Princípios de Comunicação

A comunicação entre nodos em uma rede de sensores sem fios apresenta características significativamente diferentes da comunicação em redes sem fio tradicionais, como IEEE 802.11, redes de telefonia celular ou Redes Móveis Ad-hoc (MANETs – *Mobile Ad-hoc Networks*). Estas redes tradicionais são projetadas para fornecer boas características de vazão e atraso. As tarefas de organização, roteamento e gerência de mobilidade otimizam a Qualidade de Serviço (QoS – *Quality of Service*) e eficiência de banda larga [SGAP00]. As redes celulares e MANETs devem ainda operar em condições de ampla mobilidade. O consumo de energia é um problema secundário, já que as baterias dos sistemas que utilizam essas redes podem ser carregadas ou substituídas conforme necessário.

Uma rede de sensores pode ser composta por centenas de nodos projetados para operação autônoma e não interativa. Nodos podem ser instalados em locais onde a substituição de baterias é difícil ou impossível. A maioria dos nodos tem localização fixa, e mobilidade normalmente não é um problema importante. O tráfego é normalmente pequeno quando comparado às redes tradicionais, e o fluxo de dados é predominantemente unidirecional, indo dos nodos de sensor até um sorvedouro ou *gateway*. O objetivo principal no projeto de estratégias de comunicação em redes de sensores sem fios é maximizar o tempo de vida da rede, minimizando o consumo de energia nos nodos.

Akyildiz et. al. [ASSC02] listam algumas das principais diferenças entre redes sem fios ad-hoc tradicionais e redes de sensores sem fios:

- O número de nodos nas redes de sensores sem fios pode ser ordens de magnitude maior;
- Os nodos sensores são mais densamente instalados;
- Os nodos sensores tendem a falhar com o tempo;
- A capacidade de processamento e energia nos nodos é limitada;
- Possível ausência de um identificador global por nodo na rede de sensores.

As seções subseqüentes apresentam as características e principais tecnologias e protocolos de comunicação em redes de sensores sem fios.

2.2 Tecnologias de Hardware para Comunicação

Os rádios de baixa potência têm se mostrado a melhor alternativa para comunicação em redes de sensores. Tecnologias alternativas, como comunicação óptica a laser ou infra-vermelha, têm alcançado sucesso limitado, principalmente devido à necessidade de que os nodos comunicantes estejam em uma *linha de visão* [VdCdM03].

A maioria dos rádios transmite dados modulando o sinal de uma onda portadora, tipicamente em amplitude, freqüência ou fase. A modulação em amplitude é mais simples para codificar e decodificar, mas é mais sucetível a interferências, já que os dados são codificados na potência da transmissão, e ruídos externos são adicionados ao sinal. A modulação em freqüência é menos sujeita a ruídos, já que todos os dados são transmitidos no mesmo nível de potência [Hil03].

O uso de um Oscilador Controlado por Tensão (VCO – *Voltage Controlled Oscillator*) permite que os transceptores de rádio comuniquem-se em um intervalo amplo de freqüências, tornando possível a modulação em freqüência, e o uso de técnicas de espalhamento espectral. Tais técnicas aumentam a tolerância do sistema a interferências, ao espalhar o sinal a ser comunicado por uma ampla gama de freqüências [Hil03]. Entretanto, estas técnicas têm pouco uso em redes de sensores sem fios, já que o custo de manter a sincronização entre canais não é compatível com os requisitos de baixo consumo de energia dessas redes.

Rádios de banda estreita tipicamente suportam sistemas de modulação mais simples, deixando um nível de controle maior para o software e, por conseqüência, pagando um custo maior de processamento [Pol05]. Já rádios de banda larga apresentam técnicas de modulação sofisticadas como Seqüência Direta de Espalhamento de Espectro (DSSS – *Direct-Sequence Spread Spectrum*) [CEM01] e Chaveamento por Deslocamento de Fase (PSK – *Phase Shift Keying*) [GDS$^+$03], que são bastante robustos a ruídos, mas têm pouca flexibilidade e alto consumo de energia. A tabela 2.1 sumariza as características de alguns módulos de rádio utilizados em redes de sensores atualmente [RF 06, Chi04a, Chi04b], juntamente com rádios Bluetooth e IEEE 802.11 [Int02, Blu06] para comparação. A figura 2.1 sumariza as principais características dos rádios, através de gráficos de coordenadas polares. Na geração destes gráficos, os valores de taxa de dados, potência de recepção, envio e *standby* dos diferentes modelos foram colocados em escala logarítmica e normalizados, de maneira a ilustrar que, quanto maior a área do gráfico, mais adequado é o modelo para uso em redes de sensores sem fios. Os parâmetros de taxa de dados e potência de transmissão foram escolhidos por determinarem a velocidade e custo em termos de energia das transmissões na rede. Os parâmetros de potência de recepção e *standby* foram escolhidos pelo fato de que agregam

	Banda Estreita			Banda Larga	
Fabricante	RFM	Chipcon	Chipcon	Zeevo Bluetooth	Lucent 802.11
Modelo	TR1000	CC1000	CC2420	ZV4002	Orinoco Silver
Freqüência (Mhz)	916	300-1000	2400	2400	2400
Taxa max. de dados (kbps)	115.2	76.8	250	723.2	11264
Pot. de recepção (mA)	3.8	9.6	19.7	65	190
Pot. de envio (mA/dBm)	12 / 1.5	16.5 / 10	17.4 / 0	65 / 0	284 / 15
Pot. de *standby* (μA)	1	1	1	*	10
Tempo de *wakeup* (ms)	0.02	2	0.58	*	*
Modulação	OOK,ASK	FSK	DSSS,QPSK	DSSS,GFSK	DSSS
Interface	bit	byte	pacote,byte	pacote	pacote
Detecção de erros	não	não	sim	sim	sim
Buffer (bytes)	não há	1	128	*	*

* Dados não disponíveis

Tabela 2.1: Características de módulos de rádio

o maior custo total de energia nas redes, uma vez que os rádios passam a maior parte do tempo em *standby* ou escutando o canal de dados.

2.3 Protocolos de Controle Acesso ao Meio

Um protocolo de Controle de Acesso ao Meio (MAC - *Medium Access Control*) decide quando um nodo da rede pode acessar o meio, e tenta garantir que um nodo não interfira com a transmissão de outro. O MAC é também responsável por tratar ou sinalizar colisões para camadas superiores da pilha de protocolos.

A maioria dos protocolos MAC para redes sem fios projetados até hoje foram desenvolvidos e otimizados para enlaces de satélite ou redes locais sem fios (*Wireless LANs*). Os requisitos de uma rede de sensores são consideravelmente diferentes desses cenários, especialmente no que diz respeito à necessidade de operação autônoma e eficiente em termos de consumo de energia de uma rede de sensores. Em muitos cenários de aplicações de redes de sensores, parâmetros como baixa latência e alta vazão de dados têm importância menor [ASD+06].

Padrões especiais de comunicação em redes de sensores também podem exercer forte influência no projeto de um MAC para redes de sensores. O estudo dos padrões de comunicação de protótipos de redes de sensores instaladas em campo [LBV06, MCP+02] mostra que as taxas de

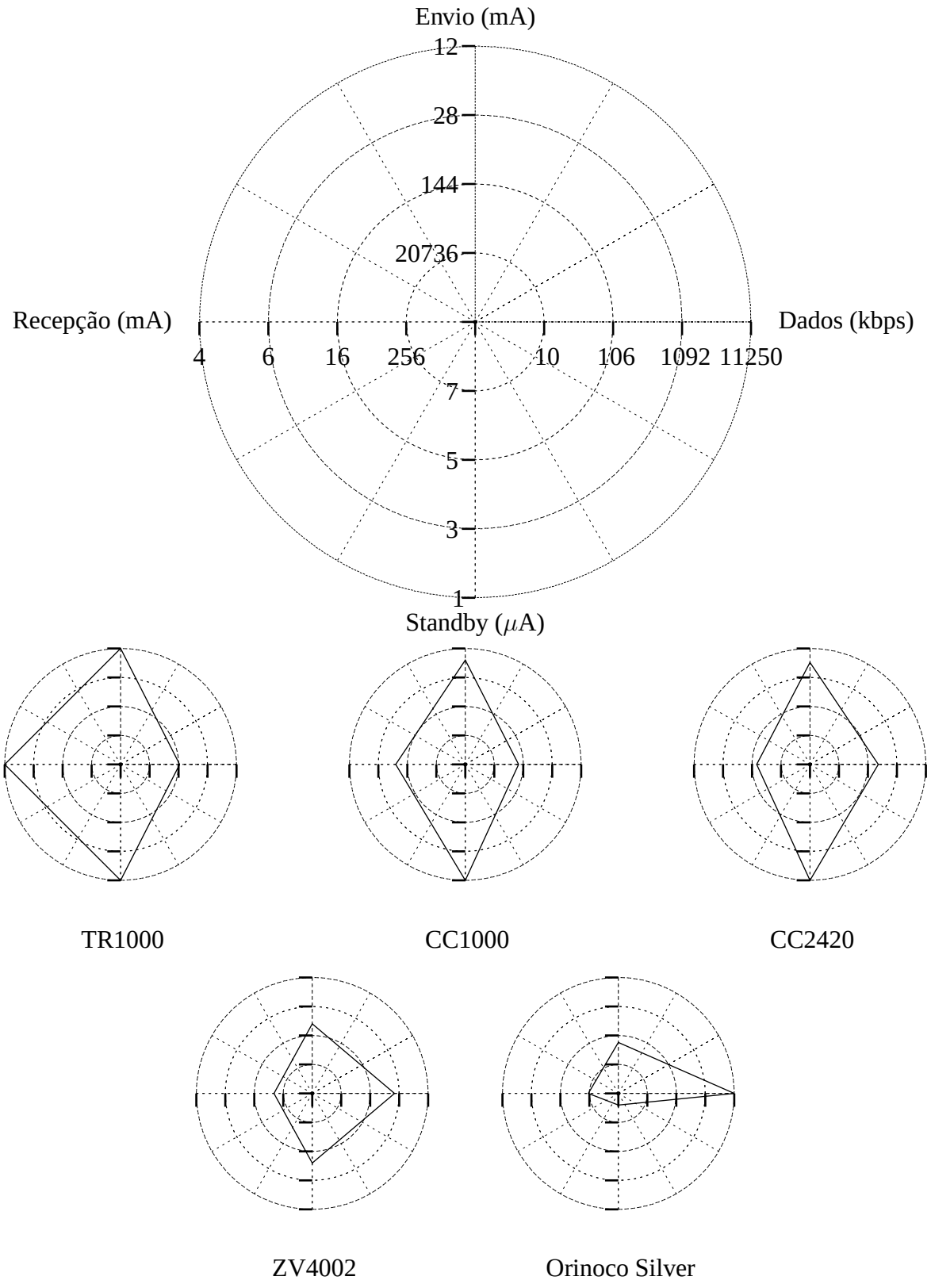
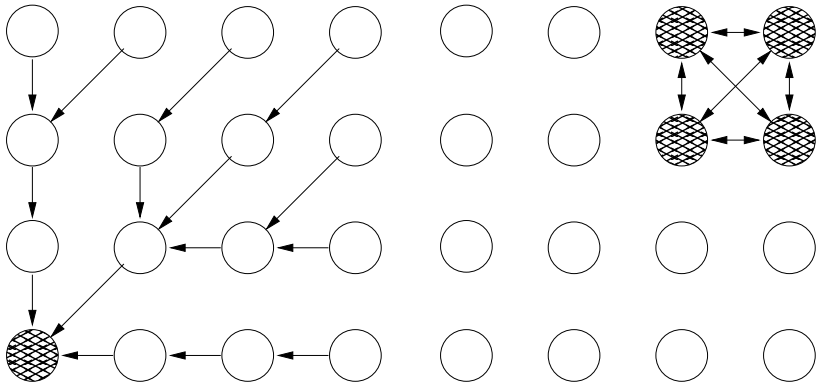


Figura 2.1: Características dos Rádios



(a) *Convergecast* (b) *Local Gossip*

Figura 2.2: Padrões de Comunicação em Redes de Sensores sem Fios

dados nos nodos são bastante baixas, da ordem de 1 - 200 bytes por segundo, com mensagens entre 20 e 25 bytes de carga útil (*payload*). A literatura apresenta ainda dois padrões de comunicação de dados em redes de sensores sem fios: *convergecast* e *local gossip* [LH05]. O primeiro diz respeito à tendência de pacotes similares atravessarem toda a rede de comunicação desde o local da coleta dos dados até um *gateway*, e o valor da agregação destes pequenos pacotes de nodos diferentes em um único pacote com macro-informações no caminho da transmissão. O segundo diz respeito ao fato de que, em uma rede densamente instalada, um mesmo fenômeno será detectado por diversos nodos distintos, e nodos que já possuem uma informação, muito provavelmente a receberão novamente através de seus vizinhos. A figura 2.2 ilustra estes fenômenos. A principal implicação destes dois padrões é que o tráfego na rede não é igualmente distribuído no espaço e no tempo. Nodos próximos ao *gateway* provavelmente receberão mais pacotes do que nodos nos limites da rede devido ao padrão de *convergecast*. Fenômenos aleatórios no tempo causarão tráfego intenso em determinadas áreas da rede devido ao padrão de *local gossip*.

Adicionalmente, nos protótipos atuais de redes de sensores sem fios, transmitir um único bit de informação pelo rádio consome mais energia do que executar algumas centenas de instruções no micro-controlador principal. Sendo assim, economia de energia é o principal fator a ser considerado no projeto de um MAC para redes de sensores. Langendoen e Halkes [LH05] identificam as principais fontes de desperdício de energia de um protocolo MAC para redes de sensores de baixo tráfego:

- Escuta sem tráfego: Se um nodo não sabe quando irá receber mensagens de um de seus vizinhos, terá que deixar seu rádio ligado em modo de recepção o tempo todo. Como o

custo de recepção é muito maior do que o custo de *standby*, esta é talvez a principal fonte de desperdício de energia.

- Colisões: Se dois nodos transmitem ao mesmo tempo, os dados são corrompidos, ambas transmissões devem ser repetidas e a energia gasta nas primeiras tentativas é disperdiçada.

- Escuta desnecessária: Como o canal de rádio é um meio compartilhado, um receptor pode ouvir pacotes que não são destinados para si.

- Flutuações de tráfego: Quando um fenômeno é detectado por muitos nodos vizinhos ao mesmo tempo (*local gossip*), os nodos competirão pelo canal de rádio, e gastarão energia detectando o canal e esperando uma janela de transmissão.

A escolha de um protocolo adequado para uma aplicação de redes de sensores sem fios depende do nível de compromisso entre eficiência de energia e flexibilidade de comunicação. Comparações em diferentes cenários de aplicação mostram que não há um *protocolo ótimo* para redes de sensores [EH02, HH04, LKR04, WC01, MB04, RR05, DL03, PHC04, YHE02]. Do ponto de vista do sistema operacional, o mais importante é dar flexibilidade de configuração para as aplicações. O restante desta seção detalha alguns dos protocolos MAC desenvolvidos para redes de sensores.

2.3.1 B-MAC

B-MAC (Berkeley MAC) [PHC04] é um protocolo MAC para redes de sensores sem fios baseado em acesso múltiplo com detecção de portadora (CSMA - *Carrier Sense Multiple Access*). Seu principal objetivo é dar flexibilidade de configuração à aplicações diferentes, permitindo que o protocolo seja configurado para cargas de trabalho (*workloads*) específicas.

O protocolo B-MAC utiliza um algorítmo de determinação da ocupação do canal (CCA - *Clear Channel Assessment*) baseado em leituras do indicador de força do sinal recebido do rádio (RSSI - *Received Signal Strength Indicator*); espera por intervalos aleatórios de tempo na transmissão (*backoff*) para arbitragem do canal; mensagens de confirmação (*acknowledgement*) ao final de cada transmissão para confiabilidade; e escuta em baixa potência (LPL - *Low Power Listening*) para comunicação. A figura 2.3 ilustra o funcionamento do protocolo, através do gráfico de tempo de comunicação entre três nodos em uma rede. Nesta figura, dois nodos tentam enviar dados a um terceiro nodo receptor.

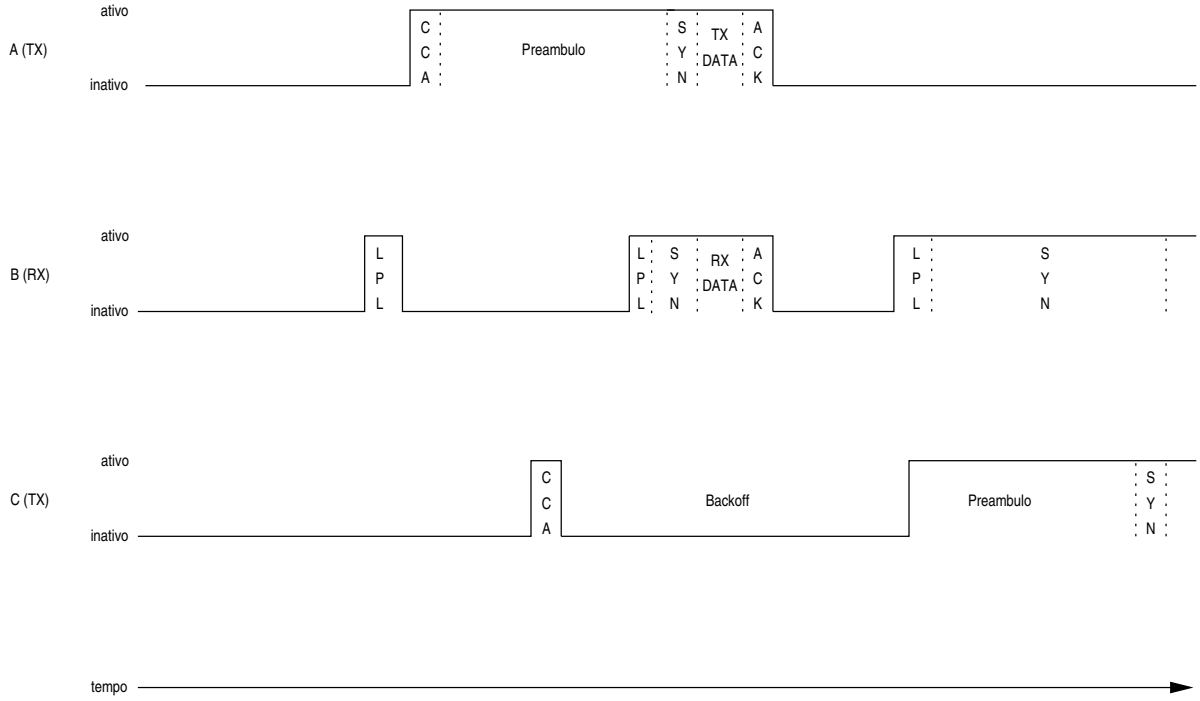


Figura 2.3: Funcionamento do protocolo B-MAC

O B-MAC periodicamente ativa o rádio, e verifica se há atividade no canal através de um algoritmo de busca por preâmbulo (LPL). Se há atividade detectada, o rádio fica ligado, e depois da recepção do pacote, volta a desligar. Se não há pacote detectado, o rádio é imediatamente desligado. Se um pacote é erroneamente detectado (falso positivo do LPL), o esgotamento de um período máximo para sincronização (*timeout*) permite que o rádio volte a ser desligado.

O período em que o rádio fica desativado na recepção é configurável, e para que os pacotes sejam efetivamente recebidos, o preâmbulo dos pacotes deve ser maior ou igual ao período de inatividade de recepção dos nodos. Se o canal é verificado a cada 100 ms, o preâmbulo deve ser de ao menos 100 ms, para que os nodos receptores possam acordar, detectar atividade no canal, receber o preâmbulo e finalmente receber a mensagem.

Uma falha conhecida dos protocolos baseados em CSMA (e, portanto, do B-MAC) é o problema do terminal oculto. Esse problema ocorre quando um nodo A pode ser escutado por um nodo B, mas não por um nodo C que também está se comunicando com B (figura 2.4). Como o nodo A está “oculto” para C, este último não detectará atividades no canal, e continuará enviando pacotes mesmo quando A já estiver ocupando o meio. Qualquer pacote enviado de C para B corromperá as transmissões acontecendo de A para B. A literatura apresenta uma série de possíveis soluções (não ótimas) para este problema [ZMS05, GLAT02, GLAT99].

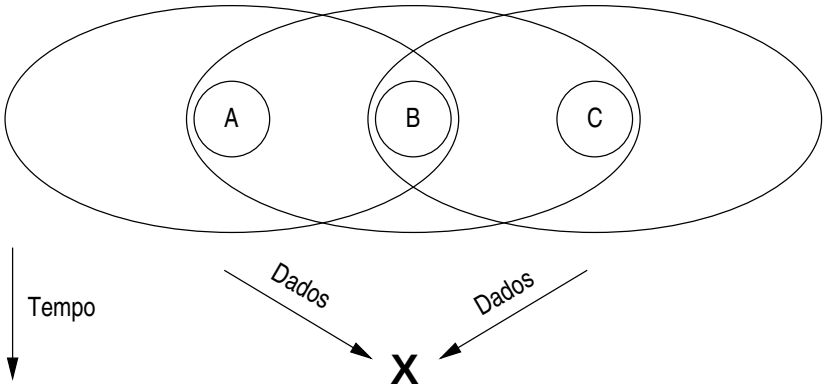


Figura 2.4: Problema do Terminal Oculto

2.3.2 S-MAC

S-MAC (Sensor MAC) [YHE02] é um protocolo de acesso ao meio organizado em *slots*, e usa um mecanismo RTS/CTS (*Request to Send / Clear To Send*) similar ao utilizado no padrão IEEE 802.11 para lidar com problemas de terminais ocultos. O protocolo funciona como uma *caixa preta* para comunicação em redes de sensores, combinando serviços de enlace, roteamento, organização, sincronização e fragmentação. O S-MAC não permite configuração dinâmica ou estática de seus parâmetros de funcionamento, e foi desenvolvido para otimizar cargas de trabalho específicas [PSC05].

A figura 2.5 ilustra o funcionamento do protocolo. Como os demais protocolos de controle de acesso ao meio em redes de sensores sem fios, o S-MAC periodicamente desliga o rádio, acorda, liga o rádio, escuta o canal e volta a desligar. Cada período ativo do protocolo é fixo, com um período inativo variável. Os nodos são organizados em células virtuais, nas quais os nodos possuem um mesmo relógio, e no começo de cada período ativo os nodos dentro de uma célula trocam informações de sincronização. Não há restrição de comunicação entre nodos de células diferentes, e nodos que ouvem mais de uma célula devem manter informações de sincronização para todas as células vizinhas, conforme ilustrado na figura 2.6 [PSC05].

Um período ativo do S-MAC é dividido em sincronização e transmissão de dados usando CSMA (*Carrier Sense Multiple Access*) e RTS-CTS. Caso um nodo precise fragmentar um pacote de dados para transmissão, este usa o esquema de RTS-CTS para reservar o canal, e transmite pacotes em rajada.

A solução do S-MAC para o problema do terminal oculto, em geral conhecida como acesso múltiplo com detecção de portadora e prevenção de colisões (CSMA/CA – *Carrier Sense Multiple*

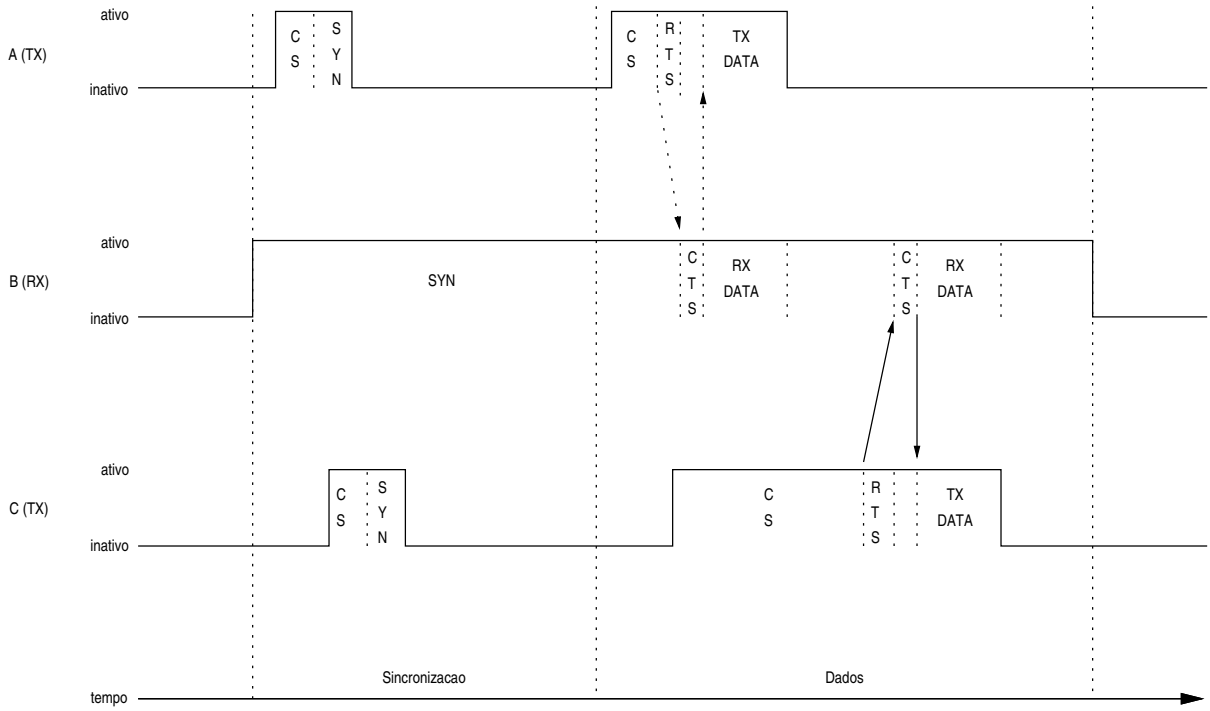


Figura 2.5: Funcionamento do S-MAC

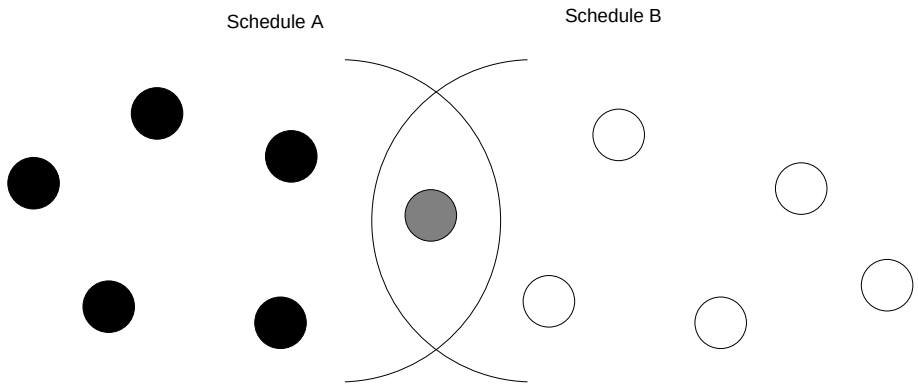


Figura 2.6: Sincronização entre nodos no S-MAC.

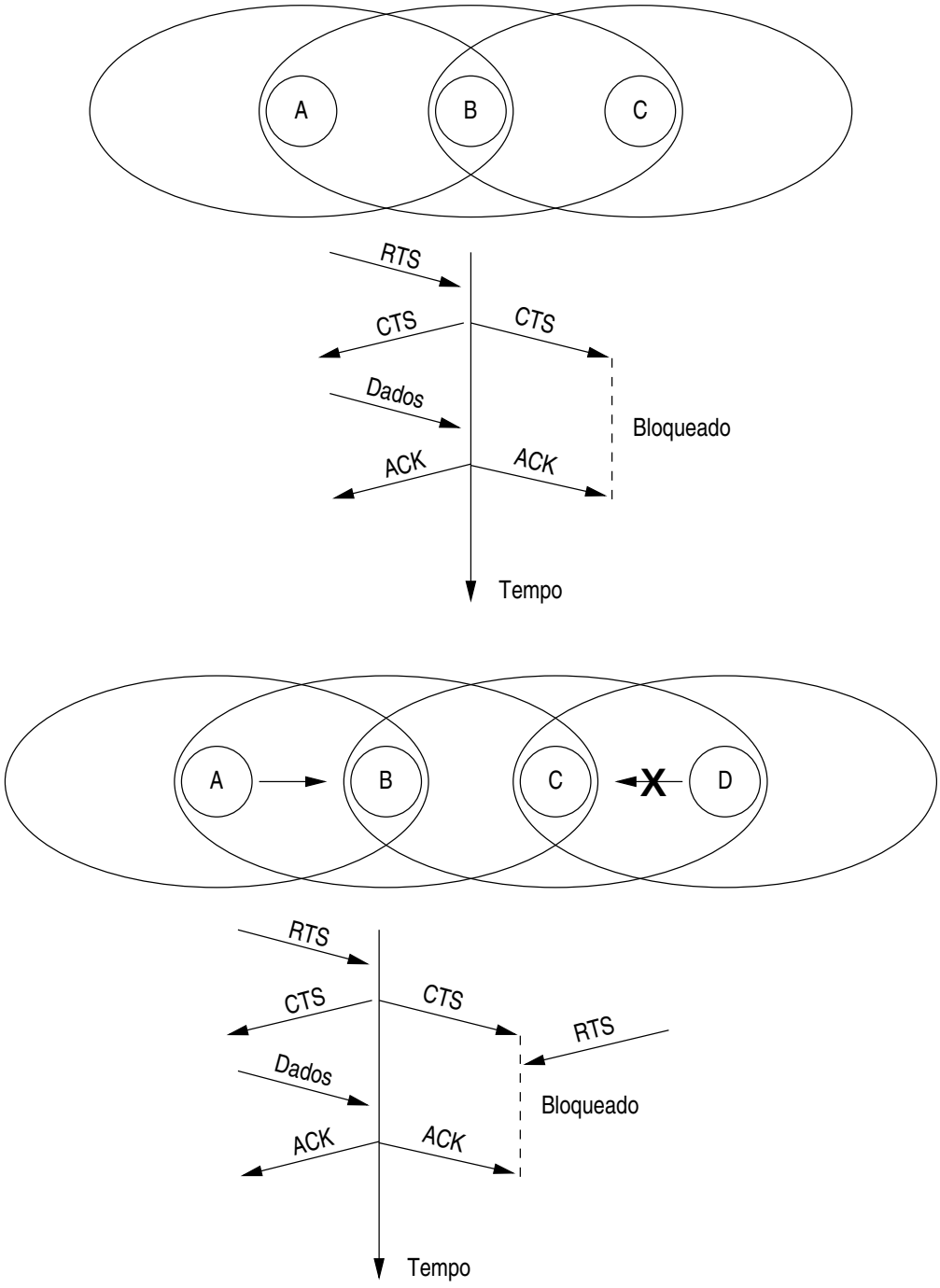


Figura 2.7: CSMA/CA e o Problema do Terminal Exposto

Access with Collision Avoidance), é efetiva na eliminação de colisões, mas introduz o chamado problema do terminal exposto (figura 2.7). O problema ocorre quando os sinais de controle do CSMA/CA silenciam nodos cujas transmissões não iriam interferir com outro par transmissor/receptor. Nas transmissões ilustradas na figura, a comunicação de A para B poderia ocorrer simultaneamente com a transmissão de D para C. Entretanto, C é silenciado pelo sinal de CTS de B, estando portanto "exposto" à transmissão entre A e B. Como os nodos expostos são proibidos de enviar, a vazão da rede pode ser reduzida.

2.3.3 T-MAC

T-MAC (Timeout-MAC) [DL03] é uma extensão adaptativa ao S-MAC, que permite a seleção automática do período ativo (*duty cycle*) do rádio, e adapta-se automaticamente às flutuações de tráfego inerentes aos padrões *convergecast* e *local gossip* presentes nas redes de sensores.

T-MAC utiliza o mesmo mecanismo de *células virtuais* do S-MAC para sincronização. O protocolo opera com *slots* de tempo fixo (615 ms) e usa um mecanismo de esgotamento de tempo (*timeout*) para determinar o fim de um período ativo. Se o nodo não detecta atividade dentro do período de tempo (15 ms), pode assumir que não há comunicação entre seus vizinhos, e pode desligar o rádio. Se o nodo ouvir ou iniciar uma comunicação, inicia um novo período depois do fim desta comunicação. Este período ativo adaptativo do T-MAC permite que este adapte-se às flutuações de tráfego na rede.

2.3.4 IEEE 802.15.4

O padrão IEEE 802.15.4 [Soc03] define um conjunto de protocolos para comunicação em redes de banda estreita e de baixa potência. O MAC do 802.15.4 é organizado em *slots*, e prevê topologias de estrela ou *peer-to-peer*. Os nodos de uma rede 802.15.4 podem ser Dispositivos de Funcionalidade Completa (FFDs – *Full Functionality Devices*) ou Dispositivos de Funcionalidade Reduzida (RFDs – *Reduced Functionality Devices*). Os primeiros podem comunicar-se com qualquer nodo na rede, e podem exercer o papel de coordenador da rede. Os últimos só podem comunicar-se diretamente com um coordenador. Estes organizam a rede enviando *beacons* que sinalizam a duração dos quadros de comunicação, e servem para sincronização dos nodos.

Depois de receber o *beacon* e sincronizar-se com um coordenador, os nodos podem solicitar a este uma associação. Depois da confirmação da associação, os nodos associados podem enviar

	B-MAC	S-MAC	T-MAC	802.15.4
Canais	1	1	1	1
Organização	Aleatória	Slots	Slots	Slots
Sincronização	-	Clusters Virtuais	Clusters Virtuais	Clusters (PANs)
Arbitragem	CCA	RTS-CTS	RTS-CTS	Coordenador

Tabela 2.2: Características dos MACs

os dados ao coordenador. O coordenador por sua vez anuncia os dados disponíveis aos nodos associados em sua mensagem de *beacon*. Os nodos podem então solicitar dados ao coordenador, que os envia. O *beacon* do coordenador anuncia também quais *slots* estão livres para transmissão dos nodos associados.

2.3.5 Comparação

Os protocolos de acesso ao meio para redes de sensores balanceiam desempenho (latência, vazão) por custo (consumo de energia). O consumo de energia é minimizado principalmente diminuindo o período em que o rádio ouve o canal sem que haja comunicação (*idle listening*) [LH05].

Protocolos baseados em contenção, como o B-MAC, são bastante robustos às flutuações de tráfego existentes em redes de sensores. A eficiência no consumo de energia é atingida aumentado o preâmbulo, o que permite que o canal seja verificado com uma periodicidade menor. Já os protocolos organizados em *slots*, como o S-MAC, T-MAC e o MAC do IEEE 802.15.4, reduzem o consumo de energia limitando a comunicação a períodos bem definidos. Comparações diretas [LH05, YH04] mostram que não há um único protocolo MAC para redes de sensores que possa ser considerado melhor do que os outros em todos os cenários de aplicação. Caracterísitcas como complexidade, necessidade de hardware especial (por exemplo para sincronização), e padrões de comunicação dados da aplicação devem ser levados em consideração na determinação do MAC ideal para determinado cenário. Entretando, alguns autores [PSC05] consideram que a adaptabilidade do protocolo é fundamental, e que protocolos com arquiteturas rígidas, como o IEEE 802.15.4 estão fadados à obsolência. A tabela 2.2 sumariza as características dos MACs apresentados.

2.4 Tecnologias de Sensoriamento

Um sensor é um dispositivo que responde a estímulos físicos (e.g., luz, temperatura, pressão, magnetismo ou movimento) e transmite um impulso resultante. Um sensor normalmente interage com um sistema digital, provendo informações sobre o *mundo analógico*. Essa informação pode ser fornecida através de um sinal analógico, que é convertido para valores digitais através de um conversor analógico-digital externo ao sensor, ou através de uma interface digital que converte internamente os sinais analógicos.

Interfaces de dados de sensores podem variar amplamente mesmo dentro da mesma classe de sensores. Um sistema de aquisição de dados deve abstrair estas diferenças com o mínimo *overhead* possível. De maneira a possibilitar um melhor entendimento dos desafios envolvidos no projeto e implementação de um sistema de aquisição de dados de sensores, esta seção apresenta alguns dispositivos sensores usados em nodos de sensor contemporâneos [Cro05b]. Esta não é uma lista exaustiva, mas deve prover um caso geral de uso de dispositivos sensores em redes de sensores sem fios. No final da seção, a figura 2.8 ilustra os sensores apresentados.

2.4.1 Termistores

Um termistor é um resistor que varia sua resistência com mudanças de temperatura. A equação de Steinhart-Hart é uma aproximação de terceira ordem amplamente utilizada para determinar a curva de resistência/temperatura de um termistor:

$$T = \frac{1}{a + b \ln R + c(\ln R)^3} \tag{2.1}$$

onde a, b, e c são parâmetros Steinhart-Hart específicos para cada dispositivo, T é a temperatura em graus Kelvin, e R é a resistência em Ohms. Um termistor normalmente é ligado a um conversor analógico-digital através de um circuito divisor de tensão simples. A estimativa de temperatura baseada em leituras do ADC pode depender do cálculo em tempo de execução de funções de aproximação complexas (e.g., a equação Steinhart-Hart), ou pode fazer uso de tabelas pré-calculadas de conversão. Termistores diferentes podem ter constantes de tempo e precisão diferentes, bem como diferentes constantes de dissipação de energia.

2.4.2 Sensores Digitais de Temperatura

A família SHT1x [Sen05] de sensores de umidade e temperatura provê um exemplo de sensores digitais de temperatura usados em redes de sensores sem fio. O sensor é fabricado pela Sensirion, e provê leituras digitais calibradas por uma interface de dois fios. Um micro-controlador pode ler dados do sensor enviando um comando *medir temperatura* para o sensor pela interface de dois fios. Ao completar da leitura, o sensor envia um sinal de *dados prontos*. O micro-controlador pode, então, ler os dados acompanhados de um código CRC para validação. Depois dessa transmissão, o sensor entra em modo inativo. Os coeficientes de calibração são programados na memória interna do sensor, e são usados internamente durante as leituras para calibrar os sinais dos sensores. Dados sensoriais providos pelo SHT1x podem ser convertidos para valores de temperatura através de uma função linear. Um *registrador de status* provê uma interface de detecção de baixa tensão, configura a resolução da leitura (por exemplo, 8 bits, 16 bits) e controla um aquecedor interno.

2.4.3 Foto-resistores e Foto-diodos

Um foto-resistor é um componente eletrônico cuja resistência diminui com o aumento da intensidade de luz incidente. A maioria dos foto-resistores são implementados através de células de sulfeto de cádmio, que usam a habilidade desse material de variar sua resistência (por exemplo, 2kΩ em condições de baixa luminosidade e 500Ω quando exposto à luz). Essas células também são capazes de reagir a uma ampla gama de frequências, incluindo infra-vermelho, luz visível e ultravioleta. Como no caso dos termistores, foto-resistores normalmente fazem interface com um conversor analógico-digital com um circuito divisor de tensão.

Um foto-diodo é um semicondutor que responde a estímulo óptico. Foto-diodos operam pela absorção de fótons que geram uma variação na corrente que flui através deles. Um circuito RC (resistor-capacitor) auxiliar tem sua fonte de corrente afetada por este sensor [UDT02], o que permite a detecção presença ou ausência de quantidades diminutas de luz através de medidas de tensão no resistor desse circuito auxiliar.

2.4.4 Sensores Digitais de Luminosidade

O sensor TSL2550 [TAO05], fabricado pela TAOS, provê um exemplo de sensor digital de luminosidade usado em redes de sensores sem fio. Ele combina dois foto-diodos e um conversor analógico-digital de 12 bits num circuito integrado para prover medidas de luz com uma sensitivi-

dade parecida com a do olho humano. Um dos foto-diodos é sensível a luz visível e infravermelha, enquanto o segundo foto-diodo é sensível primariamente a luz infravermelha.

Dados de sensoriamento são lidos através de uma interface de dois fios SMBus. Um *registrador de controle* controla o dispositivo, e dois registradores de ADC armazenam dados de sensoriamento para leitura. As saídas dos dois canais ADC podem ser usadas em uma função linear para se obter um valor que aproxima a resposta do lho humano na unidade comumente utilizada de Lux.

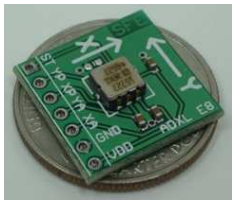
2.4.5 Magnetômetros

Magnetômetros são sensores capazes de medir força e/ou direção de campos magnéticos. Magnetômetros podem ser divididos em *magnetômetros escalares*, que medem a força total do campo magnético ao qual eles são expostos, e *magnetômetros de vetor*, que têm a capacidade de medir a componente do campo magnético em determinada direção. Um conjunto de magnetômetros de vetor podem ser combinados para permitir a definição de força, declinação e inclinação de um campo magnético.

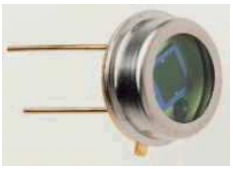
A família Honeywell HMC100x [Hon05] de sensores magnetoresistivos são dispositivos de pontes resistivas simples que simplesmente requerem uma tensão de fonte para medir campos magnéticos. Esses dispositivos são capazes de medir qualquer campo magnético ambiente ou aplicado em um eixo sensível. A tensão de saída do sensor é linear em relação ao campo magnético aplicado.

2.4.6 Acelerômetros

Acelerômetros são usados para medir mudanças na velocidade. Na sua forma mais simples, um acelerômetro é composto por uma massa suspensa e um dispositivo sensível à deflexão. A família Analog Devices ADXL202E [Ana05] provê um exemplo de acelerômetros com 2 eixos e baixo consumo de energia. O sensor provê tanto saídas analógicas quanto digitais. O valor de saída do sensor é linear em relação à aceleração aplicada, e calibração para baixas gravidades podem usar o campo gravitacional da Terra como referência.

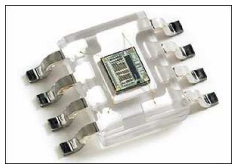


(a) Acelerômetro ADXL202

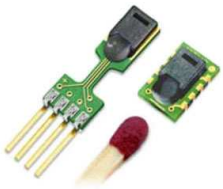


(b) Foto-diodo

(c) Foto-resistor

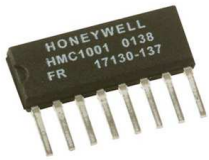


(d) Sensor de Luz TSL2550



(e) Sensor de Temperatura SHT11x

(f) Termistor



(g) Magnetômetro HMC1001

Figura 2.8: Exemplos de Dispositivos Sensores

2.4.7 Padrão IEEE 1451 para *Transdutores Inteligentes*

O padrão IEEE 1451 para *transdutores inteligentes* tem como objetivo padronizar as interfaces e protocolos entre *transdutores* como, por exemplo, sensores e microprocessadores, redes de controle e sistemas de aquisição de dados. Os principais elementos do IEEE 1451 são um *processador de aplicação com capacidade para operar em rede* (NCAP – *Network-Capable Application Processor*), um *módulo de interface para transdutores inteligentes* (STIM – *Standard Transducer Interface Module*) e uma interface independente de transdutor para sinais de controle e dados. O NCAP normalmente fornece um transceptor de rede e um micro-controlador implementando um protocolo de comunicação de redes. O STIM fornece a interface do transdutor e um *data sheet eletrônico do transdutor* (TEDS – *Transducer Electronic Data Sheet*) [CBBD98]. O TEDS é armazenado em memória não-volátil e contém campos que descrevem o tipo, atributos, operação e calibração do transdutor. O TEDS permite auto-descrição de transdutores para a rede, e elimina erros humanos associados à entrada manual de parâmetros. Também permite que a troca de transdutores por outros com maior precisão ou capacidades avançadas seja uma operação *plug-and-play* [Lee00].

A figura 2.9 ilustra os blocos do padrão IEEE 1451. O STIM *apresenta* os transdutores de forma independente de tecnologias específicas. Uma lógica de endereçamento de descrição

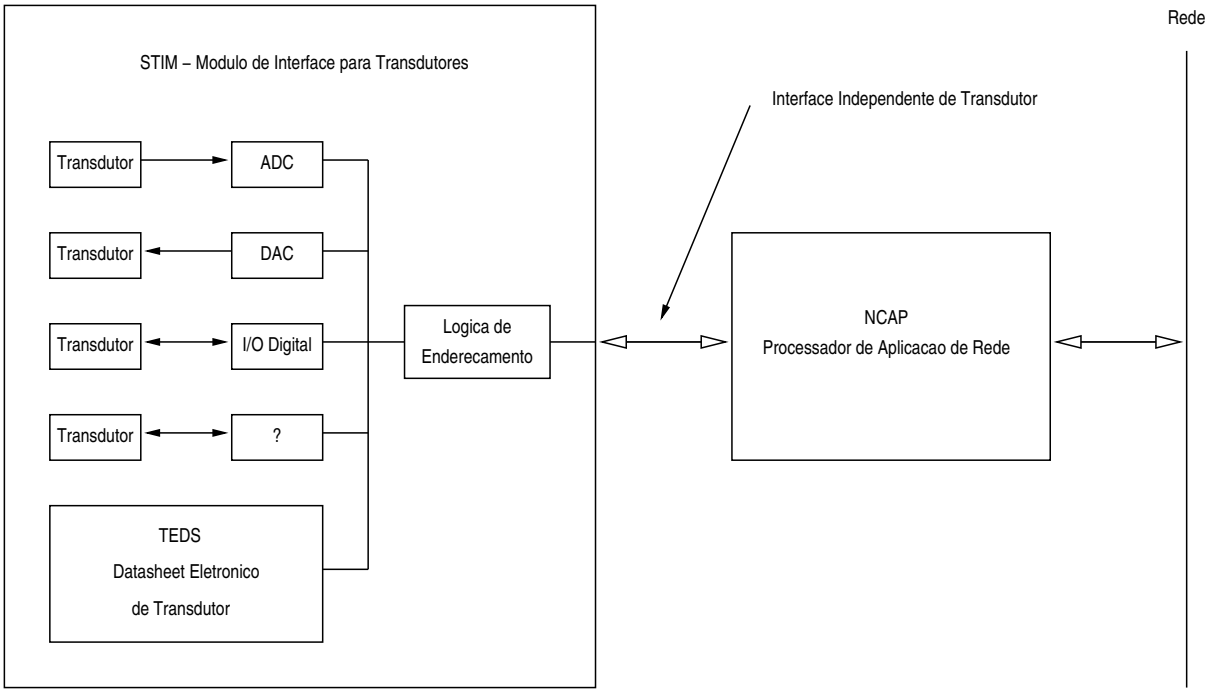


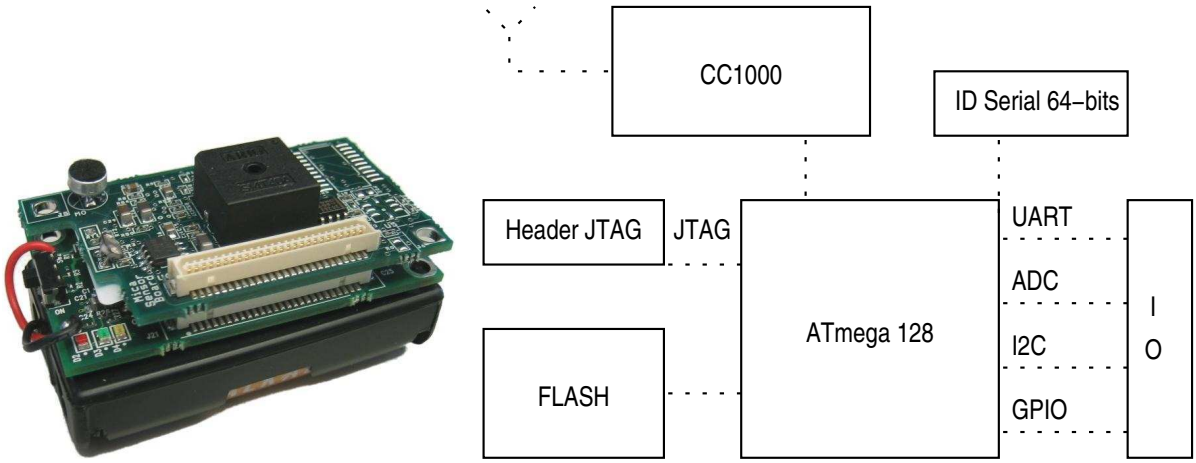
Figura 2.9: Blocos do Padrão IEEE 1451

permite acessar diferentes transdutores através de um mesmo STIM. Cada transdutor individual apresenta dados sobre suas características, funcionamento e coeficientes de calibração através do TEDS, cujos campos são definidos de acordo com a finalidade do transdutor (e.g., termistor, acelerômetro, DAC). O NCAP, por sua vez, exporta os sensores para uma rede (e.g., CAN, rede sem fios), através de um modelo de objetos definido pelo padrão.

Apesar das vantagens evidentes de uma interface padrão para transdutores de diferentes tipos, o padrão IEEE 1451, que começou a ser definido em 1997, em 2006 ainda não tem grande aplicação na indústria de transdutores.

2.5 Plataformas de Hardware

Esta seção apresenta as principais plataformas de hardware utilizadas atualmente em aplicações de redes de sensores sem fios.



(a) Plataforma (b) Organização do Hardware

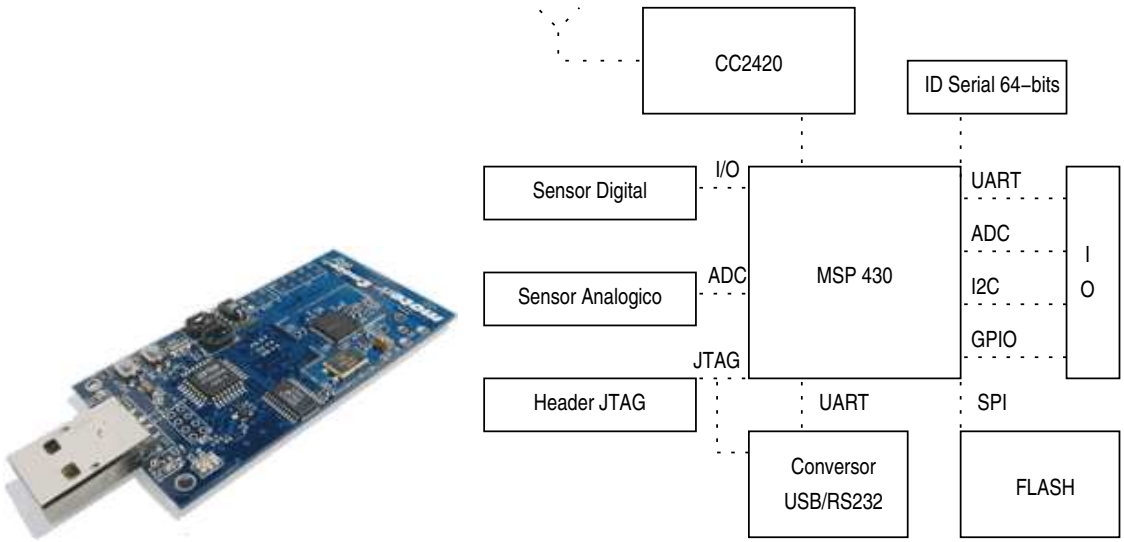
Figura 2.10: Mica2

2.5.1 Mica

Nos últimos anos a família de *motes* [1] [HSW+00] da Universidade da California em Berkeley evoluiu para uma plataforma estável para pesquisa em Redes de Sensores. Sua geração atual, o Mica2 (ver Figura 2.10) [Cro05a] usa um rádio de único canal da Chipcom CC1000, um microcontrolador Atmel Atmega128 de 8 bits, executando a 8MHz, com 4KB de memória RAM e 128KB de memória flash, e um chip de memória Flash de 512KB para armazenamento de dados. Um par de pilhas AA é usado como fonte de energia. Seu tamanho pequeno (aproximadamente 5 x 4 x 1.5 cm) permite a instalação em locações remotas com mínima interferência com o habitat existente.

O microcontrolador nos motes está conectado a quatro blocos de hardware (Rádio, LEDS, Memória Flash e Interface para Placas de Sensores / Programação). A figura 2.10 apresenta o hardware e um esquemático simplificado da arquitetura. O Mica2 não possui sensores integrados, mas conecta-se com dispositivos externos através de um conector de 51 pinos ligados a pinos de I/O da CPU. Este conector provê acesso aos pinos de GPIO (*General Purpose Input or Output*), UART e barramento I^2C do AVR, e é utilizado para programação do dispositivo e como interface para placas de sensor.

[1] Mote, s. Uma pequena partícula, como de poeira; qualquer coisa proverbialmente pequena. “The little motes in the sun do ever stir, though there be no wind” (Bacon). [Por13]



(a) Plataforma (b) Organização do Hardware

Figura 2.11: Telos

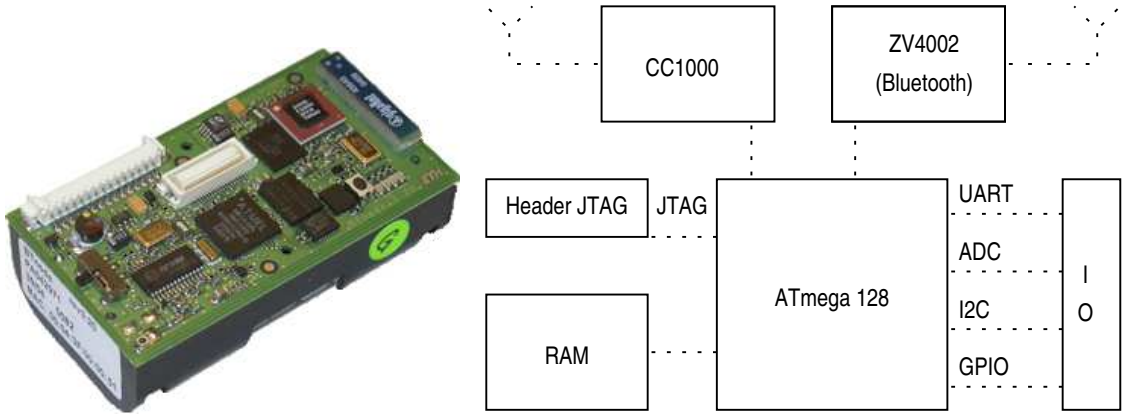
2.5.2 Telos

A plataforma Telos [Mot05], desenvolvida também na Universidade da California em Berkeley, utiliza como base um microcontrolador Texas Instruments MSP430, de 16 bits, com 10KB de memória RAM em 60KB de memória flash, executando a 8 MHz. Para comunicação, o Telos utiliza um rádio Chipcon CC2420, compatível com IEEE 802.15.4 (ver seção 2.2). A plataforma conta ainda com um chip de memória flash de 1MB para armazenamento de dados, e sensores de luz e temperatura integrados.

Não há módulo de expansão similar aos do Mica2 no Telos, mas a plataforma externaliza alguns pinos do micro-controlador para entrada no conversor analógio/digital, e para comunicação serial. Cada nodo conta com um chip conversor USB/RS232, que permite *logging* de dados e programação sem hardware externo.

2.5.3 BTnode

A plataforma BTnode [ETH05] (Figura 2.12), desenvolvida na ETH Zurique, integra um micro-controlador, módulo de rádio de baixa-potência e um módulo de rádio Bluetooth. O dispositivo não apresenta sensores integrados, mas fornece diversas interfaces de propósito geral, que podem ser utilizadas com periféricos como sensores, atuadores, DSPs, dispositivos seriais (e.g., GPS, leitores RFID, etc.). Os módulos BTnode medem 6 x 4 x 1.5 cm [BKM+04].



(a) Plataforma (b) Organização do Hardware

Figura 2.12: BTnode

O hardware do BTnode faz uso do microcontrolador Atmel ATmega128, o mesmo utilizado nas plataformas Mica (ver Seção 2.5.1). A memória de dados do micro-controlador é extendida por um chip SRAM de baixa potência de 60 KB. A plataforma é equipada com um relógio de tempo real de 32 kHz, três chips de 60 KB de SRAM externos ao processador para cache de dados, quatro LEDs para depuração, e é programável através de um programador serial ISP (*In System Programming*) ou interface JTAG.

A revisão 3.2 do BTnode traz um módulo Bluetooth Zeevo ZV4002 [Blu06] é conectado a uma das portas seriais do micro-controlador. Um módulo de rádio de baixa potência Chipcon CC1000 é ligado através de portas GPIO e interface SPI (*Serial Peripheral Interface*) do micro-controlador, e opera a 868 MHz [ETH05]. O sistema é alimentado por duas pilhas AA comuns. A figura 2.12 apresenta a organização de hardware do BTnode.

2.5.4 Comparação

A tabela 2.3 sumariza as características das plataformas apresentadas, juntamente com os dados da plataforma René [HSW+00] (protótipo precursor da família Mica) e da plataforma iMote [KAH+04], desenvolvida pela Intel como um *gateway* para redes de sensores sem fios. A figura 2.13 sumariza esses dados na forma de gráficos de coordenadas polares.

Mote					
Tipo	René	Mica2	iMote	BTnode	Telos
Ano	2000	2003	2003	2003	2004
Instituição	UCB	UCB	Intel	ETHZ	UCB
CPU					
μcontrolador	AVR	AVR	ARM	AVR	MSP430
Clock	4 MHz	8 MHz	12 MHz	8 MHz	8 MHz
Memória de Programa	8 KB	128 KB	512 KB	128 KB	60 KB
RAM	0.5 KB	4 KB	64 KB	64 KB	10 KB
Dispositivos de Rádio					
Tipo	RFM	Chipcon	Bluetooth	Bluetooth	802.15.4
Frequência (MHz)	916	433/916	2400	2400	2400
Taxa de Transferência (kbps)	10	40	700	700	250

Tabela 2.3: Plataformas de Hardware para Redes de Sensores sem Fios

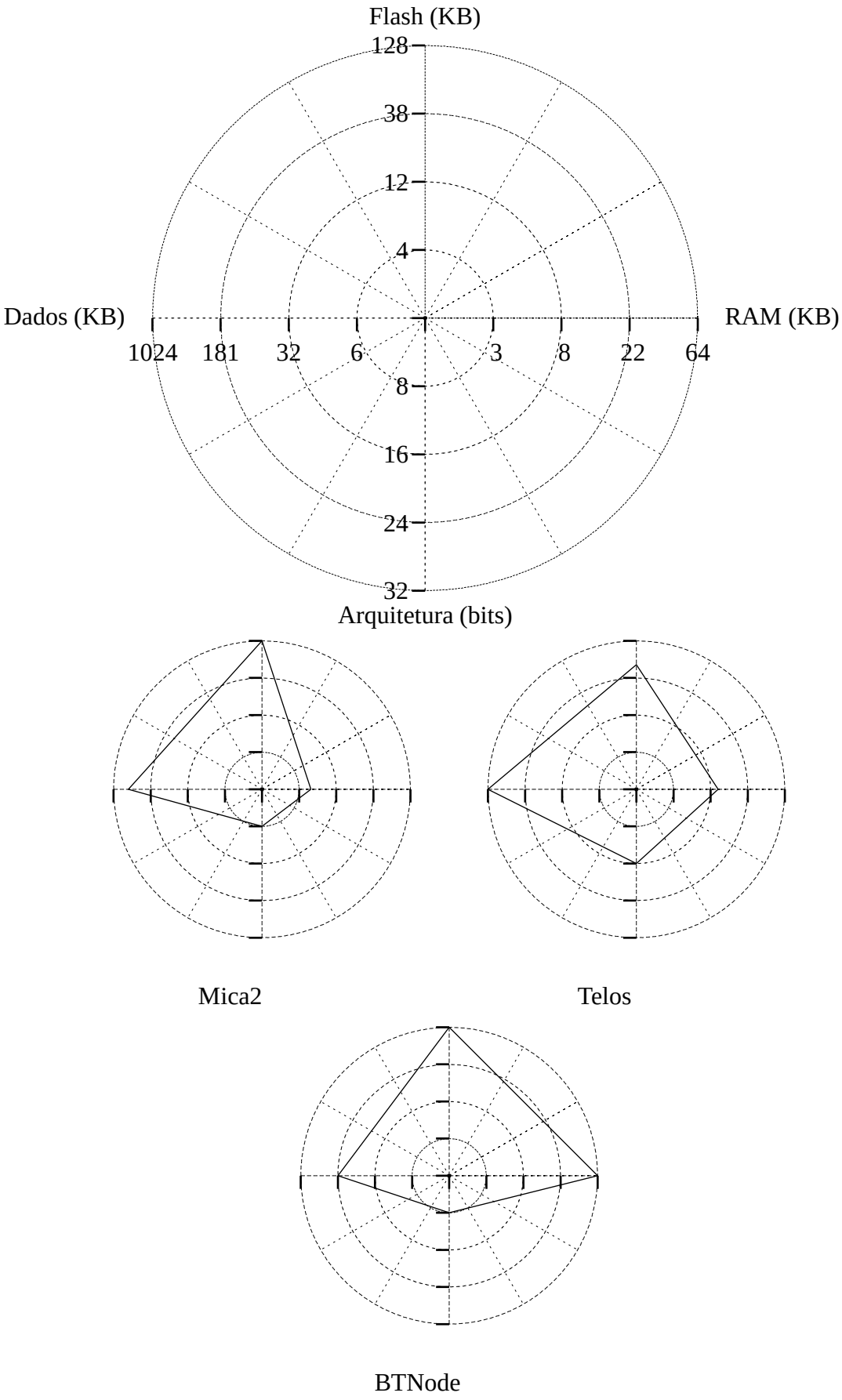


Figura 2.13: Características dos Motes

Capítulo 3

Sistemas Operacionais para Redes de Sensores

Em uma rede de sensores, requisitos específicos de aplicações guiam o projeto de hardware, desde a capacidade de processamento até a largura de banda de rádio e módulos de sensores, o que exige que o projeto de hardware seja modular. Entretanto, estes requisitos levaram ao surgimento de uma grande variedade de componentes de hardware, fazendo com que as plataformas de redes de sensores sem fios sejam não apenas modulares, mas também heterogêneas. Sendo assim, uma aplicação desenvolvida para uma dada plataforma de sensores dificilmente será portável para outra diferente, a menos que o sistema de suporte de execução nestas plataformas empregue mecanismos adequados para abstração e encapsulamento da plataforma de sensoriamento. Ao mesmo tempo, as limitações de recursos (e.g. memória, capacidade de processamento, energia) tipicamente encontradas nas redes de sensores, exigem que qualquer suporte de tempo de execução nesses sistemas seja eficiente e não consuma recursos excessivos.

Sendo assim, a necessidade de conectividade, abstração de hardware e gerência de recursos limitados torna o suporte no nível de sistema operacional imperativo para aplicações em redes de sensores sem fios. Baseado na tecnologia (apresentada no capítulo 2), aplicações e pesquisa atuais [HSW+00], é possível listar uma série de requisitos de *sistemas operacionais* para redes de sensores sem fios. Um sistema de suporte à execução de aplicações para uma rede de sensores deve:

- Fornecer a funcionalidade básica de um sistema operacional.

  De maneira a não restringir a funcionalidade e portabilidade das aplicações de redes de sensores, um sistema operacional para esses dispositivos deve prover serviços tradicionais

como: abstração de hardware, gerência de processos (normalmente segundo o prisma de monotarefa, *multi-thread*), serviços de temporização, e gerência de memória.

- Fornecer mecanismos para gerência do consumo de energia nos nodos.

  O gerenciamento eficiente do consumo de energia nos nodos é um fator determinante para o tempo de vida da rede. Um sistema de suporte à execução de aplicações para redes de sensores deve fornecer mecanismos de gerência de energia às aplicações, bem como utilizar os recursos de hardware de maneira a consumir o mínimo de energia necessário à execução adequada das aplicações.

- Prover mecanismos para reprogramação em campo.

  Dado que os nodos de uma rede de sensores sem fios podem estar localizados em regiões de difícil acesso, e que os requisitos e parâmetros das aplicações de sensoriamento podem mudar com o tempo, a reprogramação em campo, via rede de comunicação, é um fator importante neste tipo de redes. Um sistema operacional para redes de sensores idealmente deve prover algum mecanismo de reprogramação total ou parcial de aplicações já instaladas.

- Abstrair o hardware de sensoriamento heterogêneo de maneira uniforme.

  Os requisitos de modularidade das aplicações de rede de sensores fazem com que o hardware utilizado seja amplamente heterogênio. Considerando isto, uma aplicação de sensoriamento, desenvolvida para determinada plataforma, dificilmente será portável para outra diferente, a menos que os sistemas de suporte de tempo de execução destas plataformas abstraiam e encapsulem adequadamente a plataforma de sensoriamento. Além das diferenças arquiteturais, módulos sensores (e.g. sensores de temperatura, luz ou movimento) apresentam uma gama de variabilidade ainda maior. Módulos sensores apresentando a mesma funcionalidade muitas vezes variam em suas interfaces de acesso e características e parâmetros operacionais.

- Fornecer uma pilha de protocolos de comunicação configurável

  Dadas as necessidades específicas de comunicação de diferentes aplicações, e o requisito de que o hardware de comunicação seja amplamente configurável, é papel do sistema operacional prover um mecanismo de configuração dos protocolos de configuração da pilha de comunicações a ser utilizada na rede, especialmente no que diz respeito aos protocolos de controle de acesso ao meio.

- Operar com recursos limitados

	Plataformas	Memória	Proteção	Configuração	Preempção
VxWorks	PowerPC, 68K IA32, MIPS, ARM	256 $\sim$ 512 KB	Sim	Dinâmica	POSIX
μClinux	68K, H8/S, ARM Microblaze, i960 Blackfin	$\sim$ 1 MB	Não	Dinâmica	POSIX
QNX	PowerPC, IA32, SH, MIPS	$\sim$ 64 KB	Sim	Dinâmica	POSIX
OS-9	PowerPC, IA32, SH, MIPS, SPARC	32 $\sim$ 64KB	Sim	Dinâmica	Processo

Tabela 3.1: Características de Sistemas Operacionais Embarcados

O requisito de baixo consumo de energia para redes de sensores traz como implicação prática a escolha de micro-controladores de baixa potência para executar as funções de processamento nos nodos. Esses são geralmente bastante restritos em sua capacidade computacional, e possuem pequenas quantidades de memória disponível. Um sistema operacional para redes de sensores deve entregar os serviços necessários à aplicação, sem consumir uma parcela significativa dos recursos computacionais disponíveis nos nodos.

Sistemas operacionais típicos para computação embarcada, como VxWorks [Win03], QNX [Hil92], OS-9 [Rad01], WinCE [Mic06] e μClinux [Emb06] fornecem um ambiente de programação semelhante ao existente em computadores de mesa, em geral através de serviços compatíveis com POSIX. Muitos destes SOs fornecem e, por conseqüência, exigem suporte de hardware à proteção de memória. Apesar de serem bastante adequados para o uso em telefones celulares, *set-top-boxes* e aplicações embarcadas complexas, os requisitos de processamento e memória desses sistemas faz com que eles sejam inadequados para o uso em redes de sensores sem fios. A tabela 3.1 sumariza as características de alguns desses sistemas, e mostra que suas configurações estão muito além das existentes tipicamente em um nodo de sensor.

Entre os sistemas operacionais desenvolvidos especificamente para redes de sensores sem fios, cabe citar o AmbientRT [HDJH04], YaTOS [dAVV+04], MagnetOS [BBD+02] e Contiki [DGV04]. Entretanto, os mais proeminentes, e que mais se aproximam dos requisitos levantados nesta dissertação são os sistemas TinyOS [HSW+00], MANTIS OS [ABC+03] e SOS [HKS+05].

Este capítulo descreve e analisa os sistemas operacionais atualmente utilizados em redes de sensores sem fios. As seções 3.1, 3.2 e 3.3 apresentam em detalhe os sistemas TinyOS, MANTIS OS e SOS, respectivamente. A seção 3.4 apresenta brevemente outros sistemas relevantes. A seção 3.5 apresenta uma análise e comparação com base no cumprimento de requisitos de cada um desses sistemas, e a seção 3.6 apresenta o sistema EPOS, usado como base para os serviços de comunicação e sensoriamento desenvolvidos neste trabalho.

3.1 TinyOS

TinyOS é um sistema operacional baseado em eventos desenvolvido para uso em redes de sensores sem fios. O sistema foi desenvolvido originalmente no contexto do projeto NEST da Universidade da California em Berkeley, coordenado pelo Professor David Culler. Atualmente, TinyOS é o sistema operacional mais utilizado em redes de sensores, e é mantido através de um projeto de código aberto [Tin06]. Existem portes do sistema para as plataformas Mica, Telos e BTnode.

O TinyOS é organizado como uma coleção de componentes de software. O conjunto composto por aplicação e sistema é chamado no TinyOS de *configuração* e consiste de um escalonador e um grafo de componentes. Cada componente é composto por tratadores de *comandos*, tratadores de *eventos*, *tarefas* e um *frame* de execução. Cada componente declara os comandos aos quais responde, e os eventos que sinaliza. Comandos, eventos e tarefas executam no *frame* de um componente, e alteram seu estado. A composição de componentes no TinyOS cria camadas de componentes, onde componentes de nível mais baixo respondem a comandos e sinalizam eventos para componentes de nível mais alto.

Comandos são chamadas de métodos não bloqueantes, e são usados tipicamente para iniciar solicitações de hardware e software e, condicionalmente, iniciar tarefas. Tratadores de eventos são invocados para lidar com interrupções de hardware, e podem chamar comandos e postar tarefas. Tarefas são chamadas de métodos com execução adiada. Essas são atômicas com relação a outras tarefas, executam até a sua completude, e só podem ser preemptadas por eventos. O escalonador do TinyOS utiliza uma fila FIFO de tamanho fixo para postagem de tarefas [HSW+00]. Quando a fila de tarefas está vazia, o escalonador desliga o processador, mantendo os periféricos ligados para que esses possam sinalizar novas interrupções. Uma vez que a fila esteja vazia, novas tarefas só serão postadas como resultado da execução de tratadores de eventos.

O TinyOS é desenvolvido em nesC [GLvB+03], uma linguagem de programação e

especificação de componentes inspirada em linguagens de descrição de hardware. A linguagem foi desenvolvida especialmente para suportar o modelo de componentes do TinyOS, e seus princípios fundamentais são [GLCB03]:

- Especificação de comportamento em termos de interfaces bi-direcionais. Interfaces podem ser fornecidas ou utilizadas por componentes. As interfaces fornecidas por um componente representam a funcionalidade que o componente fornece, e as utilizadas representam a funcionalidade que este componente precisa para realizar suas funções. As interfaces representam o conjunto de serviços implementados pelo provedor da interface (comandos) e serviços a serem implementados pelo usuário da interface (eventos).
- Separação de construção e composição. Os componentes definem dois escopos, um para a especificação (que contém a interface e composição) e outro para implementação.
- Componentes são ligados estaticamente uns aos outros através de suas interfaces.
- Concorrência é baseada em tarefas não-preemptivas entre si, que podem ser interrompidas por interrupções.

Uma aplicação ou *configuração* do TinyOS consiste na especificação de um componente que conecta uma série de outros componentes. A figura 3.1 sumariza o desenvolvimento de uma aplicação para o TinyOS [Tin06]. Essa aplicação pisca periodicamente os LEDs da plataforma de sensores. A configuração liga o módulo `Main` (primeiro módulo a ser executado no TinyOS) aos módulos `Blink` e `SingleTimer` (não ilustrados na figura). A configuração liga também o módulo `BlinkM` (módulo que contém a funcionalidade da aplicação) ao `SingleTimer` e ao componente `Leds`.

A especificação do módulo `BlinkM` mostra que esse fornece a interface `StdControl` (implementada por todos os módulos no TinyOS), e utiliza as interfaces `Timer` e `Leds` (não ilustradas na figura). O comportamento da aplicação é especificado em uma sintaxe similar a da linguagem de programação `C`. A figura 3.2 apresenta o grafo de componentes para essa aplicação. As aplicações nesC não são compiladas diretamente para código objeto, mas primeiro traduzidas para `ANSI-C`, e depois compiladas com ferramentas `GNU` tradicionais.

3.1.1 Funcionalidade Básica

O modelo simplificado de concorrência do TinyOS, baseado em tarefas que executam até sua completude, que só podem ser preemptadas por interrupções, traz implicações positivas e ne-

```
configuration Blink {
}
implementation {
  components Main, BlinkM, SingleTimer, LedsC;

  Main.StdControl -> BlinkM.StdControl;
  Main.StdControl -> SingleTimer.StdControl;
  BlinkM.Timer -> SingleTimer.Timer;
  BlinkM.Leds -> LedsC;
}
```

(a) Configuração

```
module BlinkM {
  provides {
    interface StdControl;
  }
  uses {
    interface Timer;
    interface Leds;
  }
}

implementation {

  command result_t StdControl.init() {
    call Leds.init();
    return SUCCESS;
  }
  command result_t StdControl.start() {
    return call Timer.start(TIMER_REPEAT, 1000) ;
  }
  command result_t StdControl.stop() {
    return call Timer.stop();
  }
  event result_t Timer.fired() {
    call Leds.redToggle();
    return SUCCESS;
  }

}
```

(b) Módulo `BlinkM` utilizado pela configuração

```
interface StdControl {
  command result_t init();
  command result_t start();
  command result_t stop();
}
```

(c) Interface `StdControl` fornecida pelo módulo

Figura 3.1: Aplicação exemplo do TinyOS

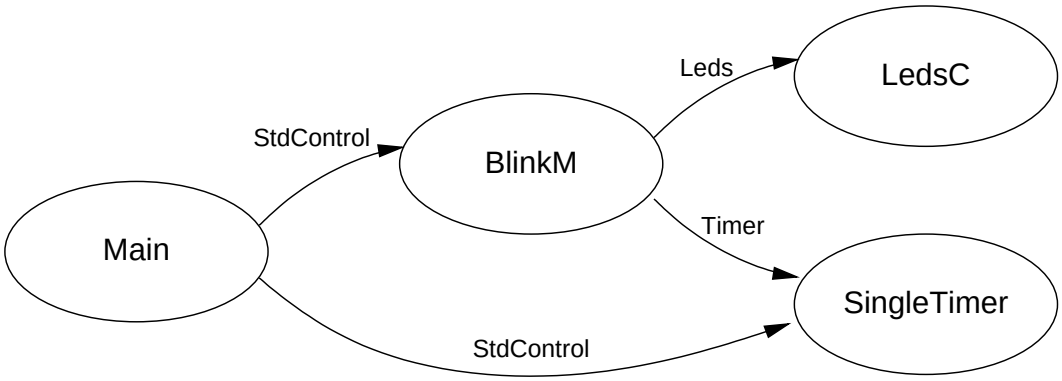


Figura 3.2: Grafo de componentes para a aplicação `Blink`

gativas. Um modelo tradicional, baseado em *threads* com pilhas próprias exige que cada *thread* reserve espaço para seu contexto de execução na memória. Isto pode ser crítico em sistemas com muito pouca memória, como é o caso dos nodos de redes de sensores. Dependendo da arquitetura e número de registradores do processador, a troca de contexto pode ser bastante demorada. Restringindo este modelo ao máximo, não permitindo concorrência entre tarefas, o TinyOS reduz boa parte deste sobrecusto, mas perde também boa parte das características de um modelo *multithread* tradicional. A restrição de concorrência no TinyOS também limita a capacidade do sistema sistema em tratar de métricas de tempo real.

O TinyOS não provê nenhum mecanismo para alocação dinâmica de memória. Aplicações que necessitem deste serviço precisam implementar um banco de recursos (estático) e mecanismo de gerência próprio. O sistema provê serviços de temporização (alarmes e contagem de tempo) através da interface `Timer`.

3.1.2 Controle do Consumo de Energia

A gerência do consumo de energia no TinyOS é implementada pelo escalonador de tarefas, que faz uso da interface `StdControl`. A interface provê métodos para início e parada de componentes. Quando a fila de tarefas do escalonador está vazia, o escalonador pára o processador, deixando os periféricos operando. Desta forma, novas tarefas só serão postadas no evento de uma interrupção. Este controle simples fornece resultados bastante satisfatórios para a gerência de consumo do microcontrolador principal, e mecanismos de gerência de energia mais agressivos são deixados a cargo da aplicação.

3.1.3 Reprogramação

O TinyOS incorpora duas soluções para reprogramação em campo: o protocolo Deluge [HC04] (para reprogramação total) e a máquina virtual Maté [LC02].

Deluge é um protocolo epidêmico de disseminação de objetos de dados de tamanho grande para nodos de sensor. Nesse protocolo, cada nodo anuncia periodicamente para todos os outros nodos a versão mais recente do objeto de dados que possui. Se um nodo S recebe um anúncio de um nodo mais velho R, S responde com um novo anúncio. Através do anúncio, R determina quais partes do objeto são necessários para sua atualização, e solicita os dados aos nodos anunciantes. Para reprogramação de um nodo sensor, o protocolo precisa de suporte de sistema operacional para reprogramação total (normalmente na forma de um gestor de inicialização (*bootloader*) que reprograma o sistema a partir de uma memória externa), e de uma memória de dados no mínimo maior do que a própria memória de programa do microcontrolador.

O protocolo Deluge foi também expandido para suportar criptografia e verificação de integridade para objetos de dados [DHCC06]. Seu maior problema, entretanto, está relacionado à própria técnica de reprogramação total da memória *flash* do micro-controlador, que tem um custo muito elevado em termos de consumo de energia [LGC05].

A máquina virtual Maté contorna este problema em particular, transformando o código a ser executado no sensor em pequenas aplicações interpretadas. A aplicação é armazenada em memória RAM, e a reprogramação do nodo não passa pela programação da memória *flash* do micro-controlador. A figura 3.3 apresenta uma aplicação exemplo da máquina virtual Maté. Essa aplicação periodicamente lê o sensor de luz da plataforma. Se o valor lido do sensor difere do valor lido na última iteração por mais do que um dado valor (32 no exemplo), o programa envia os dados através da interface de comunicação. As instruções da máquina virtual são de “alto nível” (e.g. `sense` para ler um sensor, `send` para enviar dados) e, portanto, sua semântica depende da plataforma de execução. A máquina virtual Maté utiliza cerca de 16KB de memória flash e 850 bytes de memória RAM. O custo de execução das instruções quando comparado à execução de código nativo equivalente varia de 33:1 (instrução `and`) até 1.03:1 (instrução `send`) [LC02]. As principals limitações da máquina virtual Maté estão, portanto, no sobrecusto introduzido no sistema pelo interpretador, na dependência das instruções com a plataforma de execução, e na falta de suporte a linguagens de programação de alto nível.

```
pushc 1     # Push one on the operand stack
sense       # Read sensor 1 (light)
copy        # Copy the sensor reading
gets        # Get previous sent reading
inv         # Invert previous reading
add         # Current - previous sent value
pushc 32
add
blez 17     # If curr < (prev-32) jump to send
copy        # Copy the sensor reading
inv         # Invert the current
gets        # Get the previous reading
add         # Previous - current
pushc 32
add
blez 17     # If (curr+32) > prev jump to send
halt
copy        # PC 17 -- jump-to point from above
sets        # Set shared var to current reading
pushm       # Push a message onto operand stack
clear       # Clear out the message payload
add         # Add the reading to message payload
send        # Send the message
halt
```

Figura 3.3: Aplicação exemplo Maté

3.1.4 Abstração de Hardware

O TinyOS apresenta um projeto de três camadas para abstração de hardware [HPH+05]. Uma *Camada de Apresentação de Hardware* fica diretamente sobre a interface software/hardware. Componentes nesta camada exportam uma interface que é completamente determinada pelas características do hardware. Uma *Camada de Adaptação de Hardware* usa estas interfaces para construir componentes específicos para um domínio, como `Alarm` e `ADC Channel`. Todos os componentes nessa camada são implementados para dispositivos específicos de hardware e possuem características específicas. Finalmente, uma *Camada de Interface de Hardware* usa os componentes específicos para plataformas e os converte em interfaces independentes de hardware através de adaptação por software (emulando ou ignorando características de hardware). Não há abstrações independentes de plataforma específicas para dispositivos sensores além da interface de canal de ADC. Isto faz com que as aplicações tenham que completar a funcionalidade dos *drivers* de sensores, comprometendo a portabilidade das mesmas.

3.1.5 Canal de Comunicação

O TinyOS tem utilizado diferentes soluções para comunicação de dados desde sua implementação inicial até suas últimas versões. As pilhas de comunicação originais eram construções monolíticas, dependentes de plataforma (e.g. rádio TR1000 ou CC1000). Além do MAC, essas soluções incorporavam também uma implementação de mecanismos de comunicação single-hop e multi-hop através de mensagens ativas [LMG+04]. O protocolo S-MAC (descrito na seção 2.3.2) segue também esta alternativa, sendo uma solução monolítica e pouco configurável, otimizada para determinadas cargas de trabalho.

A implementação do protocolo B-MAC (descrito na seção 2.3.1) foi projetada em camadas (controle de hardware de baixo nível e lógica do protocolo), e não incorpora implementação de serviços de comunicação de alto nível. O controle de hardware de baixo nível permite configuração dinâmica e estática dos parâmetros de comunicação (e.g. frequência, potência de transmissão). A configuração dinâmica implica na inclusão de algoritmos para cálculos de valores de registradores de hardware. A configuração estática depende de parâmetros específicos de compilação ou interferência direta do usuário no sistema. Da mesma forma, o sistema permite configuração da lógica do protocolo (tamanho do período ativo, uso do algoritmo CCA e uso de pacotes de confirmação).

É possível trocar uma pilha de comunicação por outra (e.g. SMAC por BMAC), mas, dadas as diferenças entre serviços e interfaces das diferentes soluções, aplicações projetadas com uma solução dificilmente serão diretamente portáveis para outra. Além disto, este tipo de configuração exige interferência do usuário na estrutura do sistema.

3.1.6 Uso de Recursos Pelo Sistema

O modelo de componentes e concorrência do TinyOS torna o sistema fiel ao seu nome (*tiny* - minúsculo em português) tanto no que diz respeito aos recursos consumidos quanto à sua funcionalidade. O sistema em si consume poucos recursos, e é capaz de executar em plataformas de 8 bits com menos de 1KB de RAM. Entretanto, o modelo de concorrência simplificado, a falta de gerência dinâmica de recursos e modelo de abstração de hardware excessivamente dependente de plataforma fazem com que as aplicações muitas vezes tenham que completar a funcionalidade do sistema operacional. Isto é evidentemente indesejável, já que compromete a portabilidade das aplicações e a reusabilidade de código, além de agregar tamanho e consumo de recursos ao sistema.

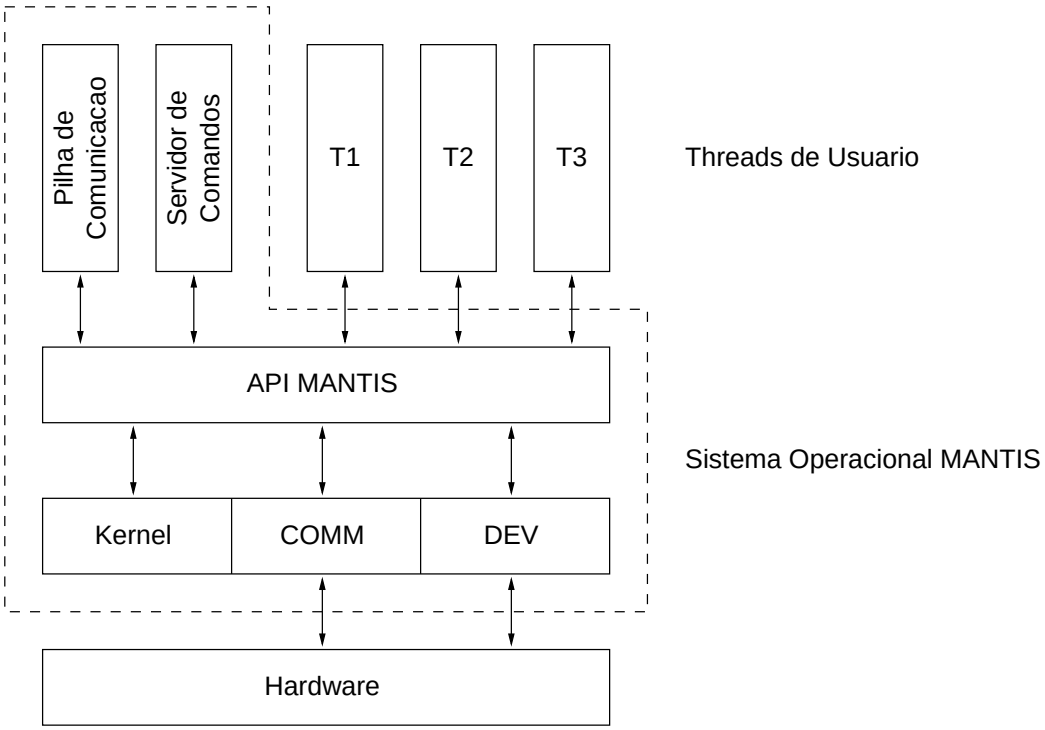


Figura 3.4: Projeto do MANTIS OS

3.2 MANTIS OS

O sistema operacional MANTIS (*Multimodal Networks of In-situ Sensors*) [ABC+03] é um sistema *multithread* de código aberto. O sistema foi criado no contexto do projeto MANTIS, da Universidade do Colorado [MAN06]. MANTIS OS é desenvolvido em linguagem `C` e tem uma API inspirada em POSIX, adaptada às necessidades das redes de sensores sem fios. O sistema possui portes para as plataformas Mica e Telos (parcial [MAN06]).

A arquitetura do sistema MANTIS é baseada no projeto de sistemas *multithread* em camadas clássico (ver Figura 3.4). O projeto do sistema prevê uma Interface para Programação de Aplicações (API – *Application Programming Interface*) preservada entre plataformas. O núcleo do sistema é composto por um escalonador e *drivers* de baixo nível para comunicação e leitura de dispositivos (e.g. ADC). O sistema fornece ainda uma pilha de comunicação em nível de usuário e um servidor de comandos. A figura 3.5 mostra uma aplicação *lê sensor e envia* desenvolvida como exemplo padrão do sistema [MAN06]. Essa aplicação, que executa no *mote* Mica2, lê o valor de um canal do ADC, pisca os LEDs da plataforma, e transmite o valor lido pelo rádio.

```
#include <inttypes.h>
#include "led.h"
#include "dev.h"
#include "com.h"

void start(void)
{
  uint16_t delay;
  uint8_t value;
  comBuf send_pkt;

  value = dev_get(DEV_MICA2_LIGHT);
  mos_led_toggle(0);
  send_pkt.data[0] = value;
  send_pkt.size=1;
  com_send(IFACE_RADIO, &send_pkt);
}
```

Figura 3.5: Aplicação exemplo do MANTIS OS

3.2.1 Funcionalidade Básica

O escalonador do MANTIS OS fornece um subconjunto do pacotes de *threads* POSIX, com escalonamento *round-robin* com prioridades. O sistema suporta tanto alocação dinâmica quanto estática de *heaps* para as *threads*. A principal estrutura do *kernel* é uma tabela de *threads*, com tamanho especificado em tempo de compilação. Cada entrada na tabela contém um ponteiro para a pilha da *thread* (*stack pointer*), limite inferior e superior da pilha, ponteiro para a função de início, prioridade, e um ponteiro para a próxima *thread* na fila do escalonador [BCD+05]. O escalonador é acionado periodicamente por um temporizador dedicado, ou por operações em semáforos. Uma *thread idle* é criada na inicialização do sistema, e executa quando todas as outras *threads* estão bloqueadas, funcionando como ponto de entrada para políticas de gerência do consumo de energia. Serviços de sincronização e temporização são fornecidos em interfaces similares a POSIX.

3.2.2 Controle do Consumo de Energia

O sistema de controle de consumo de energia do MANTIS OS implementa um método similar à função UNIX `sleep` [BCD+05]. Para utilizá-la, uma aplicação deve habilitar a gerência de energia, através da função `mos_enable_power_mgt()`. A seguir, a aplicação pode utilizar a função `mos_thread_sleep(P)` para *dormir* por um período `P`. A chamada dessa função acar-

reta na execução do escalonador do sistema que, se não houver mais *threads* a serem executadas, coloca o microcontrolador em um estado de *sleep*. A função `sleep` controla apenas o microcontrolador principal do sistema, e não há controle específico para os periféricos além do fornecido pelos *drivers*.

3.2.3 Reprogramação

O problema de reprogramação remota não é endereçado diretamente pelo MANTIS OS, mas faz parte do plano de desenvolvimento do sistema [BCD+05].

3.2.4 Abstração de Hardware

O MANTIS OS utiliza uma camada de abstração de hardware monolítica tradicional, com funções `dev_read()`, `dev_write()`, `dev_mode()` e `and dev_ioctl()` em estilo POSIX. Cada função passa como parâmetro o dispositivo a ser tratado, e uma tabela de ponteiros para funções redireciona, em tempo de execução, as chamadas gerais às chamadas específicas. Os parâmetros de `dev_mode()` e `and dev_ioctl()` são específicos para cada dispositivo. Como no caso da pilha de comunicação do MANTIS OS, este sobrecusto é indesejável e não traz vantagens contundentes (ver seção 3.2.5). Não há no MANTIS OS abstrações unificadas para sensoriamento (cada *driver* de sensor tem semântica específica).

3.2.5 Canal de Comunicação

O MANTIS OS fornece uma interface unificada de comunicação na forma de uma ou mais *threads* de usuário. Há um formato de pacote unificado para as diversas interfaces de comunicação (e.g. interfaces seriais, USB ou rádio). Essa camada de comunicação gerencia sincronização e buferização de pacotes. As aplicações interagem com todas as interfaces de comunicação através de quatro funções: `com_send`, `com_recv`, `com_mode` e `ioctl`. Abaixo desta API de comunicação, o MANTIS OS utiliza *drivers* de dispositivo tradicionais para tratar de suporte de hardware e implementar suporte de MAC.

Uma *thread* chama `com_send` passando um ponteiro para uma estrutura `comBuf`. A *thread* fica bloqueada até que o envio esteja completo. A recepção de pacotes acontece assincronamente, e pacotes são armazenados em *buffers* da pilha de comunicação até que sejam solicitados pela aplicação com um `com_recv`. As chamadas `com_mode` e `ioctl` são dependentes de dispositivo,

e podem ser utilizadas para controlar parâmetros de comunicação. O sistema implementa protocolos MAC próprios baseados no S-MAC e T-MAC, e inclui variações do B-MAC (seção 2.3.1) com extensões para *frequence hopping* [BYAH06, BCD+05].

A abordagem monolítica da pilha de comunicações do MANTIS OS pode ocasionar um sobrecusto grande de desempenho e tamanho de código no sistema. Uma API unificada com, por exemplo, um único método `com_send` para todos os dispositivos no sistema implica em testes na execução do método (para que o sistema descubra através de qual interface o método deve enviar), e na inclusão de código de envio para todos os dispositivos possíveis (mesmo que a aplicação não os utilize). Por outro lado, a vantagem aparente de um único ponto de entrada no sistema de comunicação é diminuída pelas chamadas específicas aos métodos `com_mode` e `ioctl`. Como a semântica e parâmetros desses métodos varia de hardware para hardware, a aplicação não pode trocar transparentemente de interface de comunicação (e.g. trocar um modelo de rádio por outro, ou trocar uma interface serial por um rádio). Um aspecto positivo da pilha de comunicação do MANTIS OS é a configuração relativamente ampla do protocolo de acesso ao meio (realizada através das funções `com_mode` e `ioctl`). Técnicas de pré-processamento podem eliminar parte do sobrecusto de configuração nesses métodos (e.g. eliminando partes do protocolo inválidas em determinado contexto de execução). Entretanto, a configuração propriamente dita se dá em tempo de execução e pode ocasionar sobrecusto indesejável à aplicação.

3.2.6 Uso de Recursos pelo Sistema

O modelo bastante completo de escalonamento do MANTIS OS faz com que o sistema tenha requisitos maiores do que um modelo mais simples, baseado em eventos. Essa diferença está, entretanto, relacionada primariamente com as próprias diferenças entre os modelos, e não com eventuais falhas no projeto de um sistema em particular. O mesmo vale para os sistemas monolíticos de abstração de hardware e comunicação presentes no sistema. Este último caso, entretanto, pode ser considerado um ponto bastante desfavorável ao sistema MANTIS OS, já que o projeto do sistema incorpora os problemas associados ao projeto monolítico [FSP00, PF04, PFHD04], sem trazer grandes vantagens (já que muitas chamadas são específicas para dispositivos) e incorporando sobrecusto ao já restrito modelo computacional das redes de sensores sem fios. Comparações diretas de tamanho e sobrecusto de sistema com outros sistemas são, em geral, bastante desfavoráveis ao MANTIS OS [WHPF05a, WHPF05b]. Ainda assim, o sistema mostra-se adequado à execução nos protótipos atuais de nodos de redes de sensores.

3.3 SOS

SOS [HKS+05] é um sistema operacional para redes de sensores sem fios desenvolvido no *Networked and Embedded Systems Lab* da Universidade da Califórina em Los Angeles. O principal objetivo do sistema é tratar de reconfiguração dinâmica dos serviços em redes de sensores. Para tanto, o núcleo do SOS inclui serviços de passagem de mensagens, alocação dinâmica de memória e carga dinâmica de módulos. O sistema foi projetado para executar nos motes Mica2 e em diferentes simuladores [SOS06].

O SOS é organizado como um conjunto de *módulos binários* que implementam tarefas ou funções específicas, comparáveis em funcionalidade com componentes do TinyOS. Uma aplicação do sistema é composta por uma série de módulos que interagem entre si. Os módulos apresentam interfaces de métodos e passagem de mensagens. A passagem de mensagens é implementada utilizando uma função tratadora específica para cada módulo. Todos os módulos implementam, no mínimo, tratamento para as mensagens `init` e `final`, utilizadas pelo *kernel* do sistema na carga e finalização dos módulos. O tratador de inicialização registra o estado inicial do módulo, incluindo registro e assinatura de funções. O tratador de finalização libera todos os recursos alocados pelo módulo.

A passagem de mensagens do SOS funciona de maneira assíncrona e é coordenada por um escalonador que retira mensagens de uma fila ordenada por prioridade e passa a mensagem ao tratador adequado do módulo de destino. Chamadas diretas de funções são utilizadas quando há necessidade de operações síncronas entre módulos. A carga e distribuição de módulos são realizadas através de estruturas de meta-descrição de módulos, e protocolos de distribuição de imagens de módulos independentes do *kernel*.

A figura 3.6 mostra uma aplicação desenvolvida como exemplo padrão do sistema [SOS06]. Esta aplicação pisca periodicamente os LEDs da plataforma de sensores. Somente o tratador de mensagens é mostrado na figura, a aplicação completa inclui ainda definições de estruturas para carga e descrição do módulo. O módulo está conectado a um módulo de temporização (`TIMER`) e ao *kernel* do sistema. O tratador do módulo recebe mensagens de inicialiação (`MSG_INIT`), finalização (`MSG_INIT`), e *timeout* do temporizador (`MSG_TIMER_TIMEOUT`). Quando o tratador detecta uma mensagem do temporizador, testa os parâmetros da mensagem e verifica o estado de execução atual para determinar qual LED da plataforma deve ser ativado.

```
#include "blink.h"

// ...

static int8_t blink_msg_handler(void *state, Message *msg)
{
    app_state_t *s = (app_state_t*)state;

    switch (msg->type){
    case MSG_INIT: {
        LED_DBG(LED_GREEN_TOGGLE);
        s->pid = msg->did;
        s->state = 0;
        ker_timer_init(s->pid, BLINK_TID, TIMER_REPEAT);
        if(msg->data == NULL) {
            ker_timer_start(s->pid, BLINK_TID,
                            BLINK_TIMER_INTERVAL);
        } else {
            uint16_t period = *(uint16_t*)(msg->data);
            ker_timer_start(s->pid, BLINK_TID, period);
        }
        break;
    }
    case MSG_FINAL: {
        ker_timer_stop(s->pid, BLINK_TID);
        break;
    }
    case MSG_TIMER_TIMEOUT: {
        MsgParam* params = (MsgParam*)(msg->data);

        if (params->byte == BLINK_TID)  {
            if(s->state == 1)
                LED_DBG(LED_GREEN_OFF);
            else
                LED_DBG(LED_GREEN_ON);
            s->state++;
            if(s->state > 1) s->state = 0;
        }
        break;
    }
    default:
        return -EINVAL;
    }
    return SOS_OK;
}
```

Figura 3.6: Aplicação exemplo do SOS

3.3.1 Funcionalidade Básica

O modelo de concorrência do SOS é baseado em mensagens escalonáveis. As mensagens funcionam de maneira análoga às tarefas do TinyOS, mas o escalonador do SOS permite ordenação de mensagens de acordo com sua prioridade. O sistema integra serviços de alocação dinâmica de memória e *garbage collection* (usada na finalização de módulos). O sistema inclui módulos para serviços de temporização e periféricos.

3.3.2 Controle do Consumo de Energia

O SOS usa o mesmo modelo de controle do consumo de energia do TinyOS e MANTIS OS: quando não há mensagens a serem escalonadas, o microcontrolador é colocado em modo de baixo consumo de energia, com os periféricos ligados. Módulos podem implementar políticas próprias para seus recursos individuais utilizados.

3.3.3 Reprogramação

Como um sistema dinâmico, reconfiguração é a principal preocupação do SOS. O sistema provê serviços para inclusão, atualização e remoção de módulos aos programas de sensoriamento já instalados. Para tanto, o sistema divide a memória de programa do nodo em páginas, e guarda variáveis de estado e contexto em RAM. O custo de reprogramação de páginas específicas é bastante alto, mas pequeno em comparação com a reprogramação total. Esta economia de energia implica, entretanto, em um custo elevado em termos de uso de memória RAM, já que o sistema precisa manter uma série de estruturas de descrição e controle dos módulos carregados.

O algoritmo de disseminação de código na rede é implementado como um módulo independente do *kernel* e, em princípio, o sistema poderia utilizar qualquer esquema de propagação disponível (e.g. Deluge [HC04], MOAP [SHE03]).

3.3.4 Abstração de Hardware

Para a abstração de hardware de sensoriamento, o sistema utiliza os módulos carregáveis do *kernel*. Através destes, um sensor analógico *conecta-se* a um canal de ADC e registra um tipo de sensor (e.g. `PHOTO`). Quando a aplicação requisita dados de um tipo de sensor, o kernel envia o pedido para o *driver* registrado e recebe a leitura de ADC apropriada. Essa solução aproxima-se de uma abstração consistente, já que modelos diferentes de hardware com a mesma função podem

ser abstraídos de forma similar, ainda que com grande sobrecusto. O registro de *drivers* ocasiona sobrecusto de memória, já que o sistema operacional tem que armazenar uma tabela de ponteiros de função indexada por tipo de sensor. A troca de mensagens entre módulos ocasiona sobrecusto de execução, já que o módulo deve testar o tipo da mensagem recebida e determinar a ação apropriada em tempo de execução.

3.3.5 Canal de Comunicação

O SOS utiliza, em sua versão atual, o subsistema de comunicação do TinyOS, encapsulado na forma de módulos próprios do sistema.

3.3.6 Uso de Recursos pelo Sistema

O modelo de reconfigurabilidade dinâmica do SOS faz com que o sistema tenha requisitos de memória e sobrecusto consideravelmente maior do que os outros sistemas analisados. Entretando, os autores do sistema afirmam que este sobrecusto é aceitável para a maioria de aplicações de redes de sensores sem fios [HKS+05]. O sistema é capaz de executar aplicações relativamente complexas na plataforma Mica2 [SOS06].

3.4 Outros Sistemas

O sistema AmbientRT [HDJH04] é um protótipo de sistema operacional de tempo real para redes de sensores sem fios. O sistema implementa um escalonador baseado em EDF (*Earliest Deadline First*), com herança de *deadline* para recursos compartilhados. O protótipo do *kernel* utiliza aproximadamente 3800 bytes de memória flash no microcontrolador MSP430.

O sistema YaTOS [dAVV+04], desenvolvido no projeto SensorNET da UFMG, é um sistema baseado em tarefas não-preemptivas para o microcontrolador MSP430.

O sistema MagnetOS [BBD+02] é um sistema distribuído que utiliza mecanismos de invocação remota de métodos (RMI – *Remote Method Invocation*) em Java para permitir particionamento de aplicações em componentes que podem ser distruibuídos pela rede. O sistema funciona em processadores IA32 e StrongARM.

O sistema Contiki [DGV04] é um sistema baseado em eventos projetado para permitir carga dinâmica de serviços e aplicações em nodos com recursos limitados. O sistema inclui bibliotecas para para programação *multi-thread*, que são implementadas sobre o *kernel* baseado em eventos.

3.5 Comparação

Os três sistemas analisados em detalhe neste capítulo foram desenvolvidos com objetivos diferentes: sobrecusto mínimo (TinyOS), similaridade a sistemas tradicionais POSIX (MANTIS OS), e reconfigurabilidade dinâmica (SOS). Como tal, cada sistema tem pontos fortes e fracos no cumprimento dos requisitos levantados na introdução deste capítulo.

Quanto ao modelo de escalonamento, os sistemas analisados podem ser classificados como baseado em tarefas não preemptivas (TinyOS e SOS) e baseado em *threads* (MANTIS OS). As diferenças entre esses modelos de escalonamento são brevemente exploradas nas seções anteriores, e amplamente comparadas na literatura [vBCB03, LN79, Ous96]. Como um sistema estritamente estático, o TinyOS não fornece serviços de gerência dinâmica de recursos (e.g. memória). Estes serviços são oferecidos tanto pelo MANTIS OS quanto pelo SOS.

O modelo de gerência de consumo de energia é virtualmente idêntico nos três sistemas: quando não há mais tarefas, *threads* ou mensagens a serem escalonadas, o microcontrolador é colocado em estado de baixo consumo de energia. Em geral, módulos individuais podem fornecer serviços adicionais de gerência de energia, mas essas políticas não são ditadas pelo sistema operacional. As vantagens, desvantagens e limitações deste modelo são exploradas em trabalhos relacionados a esta dissertação [HWF06b, HWdO$^+$06, HWF06a] e na literatura [VF05].

O modelo de abstração de hardware, especialmente no que diz respeito ao hardware de sensoriamento, é incompleto nos três sistemas revisados. Nenhum dos sistemas provê interfaces padronizadas para obtenção de dados de sensores. Os modelos do TinyOS e do MANTIS OS expõem detalhes do hardware à aplicação, comprometendo portabilidade por dar nome e semântica específica a modelos de hardware dentro de uma classe que poderia ser abstraida uniformemente. O modelo de abstração de hardware monolítica do sistema MANTIS OS é visivelmente um ponto desfavorável ao sistema, já que implica em sobrecusto de memória e execução, sem trazer vantagens contundentes. O modelo do SOS é o que mais se aproxima de uma abstração ideal, mas implica em alto sobrecusto.

O MANTIS OS não desenvolveu um modelo específico de reprogramação. O modelo do SOS é claramente superior ao modelo presente no TinyOS. No TinyOS, o modelo de máquina virtual é comprometido pelo sobrecusto e funcionalidade simplificada, e o modelo de reprogramação total é comprometido pelo alto custo de consumo de energia. O modelo do SOS é praticável em custo de energia, e o sobrecusto de execução, ainda que alto, é visivelmente menor do que o de uma máquina virtual.

Configuração do canal de comunicação é uma preocupação secundária no SOS. O sistema utiliza as pilhas de comunicação como subsistemas fechados, com pouca chance de adaptação às aplicações. No TinyOS e no MANTIS OS, a pilha de comunicação baseada no protocolo S-MAC apresenta esta mesma visão, mas a baseada no B-MAC possui uma série de parâmetros configuráveis, tanto em tempo de execução quanto em tempo de compilação. Nenhum sistema apresenta um esquema de configuração transparente, exigindo interação direta da aplicação ou ajustes de parâmetros de compilação. A combinação entre protocolos é bastante limitada, ou inexistente nos três sistemas.

Quanto ao sobrecusto e uso de recursos de memória, há uma clara ordem entre os três sistemas: TinyOS, MANTIS OS e SOS (do menor uso de recursos para o maior). Essa ordem está diretamente relacionada aos diferentes objetivos e modelos de implementação dos três sistemas. Conforme citado nas seções anteriores, o tamanho pequeno de um sistema pode também refletir uma pobreza de serviços, fazendo com que a aplicação tenha que completar a funcionalidade do sistema operacional. As seções 4.1.4 e 4.2.2 apresentam análises quantitativas dos sistemas, em comparação com as soluções apresentadas nesta dissertação.

3.6 O Sistema EPOS

O sistema EPOS (Embedded Parallel Operating System) [Frö01, PFHD04, PF04] é um *framework* baseado em componentes para geração de sistemas de suporte de execução a aplicações de computação dedicada. O projeto do sistema, baseado na metodologia de Projeto de Sistemas Orientado à Aplicação (AOSD – *Application Oriented System Design*) [Frö01], permite que programadores desenvolvam aplicações independentes de plataforma. Ferramentas de análise de aplicações permitem a geração de um sistema de suporte de tempo de execução que agregue todos os recursos que esta aplicação específica necessita, e nada mais. Por definição, uma instância do sistema utiliza somente os recursos necessários ao suporte da aplicação. Ao mesmo tempo, o repositório de componentes do sistema disponibiliza um grande conjunto de serviços tradicionais de sistema operacional através de interfaces independentes de plataforma. O sistema suporta diversas plataformas computacionais heterogêneas, como IA32, PowerPC, H8, Sparc e MIPS, entre outras [MHWF06, MHW$^+$06]. No contexto deste trabalho, o EPOS ganhou também portes para plataformas de 8-bits AVR, tradicionalmente utilizadas em sistemas embarcados de baixa potência como redes de sensores sem fios. Trabalhos relacionados a esta dissertação adicionaram mecanismos para gerência do consumo de energia aos componentes do

sistema [HWF06b, HWdO+06, HWF06a], tornando o EPOS um bom candidato à implementação de serviços de sistema operacional para redes de sensores sem fios.

Esta seção apresenta os principais conceitos relacionados ao EPOS. A seção 3.6.1 introduz o projeto de sistemas orientado à aplicação. A seção 3.6.2 apresenta os principais aspectos do projeto e funcionalidade do sistema. A seção 3.6.3 apresenta aspectos relevantes do porte do sistema para a arquitetura AVR. Finalmente, a seção 3.6.4 analisa a viabilidade do uso do sistema em redes de sensores sem fios.

3.6.1 Projeto de Sistemas Orientado à Aplicação

Projeto de Sistemas Orientado à Aplicação (AOSD) [Frö01] é uma metodologia de engenharia de domínio que expande as estratégias de análise de *características comuns* e *variabilidades* do Projeto Baseado em Famílias (FBD – *Family-Based Design*) [Par76, CHW98] e Orientação a Objetos (OO) [Boo94], adicionando os conceitos de identificação e separação de aspectos [KLM+97]. Desta maneira, AOSD guia o processo de engenharia de domínio na direção de *famílias de componentes*, nas quais dependências do cenário de execução são fatoradas como *aspectos*, e relações externas são capturadas em um framework de componentes. Esta estratégia de engenharia de domínio trata consistentemente alguns dos problemas mais relevantes no desenvolvimento de software baseado em componentes:

1. Reusabilidade: os componentes tendem a ser altamente reusáveis, já que são modelados como abstrações de elementos reais de um dado domínio, e não como partes de um sistema alvo. Além disso, como as dependências do cenário de execução são fatoradas como aspectos, os componentes podem ser reutilizados sem modificação em uma série de cenários, simplesmente com a definição de novos programas de aspecto.

2. Gerência de complexidade: a identificação e separação de dependências do cenário de execução implicitamente reduz o número de componentes em cada família, já que os componentes que seriam modelados para expressar uma variação no cenário de execução são suprimidos quando esta dependência pode ser modelada como um aspecto. Desta forma, um conjunto de 100 componentes poderia ser modelado como um conjunto de 10 componentes mais um conjunto de 10 aspectos. A complexidade e funcionalidade total deste novo conjunto de 100 componentes é a mesma, mas está confinada em menos elementos de software. Isto melhora diretamente a manutenção do sistema.

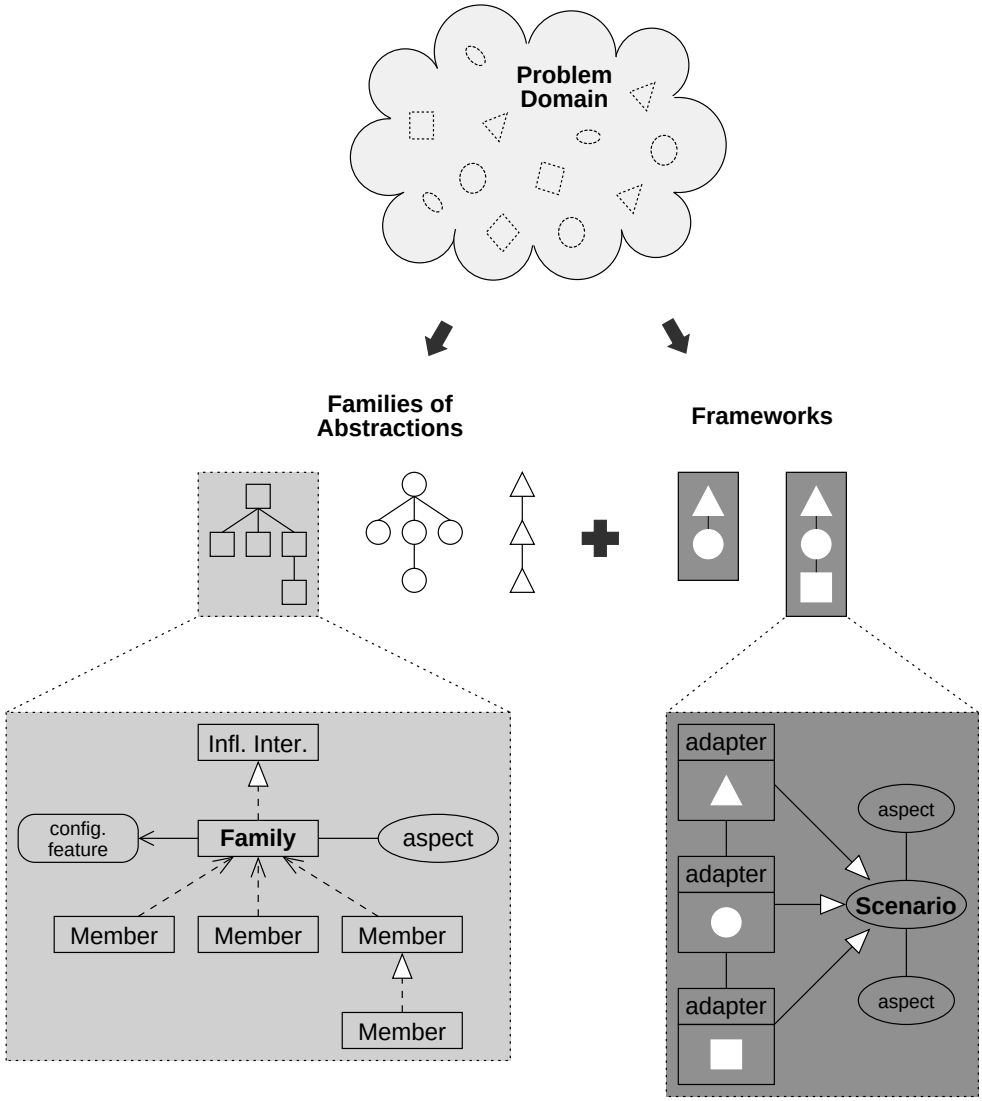


Figura 3.7: Visão geral do Projeto de Sistemas Orientado à Aplicação

3. Composição: capturando as relações entre componentes em um *framework*, AOSD permite que os componentes sejam combinados na geração do sistema. O *framework* também limita potenciais problemas na aplicação de aspectos a componentes pré-validados.

A figura 3.7 apresenta o processo de decomposição de domínio guiado por AOSD. *Abstrações* são identificadas a partir do domínio e agrupadas em famílias de acordo com suas características comuns. Dependências de cenário são modeladas como *aspectos* que podem ser aplicados através de *adaptadores de cenário*. *Famílias* de abstrações são visíveis para as aplicações através de *interfaces infladas*, que exportam seus membros como um único “super componente”. Arquiteturas de sistema são capturadas em *frameworks de componentes* definidos em termos de aspectos de cenário.

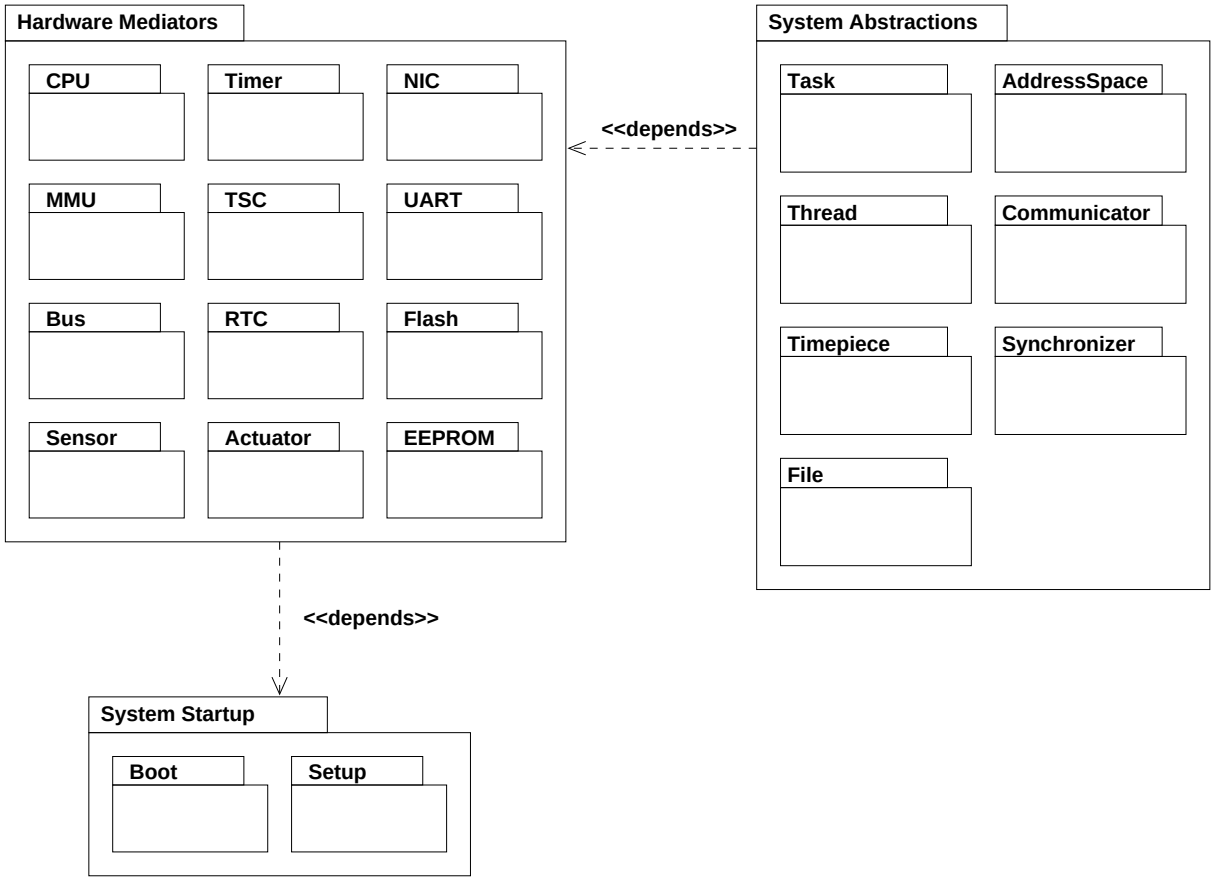


Figura 3.8: Modelo de Domínio das Abstrações do EPOS

3.6.2 Organização e Projeto do Sistema

As famílias de abstração do EPOS representam abstrações tradicionais de sistema operacional, e implementam serviços como gerência de memória e processos, coordenação entre processos, temporização e comunicação (Figura 3.8). As abstrações são modeladas e implementadas independentemente de cenários de execução de arquiteturas de sistema específicas. Todas as unidades de hardware dependentes de arquitetura são abstraídas como *mediadores de hardware* [PFHD04, PF04], que exportam, através de suas interfaces independentes de plataforma, as funcionalidades exigidas pelas abstrações. Devido ao uso de técnicas de metaprogramação e *inlining* de funções, os mediadores de hardware implementam sua funcionalidade sem formar uma camada de abstração de hardware convencional. Conseqüentemente, as abstrações do EPOS atingiram um grau de reusabilidade que permite, por exemplo, a mesma abstração da família `Thread` ser utilizada em um ambiente monotarefa ou multitarefas, como parte de um μ*kernel* ou completamente embutida em uma aplicação, em um microcontrolador de 8-bits ou um processador 64-bits.

No EPOS, cada família de abstrações é composta por um conjunto de componentes de sis-

tema operacional com funcionalidades similares entre si. As diferenças entre componentes em uma mesma família são exploradas através das hierarquias de classe. Uma *interface inflada* exporta a família como se fosse um "super" componente que implementa todas as responsabilidades atribuídas à família. Este componente é derivado das interfaces individuais dos membros da família, e realizado através de suas implementações. Por exemplo, a abstração de sistema `Thread` é composta por três componentes: `Exclusive`, `Cooperative` and `Concurrent`. A família de componentes é modelada de maneira incremental, onde o comportamento de membros mais simples é herdado por membros mais complexos. Ferramentas do sistema permitem seleção semi-automática de realizações de interface, levando em conta o hardware alvo, um modelo de custo para os componentes, e decisões explícitas do projetista da aplicação.

A portabilidade das abstrações do sistema é garantida pelo uso de *mediadores de hardware*. Estes componentes são organizados, da mesma maneira que as abstrações de sistema, em famílias cujos membros representam entidades dentro de um domínio específico (e.g. CPUs, Timers, *Network Interface Cards*). Os mediadores de hardware não representam, entretanto, uma camada de abstração de hardware (HAL – *Hardware Abstraction Layer*) universal, que encapsula todas os recursos disponíveis em uma plataforma. No EPOS, cada componente de hardware é tratado por seu próprio mediador, que mantém o *contrato de interface* entre as abstrações e o hardware. Devido ao uso de técnicas de metaprogramação estática e *inlining* de funções, quando este contrato de interface é atendido, os mediadores são *dissolvidos* no próprio código da abstração. Isto permite que as abstrações utilizem a funcionalidade do hardware de maneira transparente através dos mediadores sem o sobrecusto tradicionalmente associado às camadas de abstração de hardware [PF04]. O restante desta seção apresenta alguns detalhes do projeto e implementação das principais famílias de abstração do EPOS. Uma descrição mais detalhada destas famílias é dada por Marcondes et al [MHWF06].

3.6.2.1 Gerência de Processos

Processos são gerenciados pelas abstrações `Thread` e `Task`. As principais dependências destas abstrações para com o hardware estão relacionadas com a arquitetura alvo de execução. Esta define tanto o contexto de execução quanto o modelo de manipulação da pilha. Estas dependências arquiteturais são tratadas pelo mediador de `CPU` (figura 3.9). A classe `Context` deste mediador define todos os dados a serem armazenados por um fluxo de execução em determinada arquitetura. Cada `Thread` mantém em sua pilha o seu contexto de execução. O escalonador do sistema utiliza

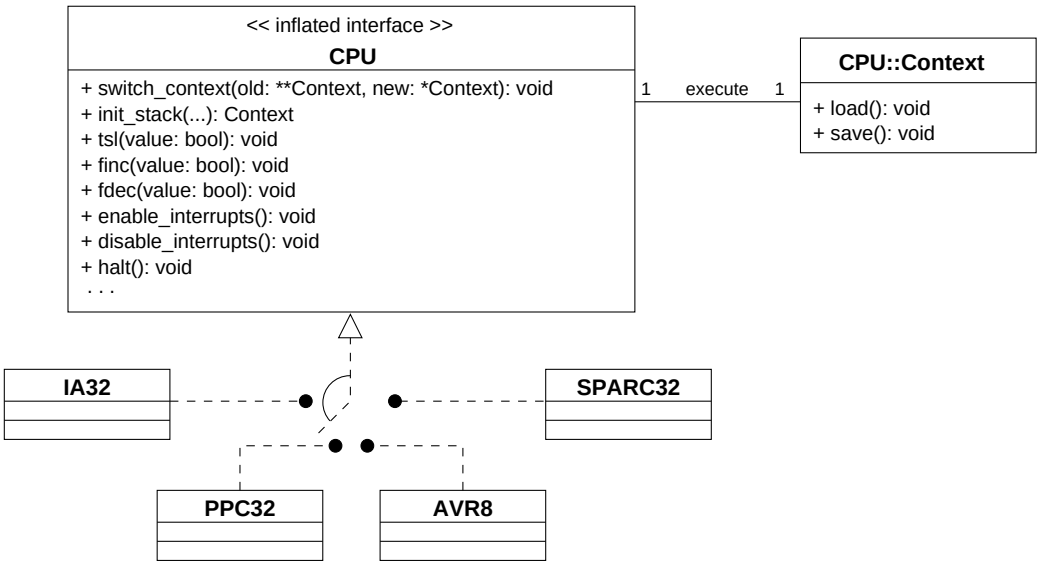


Figura 3.9: Família de Mediadores `CPU`

o método `CPU::switch_context` para trocar entre fluxos de execução. O mediador de `CPU` gerencia ainda inicialização da pilha de cada `Thread` (para permitir a passagem de parâmetros à função que executa o comportamento das mesmas), e define operações de I/O (*input/output*) da arquitetura, operações *test and set lock* e operações de troca de ordem de bytes em palavras.

3.6.2.2 Temporização

Temporização é tratada pela família de abstrações `Timepiece`. Estas abstrações são suportadas pelas famílias de mediadores `Timer`, `TSC` (Timestamp Counter) e `RTC` (Real-Time Clock). A abstração `Clock` mantém o tempo atual em sistemas que possuem um relógio de tempo real. A abstração `Alarm` pode ser usada para gerar eventos que *acordam* uma `Thread` ou chamam uma função previamente registrada. A abstração `Alarm` tem ainda um evento de alta prioridade associado a uma periodicidade de tempo configurável. Este *evento mestre* é utilizado para disparar o algoritmo de escalonamento de processos, quando o sistema está configurado com um escalonador ativo. Finalmente, a abstração `Chronometer` é utilizada para medições relativas de tempo. A família de mediadores `Timer` simplifica as características avançadas presentes em diferentes modelos de hardware de temporização, e trata o temporizador como um gerador de interrupções periódicas. Arquiteturas que não possuem um *timestamp counter* implementado em *hardware*, em geral, utilizam um temporizador adicional para simular o comportamento desejado. Esta solução tipicamente apresenta baixa precisão, mas não invalida o uso da abstração `Chronometer`.

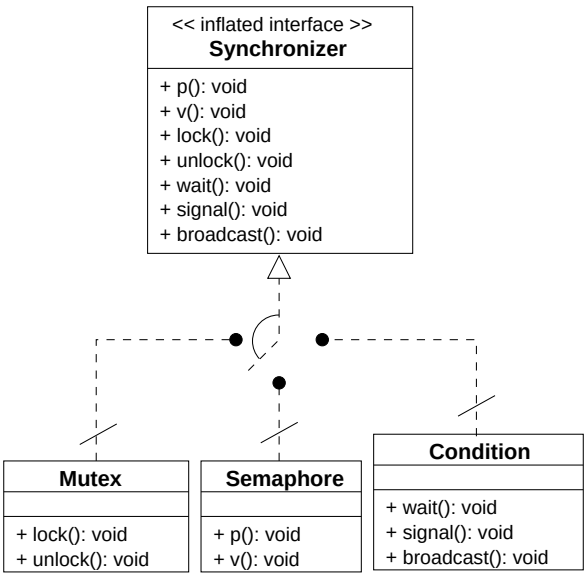


Figura 3.10: Família de abstrações `Synchronizer`

3.6.2.3 Sincronização

A família de abstrações `Synchronizer` (figura 3.10) fornece mecanismos para garantir consistência de dados em ambientes de execução concorrentes. O membro `Mutex` implementa um mecanismo de exclusão mútua que fornece duas operações atômicas: `lock` e `unlock`. O membro `Semaphore` implementa uma variável de semáforo, cujo valor pode ser manipulado indiretamente pelas operações `p` e `v`. O membro `Condition` implementa uma abstração de sistema inspirada no mecanismo de variáveis de condição, e permite que uma `Thread` suspenda sua execução até que determinado predicado em dados compartilhados torne-se verdadeiro. Estes mecanismos exigem suporte de hardware para acesso atômico a dados. Algumas arquiteturas fornecem instruções específicas para estas operações (e.g. a instruçao `xchg` do IA32), mas arquiteturas que não fornecem suporte direto em hardware podem emular esta funcionalidade desabilitanto a ocorrência de interrupções através do mediador de `CPU`.

3.6.2.4 Gerência de Memória

A abstração adequada de componentes de gerência de memória é um fator fundamental para permitir portabilidade de aplicações entre plataformas de hardware amplamente distintas (e.g. arquiteturas com hardware dedicado para gerência de memória e microcontroladores com arquitetura Harvard). No EPOS, a família de abstrações `Address_Space` é um container para segmentos de memória física. Esta abstração não trata de proteção, alocação ou tradução de endereços, dei-

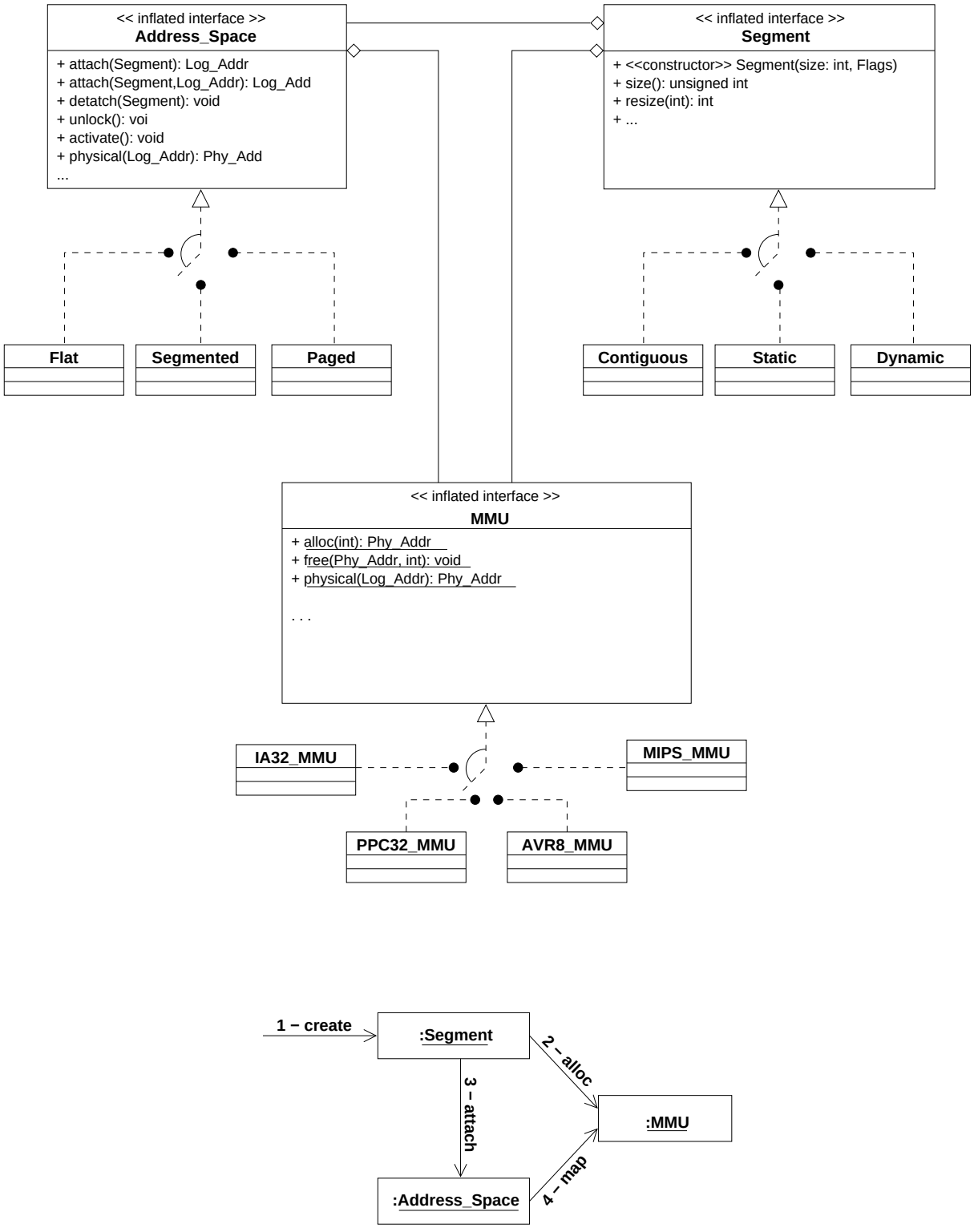


Figura 3.11: Componentes para Gerência de Memória

xando esta função para a família de mediadores `MMU` (figura 3.11). O membro `Flat` da família `Address_Space` define um modelo de memória com equivalência de endereços físicos e lógicos, eliminando a necessidade de uma unidade de gerência de memória. Esta abstração mantém o contrato de interface com outros componentes do sistema para plataformas sem uma MMU em hardware. Nestas plataformas, o mediador `MMU` é um componente que simplesmente mantém o contrato de interface com a abstração `Flat`, fornecendo implementações vazias de métodos sempre que necessário.

3.6.2.5 Entrada e Saída

O controle de entrada e saída em dispositivos periféricos no EPOS é fornecido pelos mediadores específicos do hardware em questão. Estes podem fazer uso de registradores mapeados em memória, ou utilizar operações específicas do mediador `CPU` para entrada e saída (e.g. `in8` e `out32`). O uso destas operações em geral se traduz em instruções reais do processador embutidas no código da aplicação, sem nenhum sobrecusto adicional. Nas arquiteturas onde o principal elemento do sistema é uma MCU (*microcontroller unit*) que integra periféricos e CPU, o mediador `Machine` guarda informações de localização de memória de periféricos. O mediador `Machine` também trata do registro dinâmico de tratadores de interrupções de maneira independente de plataforma. O mediador `IC` (Controlador de Interrupções), por sua vez, é responsável por habilitar, desabilitar e configurar interrupções individuais. Evidentemente, as interrupções disponíveis, bem como sua semântica, são diferentes em cada arquitetura. Para contornar isto, o EPOS dá nome e semântica comum e independente de plataforma às interrupções que são utilizadas pelas abstrações do sistema (e.g. interrupção de temporizador).

3.6.3 Porte para a Arquitetura AVR

O porte do sistema para a arquitetura AVR de 8-bits, realizado no contexto desta dissertação [Wan03], mostrou que o sistema é capaz de fornecer seus serviços em plataformas com menos de 8KB de memória de programa e 1KB de memória RAM. Esta seção apresenta e analisa alguns detalhes relevantes deste porte.

AVR é uma família amplamente utilizada de microcontroladores RISC de 8-bits da Atmel. Normalmente implementado na forma de MCUs (Microprocessor Control Units[1]), o AVR provê

[1]Em uma MCU, o processador, memória e I/O residem em um único circuito integrado.

bom desempenho por custo baixo e consumo baixo de energia em uma arquitetura de Harvard[2], fazendo dele uma das escolhas naturais para processamento e controle de Nodos de Sensor Sem Fios. O modelo ATmega128 [Atm04] do AVR, com 128KB de memória de dados e 4KB de memória RAM, é utilizado nos nodos de sensor Mica2/MicaZ [Cro05a], e BTnode [ETH05].

Conforme detalhado na seção 3.6.2, as famílias de abstrações do EPOS fazem uso de uma série de *mediadores de hardware* para interagir com a plataforma. Os mediadores essenciais ao funcionamento das abstrações básicas do sistema são: `CPU`, `MMU`, `TSC`, `Timer`, `IC` e `Machine`. Mediadores adicionais no AVR incluem ainda `UART`, `SPI` (usados para depuração e comunicação serial entre dispositivos), `EEPROM` (para armazenamento persistente de dados), uma implementação do mediador `NIC` para comunicação de rádio (apresentada em detalhes na seção 4.1), uma série de mediadores `Sensor` e `ADC` (apresentados na seção 4.2), e implementações de redes `CAN` [Mau06].

A empresa Atmel atualmente mantém em produção mais de 20 modelos de MCUs AVR da família ATmega [Atm06]. Cada um destes dispositivos possui, além de quantidades variáveis de memória, diferentes periféricos. Mesmo os periféricos comuns entre as plataformas em geral possuem semântica de operação diferente em cada um dos diferentes modelos. Restringir o uso do EPOS a um único modelo seria inaceitável, tanto pela natureza e propósitos do sistema, quanto pelas necessidades específicas de diferentes aplicações. Por outro lado, a manutenção de 20 versões diferentes de cada mediador seria impraticável. A solução deste problema passa pela definição de uma família de mediadores, cujas diferenças entre si são exploradas pela hierarquia de classes, e o uso de técnicas de programação (e.g. *inlining* de funções e gabaritos) para agregar o máximo da funcionalidade possível na base de hierarquia, sem comprometer desempenho. Desta forma, os mediadores específicos para um modelo em geral só precisam estender um mediador base com parâmetros específicos da sua funcionalidade.

O mediador `CPU` é uniforme entre todas os diferentes modelos (representados pelo mediador `Machine`). O `Context` da `CPU` guarda todos os 32 registradores de propósito da arquitetura mais o valor do registrador de estado. Em uma configuração do EPOS onde `Threads` estão habilitadas, cada `Thread` tem sua própria pilha de tamanho configurável onde também é armazenado o contexto. Se o sistema é configurado com um escalonador ativo (*threads* concorrentes), uma interrupção de temporizador de período configurável é utilizada como ponto de entrada para o algoritmo de escalonamento. Na versão atual do sistema[3], os algorítmos

[2]Uma Arquitetura de Harvard tem barramentos separados para memória de programa e dados.

[3]A versão do EPOS utilizada nesta dissertação foi obtida com um *checkout* do repositório do sistema em

e estruturas de suporte a *multithreading* (`Thread::reschedule`, `Thread::pass`, `Thread::suspend`, `Thread::join`, `Thread::resume`, `Thread::yield`, `Thread::exit`, `Thread::sleep`, `Thread::wakeup`, `CPU::switch_context`, `CPU::Context:save`, `CPU::Context:load`) somam juntos 2700 bytes de memória de programa e 16 bytes de memória RAM quando compilados para o AVR. Evidentemente, uma configuração do sistema só incluirá os métodos necessários/utilizados pela aplicação.

Os métodos `in8` e `out8` fornecem acesso aos periféricos mapeados em memória do sistema. O AVR mapeia periféricos em memória tanto em uma área especial de I/O (acessada com instruções `in` e `out`) e memória geral (acessada com instruções `ld` – load e `st` – store). As instruções `in` e `out` executam em um único ciclo de memória, enquanto as instruções `ld` e `st` executam em dois ciclos. O mediador `CPU` trata o acesso a diferentes áreas de periféricos de maneira uniforme, gerando instruções `in` e `out` sempre que possível. A figura 3.12 ilustra o uso destas operações, e seu resultado em código objeto. O primeiro exemplo escreve o valor `0xAB` no endereço de memória `0x10` (área de I/O), e resulta no uso da instrução `out`. O segundo exemplo escreve o valor no endereço de memória `0x80` (área de memória geral), e resulta no uso da instrução `sts` (*Store Direct*). O deslocamento de `0x20` no endereço do segundo exemplo é resultado do mapeamento dos 32 registradores de propósito geral do AVR no início da memória. Os mecanismos para este acesso transparente são aprofundados por Polpeta et al [PFHD04].

Operação	`CPU::out8(0x10, 0xAB);`	`CPU::out8(0x80, 0xAB);`
Código Resultante	`ldi r24, 0xAB` `out 0x10, r24`	`ldi r24, 0xAB` `sts 0x00A0, r24`

Figura 3.12: Operações de entrada e saída no AVR

A arquitetura AVR não possui hardware de gerência de memória. Assim, o mediador `MMU` é uma especialização do componente básico da família, e uniforme entre todos os dispositivos da arquitetura. Há um mapeamento direto entre endereços lógicos e físicos, e o componente guarda uma lista de segmentos de memória livre para alocação dinâmica. As funções `malloc` e `free`, bem como as chamadas `new` do sistema operam sobre um `Heap` de tamanho configurável. Ainda que bastante complexos, os algoritmos de alocação, liberação e manutenção de listas de segmentos de memória tem sobrecusto bastante aceitável. Os algorítmos associados com alocação

01/05/2006. O compilador utilizado foi o GNU GCC para o AVR, versão 4.0.2, com *flag* de otimização -O2.

e liberação dinâmica de memória somam 1798 bytes de memória de programa quando utilizados pela aplicação e compilados para o AVR.

O temporizador no AVR é um dos mediadores com variações entre as diferentes `Machines`. O mediador `AVR_Timer` serve de base para implementações específicas, e define métodos de acesso aos registradores do periférico nas diferentes MCUs e métodos de conversão entre freqüência e valores de registrador. As especializações deste mediador definem a configuração de um canal de temporizador que dispara interrupções por comparação de valores. Os mediador `IC` define um símbolo `IRQ_TIMER`, que conecta os pedidos do sistema para registro de tratador e habilitação da interrupção de temporização às efetivas interrupções nas diferentes MCUs. O próprio mediador de `IC` tem implementação dependente de `Machine`, tendo em vista as diferenças de número e semântica de interrupções nas diferentes MCUs decorrente da existência de diferentes periféricos.

Os tratadores de interrupção podem ser registrados dinamicamente através do mediador `Machine`. Tendo em vista que o AVR não suporta registro dinâmico de interrupções, o sistema utiliza um *stub* que faz chamadas às interrupções registradas na `Machine`. Este *stub* é único para todas as interrupções, e o *offset* da interrupção ocorrida é calculado através de manipulações da pilha de execução. O *stub* salva o contexto de execução, calcula o *offset* da interrupção, consulta a tabela de interrupções da `Machine` para chamar a função registrada, executa a função e restaura o contexto. Para tratadores de interrupção definidos internamente nos mediadores, que não precisam salvar o contexto de execução completo para executar, é possível realizar o registro estático através das extensões do compilador GNU. As vantagens deste método de registro estático são virtualmente anuladas quando o tratador precisa salvar o contexto completo, como é o caso da maioria dos tratadores de interrupção no sistema. O vetor dinâmico de interrupções ocupa dois bytes (um ponteiro para função) para cada interrupção existente na MCU (e.g. 30 interrupções ou 60 bytes no ATmega128).

Os mediadores de `UART`, `SPI` e `EEPROM` são implementados de maneira uniforme através de componentes parametrizados. O `TSC` é emulado por um temporizador de 16-bits. Um *overflow* deste temporizador aciona um tratador de interrupções definido pelo próprio mediador, que incrementa uma variável em software. Uma leitura do `TSC` retorna a leitura do temporizador (2 bytes menos significativos) combinada com a variável em software (6 bytes mais significativos).

3.6.4 Viabilidade de Uso em Redes de Sensores sem Fios

O objetivo do EPOS de fornecer às aplicações todos os serviços necessários, e nada mais, vai ao encontro do requisito dos sistemas operacionais para redes de sensores sem fios operarem com com recursos limitados. Se um sistema só agrega à aplicação o sobrecusto estritamente necessário à execução da mesma, os recursos utilizados pelo sistema são, em teoria, mínimos. Ao mesmo tempo, o repositório de componentes do sistema disponibiliza um grande conjunto de serviços tradicionais de sistema operacional através de interfaces independentes de plataforma. Esta seção discute as perspectivas de uso do sistema em redes de sensores sem fios.

3.6.4.1 Funcionalidade Básica

A seção 3.6.2 apresenta em detalhes o projeto e a funcionalidade básica do sistema EPOS. O núcleo do sistema inclui serviços de gerência de processos, gerência dinâmica de memória e temporização, além de contar com mediadores de hardware para diversos dispositivos de armazenamento e comunicação.

3.6.4.2 Controle do Consumo de Energia

Embora várias técnicas comprovadamente eficientes tenham sido implementadas em sistemas de propósito geral, a maioria das plataformas de sistemas embarcados não podem arcar com os custos destas estratégias. Isso se dá devido a várias restrições presentes nestes sistemas, que vão desde a falta de recursos para suportá-las (e.g., processamento, memória) até a requisitos funcionais, como disponibilidade ou restrições de tempo-real. Pesquisas anteriores [HWF06a] indicam que os métodos com melhor eficácia em termos de gerência de energia são aqueles que levam em conta o comportamento das aplicações sendo executadas no sistema. Tendo ainda em vista que a maioria das aplicações de redes de sensores sem fios tem finalidade específica e dedicada, pode-se afirmar que o melhor lugar para determinar a estratégia de gerenciamento do consumo de energia é na própria aplicação.

No EPOS, a gerência de energia é realizada através de chamadas da aplicação a uma API (*Application Programming Interface*) uniforme que é implementada por todos os componentes do sistema. De modo a garantir o correto funcionamento, as relações entre componentes do sistema foram formalizadas através de Redes de Petri. Esta formalização permite não só uma análise em alto-nível dos procedimentos de migração dos componentes, mas também o estabelecimento de um mecanismo de troca de mensagens, em que os componentes se coordenam para garantir a

consistência na troca de modos de operação de subsistemas (e.g., comunicação, processamento, sensoriamento) ou de todo o sistema.

Nesta estratégia, é esperado que o programador da aplicação especifique, em seu código-fonte, quando certos componentes não estão sendo utilizados. Para isso, foi definida uma API uniforme para o gerenciamento do consumo de energia. A mesma interface permite interação da aplicação com o sistema (através de seus componentes), de componentes do sistema entre si, de componentes do sistema e dispositivos de hardware e, inclusive, o acesso direto das aplicações aos dispositivos de hardware. Para evitar que o programador tenha que, manualmente, acordar cada um destes componentes, o mecanismo de gerência abstraído pela API garante que estes componentes retomem o seu estado anterior automaticamente quando acessados.

A aplicação pode acessar um componente global (`System`), que conhece todos os componentes instanciados no sistema, provocando a alteração do modo de operação de todo o sistema. Outra forma de acesso da aplicação à API é através dos subsistemas (e.g., Comunicação, Processamento, Sensoriamento). Deste modo as mensagens são propagadas apenas para os componentes utilizados na implementação de cada subsistema. A aplicação ainda pode acessar diretamente o hardware, utilizando a API disponível nos *drivers*, como *Network Interface Card* (`NIC`), `CPU`, `Temperature_Sensor`.

De modo a aliar a portabilidade da aplicação à facilidade de desenvolvê-las, a interface foi dotada de um conjunto mínimo de métodos e de modos de operação universais, que tem sua semântica replicada em todos os componentes do sistema. Neste caso, a portabilidade vem do fato de a aplicação não necessitar implementar procedimentos específicos para cada dispositivo de hardware ao alterar seus modos de operação. Estes procedimentos são abstraídos pela API. Já a facilidade de desenvolvimento ocorre porque o programador da aplicação está liberado de analisar os manuais do hardware a fim de identificar os modos de operação disponíveis, os procedimentos para realizar as migrações e as consequências de cada uma destas mudanças. A API inclui um método para alterar o modo de operação e outro para consultá-lo, e define quatro modos universais de operação: `FULL`, `LIGHT`, `STANDBY` e `OFF`.

Além dos requisitos funcionais, também é desejável que uma estratégia de gerência do consumo de energia seja de fácil manutenção e aplicável a sistemas já existentes. Sendo o gerenciamento de energia uma propriedade não-funcional no âmbito de sistemas operacionais [LSPS05], considerou-se importante projetar esta API como um aspecto [KLM+97], podendo assim ser isolado do restante do sistema.

Uma análise detalhada do mecanismo de gerência do consumo de energia no EPOS está fora

do escopo desta dissertação. Entretanto, estudos [HWF06a, HWF06b, HWdO+06] vêm mostrando a eficiência deste sistema no contexto de aplicações de redes de sensores sem fios, bem como dando rumos ao seu desenvolvimento futuro.

3.6.4.3 Reprogramação

Os trabalhos de reconfiguração em campo no EPOS ainda estão em estágios iniciais, e caminham na direção de mecanismos de reconfiguração dinâmica da plataforma de software e hardware (quando esta é implementada em dispositivos de lógica programável) [Frö06]. Uma vez que estes mecanismos estejam bem definidos e estudados, deve ser possível adaptá-los/utilizá-los no contexto de redes de sensores sem fios. Mecanismos de reprogramação total para redes de sensores sem fios não são explorados por este trabalho. Estes métodos são bastante ineficientes em termos de consumo de energia, e podem ser suportados facilmente pelo uso de um *bootloader* externo ao sistema [Sta05].

3.6.4.4 Abstração de Hardware

No EPOS, cada hardware é tratado por um *mediador de hardware* específico. No caso particular do hardware de sensoriamento, cada dispositivo é enquadrado em uma família de mediadores de acordo com sua funcionalidade (e.g. medir temperatura ou aceleração). A interface de sensoriamento é preservada através da família, e serve como base para a implementação de abstrações de sensoriamento. O capítulo 4.2 desta dissertação apresenta os mediadores e abstrações de sensoriamento que foram projetados, implementados e adicionados ao EPOS no contexto deste trabalho.

3.6.4.5 Canal de Comunicação

O mecanismo de comunicação original do EPOS foi projetado no contexto de redes de propósito geral e computação de alta velocidade [FSP01a, FSP01b, FTSP00], e ganhou extensões para comunicação em redes de sensores sem fios, as quais são apresentadas no capítulo 4.1 desta dissertação.

3.6.4.6 Uso de Recursos Pelo Sistema

O EPOS vem sendo utilizado nos mais diferentes cenários de aplicação, de sistemas de controle de acesso a multiplexadores MPEG [SMCF06a, SMCF06b, SdMCF06]. O AVR é a “menor” arquitetura suportada atualmente pelo EPOS. O “tamanho” do sistema é altamente dependente da

aplicação: quanto menos serviços forem utilizados, menor o código agregado. Uma configuração mínima do sistema incluirá apenas estruturas e código mínimo para inicialização. Por outro lado, o sistema não restringe as aplicações: se uma aplicação precisa de um ambiente com múltiplas *threads*, sincronizadores para dados compartilhados e alocação dinâmica de memória, o sistema entrega estes serviços de maneira funcional e *enxuta*.

Capítulo 4

Desenvolvimento de Suporte para Comunicação e Sensoriamento

Este capítulo apresenta o desenvolvimento de suporte para comunicação e sensoriamento desenvolvido neste trabalho, com base no sistema operacional EPOS.

4.1 Suporte Para Comunicação

A simplicidade do hardware de comunicação para redes de sensores faz com que o protocolo de controle de acesso ao meio (MAC – *Media Access Control*) e outros serviços da camada de enlace de dados tenham que ser implementados em software. Serviços tipicamente implementados em hardware, como detecção de pacotes de dados, detecção e tratamento de erros, endereçamento, filtragem de pacotes não só integram, mas tornam-se a parte principal da pilha de comunicações implementada por um sistema operacional para redes de sensores sem fios.

Conforme apresentado na seção 2.3, o custo em termos de consumo de energia do envio e recepção de dados nos rádios atuais é ordens de magnitude maior do que o custo da execução de instruções em um micro-controlador. Um sistema de comunicação para redes de sensores sem fios deve, portanto, utilizar os recursos de comunicação de forma conservadora, fornecendo não mais do que aplicações específicas necessitam para implementar seus serviços.

Por outro lado, a simplicidade dos rádios de comunicação para redes de sensores sem fios não é um fator limitante, mas sim uma característica desejável, já que dá liberdade total de configuração do canal de transmissão de dados. Estudos [LH05, PSC05] mostram que protocolos de controle de acesso ao meio implementados em software adequadamente projetados e adaptados à aplicação

podem ser mais eficientes do que protocolos padronizados, implementados em hardware.

Diversos protocolos foram projetados e implementados para controle de acesso ao meio em redes de sensores sem fios [EH02, HH04, LKR04, WC01, MB04, RR05, DL03, PHC04, YHE02]. Neste cenário, onde aplicações específicas podem ter cargas de trabalho de comunicação muito diferentes entre si, adaptabilidade e configuração determinada pela aplicação são fatores de grande importância. Entretanto, as implementações destes protocolos em sistemas reais muitas vezes não fornecem mecanismos de configuração adequada do canal de dados às aplicações. De fato, muitos protocolos, como o S-MAC (apresentado na seção 2.3.2) são projetados especificamente para otimizar determinadas cargas de trabalho (e.g. comunicação *multi-hop*), e apresentam pouca ou nenhuma oportunidade de configuração pela aplicação.

Este capítulo apresenta o projeto e implementação do C-MAC (Configurable MAC), um *protocolo configurável* de acesso ao meio para redes de sensores sem fios. O C-MAC funciona como um *framework* de estratégias de controle de acesso ao meio, com um sistema de configuração transparente. O protocolo agrega diferentes serviços (e.g. sincronização, detecção de dados, mensagens de confirmação, contenção, envio e recepção), implementados sob diferentes estratégias. Aplicações podem configurar diferentes parâmetros de comunicação em tempo de compilação e execução. O C-MAC deve seus conceitos fundamentais a diferentes protocolos de controle de acesso ao meio, como Aloha, B-MAC e S-MAC, e apresenta, um novo projeto para soluções conhecidamente eficazes para comunicação sem fios de baixa potência, permitindo que aplicações tomem proveito de estratégias que venham ao encontro de suas necessidades. A seção 4.1.1 apresenta os *mediadores de hardware* responsáveis por dar suporte de baixo nível à comunicação no transceptor de rádio CC1000. A seção 4.1.2 apresenta o projeto e implementação do protocolo C-MAC. A seção 4.1.3 apresenta o subsistema de comunicação do EPOS, e a integração do C-MAC com o sistema. Finalmente, a seção 4.1.4 apresenta algumas perspectivas sobre o projeto e uso do C-MAC em redes de sensores sem fios.

4.1.1 Mediador de Hardware CC1000

O transceptor de rádio CC1000 [Chi04a], fabricado pela Chipcon AS, é um transmissor/receptor de rádio FM UHF implementado em um único chip. Este transceptor é utilizado nos *motes* Mica2 e BTnode, e apresenta as seguintes características:

- Freqüência programável entre 300 e 1000 MHz, em passos de 250 Hz. A faixa de freqüência selecionável por software é dependente do circuito externo ao chip.

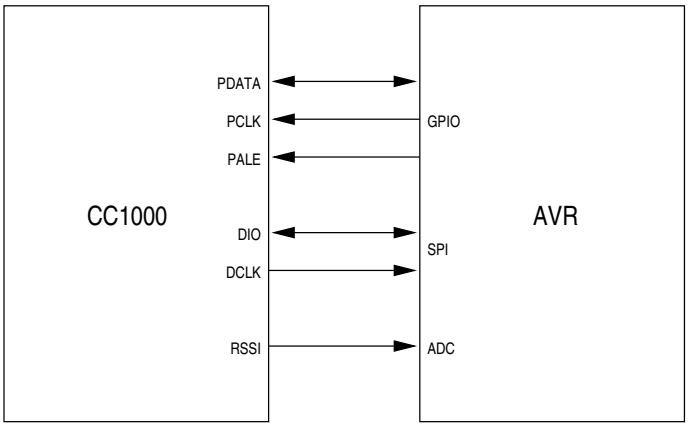


Figura 4.1: Ligação entre CC1000 e Micro-controlador

- Modulação por chaveamento de freqüência (FSK – *Frequency-Shift keying*), com taxas de dados de ate 76.8 kBaud,
- Codificação Manchester ou NRZ (*Non-Return-to-Zero*) em hardware.
- Indicador de força do sinal recebido do rádio (RSSI - *Received Signal Strength Indicator*).
- Potência de transmissão selecionável.

As características de consumo de energia e taxa máxima de transmissão, do CC1000 não o colocam nem como o mais econômico em termos de consumo de energia nem como o mais rápido dos rádios de baixa potência atuais (ver seção 2.2). Rádios mais simples e econômicos, como o RFM1000, não implementam muitas das características do CC1000 (e.g. codificação, freqüência programável). Já rádios mais sofisticados, como o CC2420, apresentam taxas de dados mais elevadas, implementam mais características (e.g. MAC) em hardware, mas têm um custo de energia mais elevado. O CC1000 é, portanto, um bom compromisso entre simplicidade/consumo de energia e sofisticação/velocidade de transmissão.

O CC1000 é configurado através de uma interface de três fios, e troca dados com o micro-controlador através de uma interface de dois fios. A figura 4.1 ilustra a ligação entre o CC1000 e o microcontrolador AVR na plataforma Mica2. O CC1000 tem 28 registradores de configuração de 8-bits, que são acessados por um endereço de 7-bits mais um bit de indicação de leitura ou escrita. A interface de configuração é implementada pelo microcontrolador, que fornece um *clock* para dados (PCLK), uma linha de dados através da qual bits são serializados para envio e recepção (PDATA). Uma linha PALE (*Program Address Latch Enable*) indica se os bits na linha PDATA são de endereçamento ou de dados. A interface de dados é comandada pelo CC1000 que, enquanto

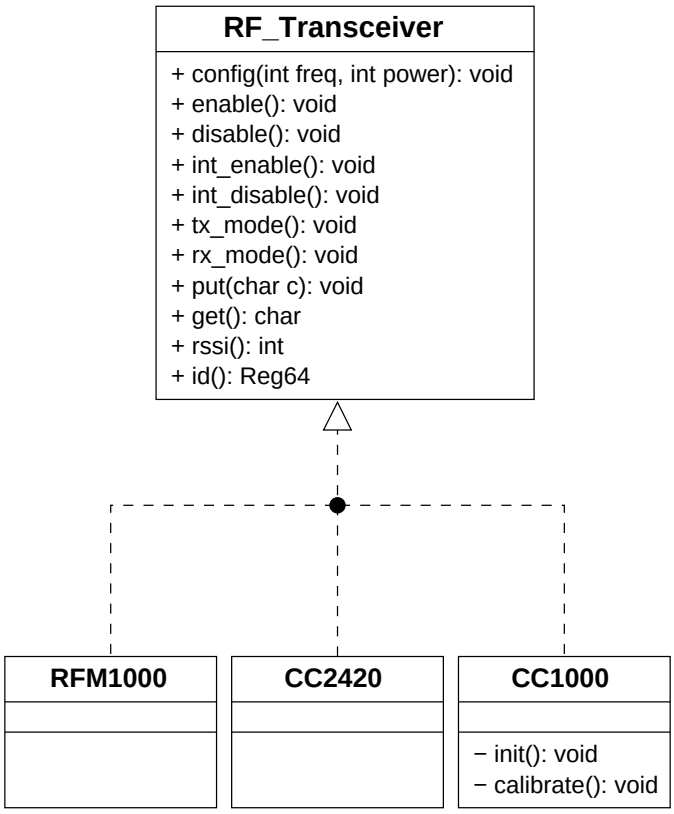


Figura 4.2: Família `RF_Transceiver`

ligado, mantém o *clock* de dados ativo, independente de estar ou não recebendo dados. No Mica2, o micro-controlador conecta-se a esta interface através de seu controlador SPI (*Serial Peripheral Interface*).

O cálculo dos valores de registradores do CC1000 para diferentes configurações de freqüência, modulação e potência é bastante complexo, e tomaria tempo considerável de execução em um micro-controlador de 8-bits. A Chipcon fornece uma ferramenta para cálculo prévio de configurações em diferentes situações de uso, que podem ser armazenadas na memória do micro-controlador para uso em tempo de execução.

No EPOS, o CC1000 foi abstraído como um mediador membro da família `RF_Transceiver` (Figura 4.2). Esta família abstrai dispositivos de comunicação com interfaces a bits ou bytes (em contraste com dispositivos com interface de pacotes), fornecendo a visão de um modulador de dados em série, *half-duplex*. Há métodos para configuração em diferentes freqüências e potências de envio, para obtenção de um indicador de força de sinal (RSSI), e de um identificador numérico para endereçamento.

O nível de abstração fornecido pela família `RF_Transceiver` permite que implementações de protocolos de controle de acesso ao meio utilizem diferentes dispositi-

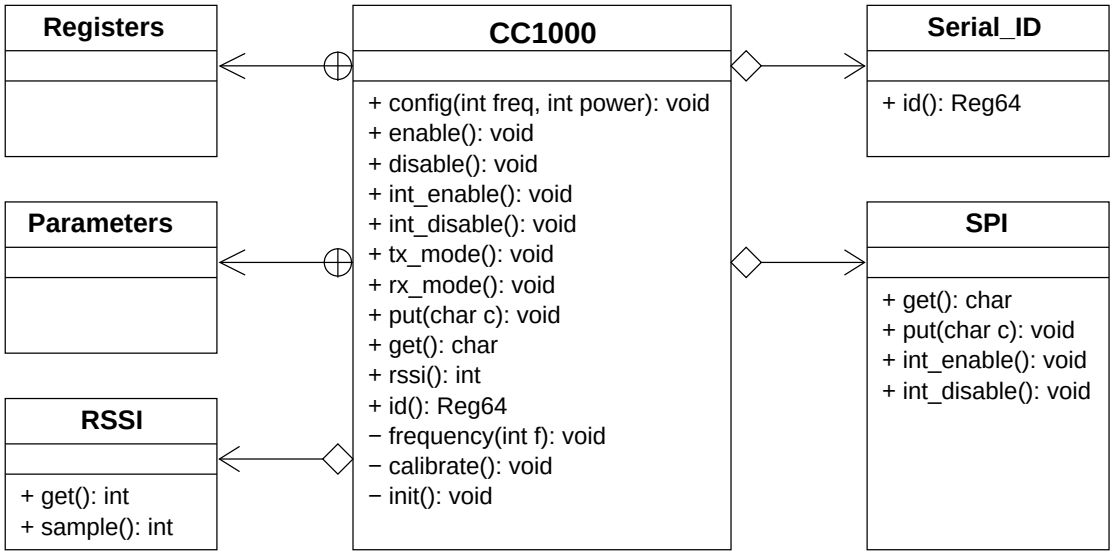


Figura 4.3: Mediador `CC1000`

vos transparentemente. Um nível de abstração mais baixo exporia detalhes do hardware, e um nível mais alto interferiria nas políticas de controle de acesso ao meio. Muitos dispositivos desta categoria fornecem apenas parte da funcionalidade necessária à implementação da família. Desta forma, a implementação do mediador `CC1000` agrega funcionalidades de diferentes dispositivos (o próprio hardware do CC1000, uma interface SPI, um sensor RSSI e um hardware de identificação serial) para satisfazer a interface da família (Figura 4.3).

A interface de comunicação entre o rádio CC1000 e o microcontrolador é dependente de plataforma. Desta forma, o mediador `CC1000` utiliza a interface interna `Registers` para abstrair o acesso aos diferentes registradores de configuração do dispositivo de forma independente de arquitetura. A interface exporta métodos de acesso aos registradores, e suas implementações abstraem o mecanismo de leitura e escrita dos mesmos. Da mesma forma, a classe interna `Parameters` é usada para armazenar parâmetros de registradores pré-calculados para diversas configurações. De maneira a não agregar dados à memória RAM do sistema, estes valores são armazenados na memória FLASH de programa.

A comunicação de dados entre o microcontrolador e o rádio CC1000 se dá por uma interface `SPI`, que é abstraída por seu mediador adequado. A classe `ID` fornece um identificador de 64-bits que pode ser usado por protocolos de controle de acesso ao meio para endereçamento. Na plataforma Mica2, essa interface dá acesso a um *chip* de identificação serial único. Em plataformas que não possuam hardware de identificação, uma implementação desta interface pode utilizar uma assinatura armazenada em memória FLASH. O indicador de força de sinal recebido do CC1000 é

abstraído pela classe `RSSI`, que implementa um membro da família `Sensor` (Capítulo 4.2).

A utilização destas estruturas internas permite que a implementação do mediador `CC1000` seja independente de plataforma, e lide somente com os procedimentos de inicialização e configuração do hardware, sem perder eficiência. Os parâmetros de freqüência e potência de operação podem ser definidos na inicialização do componente, ou alterados em tempo de execução. Já os parâmetros de configuração interna (e.g. configuração do modulador) podem ser ajustados através dos `Traits` de configuração estática do componente.

4.1.2 Projeto e Implementação do C-MAC

O C-MAC (Configurable MAC) é um *protocolo configurável* de acesso ao meio para redes de sensores sem fios equipadas com transceptores de rádio de baixa potência. Sua característica configurável permite que o usuário ajuste diversos parâmetros de comunicação (e.g. sincronização, detecção de dados, sinais de confirmação, contenção, envio e recepção) de maneira a adequar o protocolo às necessidades de diferentes aplicações.

A seção 2.3 apresenta as motivações para um protocolo configurável de controle de acesso ao meio para redes de sensores sem fios. Sistemas de comunicação configurável também tem alcançado sucesso no contexto de computação de alto desempenho em grades. Dos Santos [dS05] desenvolveu um sistema de composição estática de protocolos leves de comunicação para computação em grades que apresentou diversas vantagens com relação a arquiteturas em camadas e de natureza monolítica, como os encontrados na arquitetura da Internet (TCP/IP). Este sistema é constituído por um framework meta-programado, responsável por prover mecanismos que permitem selecionar, configurar e combinar protocolos de comunicação de acordo com os requisitos da aplicação, e um núcleo básico de comunicação sobre o qual os protocolos são projetados. Esse paradigma oferece diversas vantagens, incluindo a habilidade de criar novos serviços de comunicação sob demanda e permitir que aplicações experimentem com diferentes configurações de protocolo de comunicação, coletando métricas para identificar a melhor configuração para as suas necessidades.

O C-MAC utiliza estas mesmas premissas para construir um núcleo básico de comunicação configurável, sobre o qual outros protocolos podem ser compostos (Seção 4.1.3). Conforme apresentado na Seção 2.3, as responsabilidades delegadas ao software no controle de acesso ao meio em redes de sensores sem fios excedem as de um *driver* tradicional para placas de rede na computação propósito geral, e incluem detecção de canal livre, sincronização local e global e mensagens de

confirmação, além da definição de um formato de pacote de dados.

A escolha dos pontos configuráveis da arquitetura do protocolo C-MAC foi feita avaliando as características dos principais MACs para redes de sensores sem fios, e tendo como objetivo fornecer a maior gama de pontos configuráveis que, quando combinados, formassem um protocolo MAC completo. A configuração do protocolo em tempo de execução, ainda que desejável, não foi tratada na arquitetura do C-MAC. O sobrecusto de manter todas as possibilidades de configuração em memória, além da necessidade de um protocolo secundário para troca de configurações entre nodos torna impraticável o uso em tempo de execução de um mecanismo de configuração tão amplo quanto o C-MAC. Os pontos de configuração do C-MAC incluem:

Características Básicas da Comunicação: Estas características são tratadas pelo hardware de comunicação, e incluem: freqüência e potência de transmissão (alteráveis em tempo de execução); tipo de modulação (e.g. Manchester, NRZ); velocidade de transmissão.

Período Ativo: O período ativo indica quando o rádio pode operar. Em um protocolo simples baseado em CSMA, o rádio pode transmitir em qualquer momento que detecte um canal livre. Por outro lado, em um protocolo organizado em *slots*, este período está limitado à parte ativa do *slot* de tempo do protocolo.

Mecanismo para Evitar Colisões: O mecanismo para evitar colisões em um MAC para redes de sensores sem fios pode ser um algoritmo para detecção de atividade no canal, uma troca de pacotes *Request to Send* (RTS) e *Clear to Send* (CTS), uma combinação entre os dois. Deve haver ainda a possibilidade de não utilizar nenhum mecanismo para evitar colisões para uso, por exemplo, em uma rede esparsa com pouca comunicação, onde retransmitir eventuais pacotes corrompidos é menos custoso do que o mecanismo em si.

Mecanismo para Detectar Colisões: Em uma rede de sensores sem fios, o mecanismo para detectar colisões mais utilizado é o de mensagens de confirmação (*Acknowledgement*), enviados pelo nodo receptor para indicar que os dados foram recebidos. Em situações onde a perda de pacotes não é um problema (e.g. uma rede densamente distribuída, onde muitas informações serão redundantes), pode-se eliminar o mecanismo para detectar colisões, diminuindo o consumo de energia do protocolo.

Mecanismo para Tratar Colisões: Quando uma colisão é detectada, o protocolo pode escolher entre retransmitir o pacote, ou simplesmente incrementar um contador de estatística de pa-

cotes perdidos. Em caso de retransmissão, um algoritmo de atraso (*backoff*) pode ser usado para minimizar a chance de outra colisão ocorrer.

A seguinte seqüência permite ilustrar o procedimento básico de comunicação entre os nodos: O nodo transmissor detecta o canal livre e envia um preâmbulo (uma seqüência de zeros e uns alternados) por um período maior do que o período inativo do nodo receptor, seguido por uma seqüência de sincronização conhecida pelo receptor. O nodo receptor, quando detecta uma seqüência de zeros e uns maior do que um valor mínimo, passa a procurar pela seqüência de sincronização (uma quebra no padrão do preâmbulo). Quando encontra esta seqüência, o nodo receptor sincroniza-se com o transmissor, encontrando seu deslocamento temporal com relação a este. A figura 4.4 ilustra este processo de sincronização. Uma vez que o receptor está sincronizado com o transmissor, passa a receber os dados, verifica o CRC do pacote recebido, e envia uma confirmação. No envio da confirmação, há um novo processo de sincronização, desta vez sem a verificação do canal (que já está "reservado" para a comunicação atual), e com um preâmbulo mais curto (já que o transmissor passa a esperá-lo imediatamente após enviar seus dados). As figuras 4.5 e 4.6 apresentam, respectivamente, comunicações bem e mal sucedidas utilizando esta configuração de protocolo.

O C-MAC é implementado através de uma máquina de estado cujas transições são ativadas pelas interrupções de um temporizador dedicado (para controlar o período ativo e *backoffs*) e pelas interrupções de dados do hardware de comunicação. A figura 4.7 apresenta uma visão simplificada desta máquina de estados. Nesta figura, as transições de um estado para ele mesmo não estão representadas.

As diversas características configuráveis do C-MAC são selecionadas pelo programador através dos `Traits` de configuração do EPOS. `Traits` são classes parametrizadas cujos membros constantes estáticos descrevem as propriedades de uma certa classe. Quando determinada propriedade é selecionada, a funcionalidade que elati descreve é incluída no protocolo. Por outro lado, devido ao uso de meta-programação estática e *inlining* de funções, quando uma característica não é selecionada, nenhum sobrecusto relativo a ela é adicionado ao código objeto final do protocolo.

O componente de software do C-MAC pertence à família `Low_Power_Radio`, que descreve um conjunto de métodos e estruturas comuns aos protocolos de controle de acesso ao meio para rádios de baixa potência. Esta família define o formato de pacote, já apresentado nesta seção, o tamanho da palavra de endereçamento, uma estrutura para armazenamento de estatísticas de transmissão, além dos métodos para envio e recebimento de *frames* de dados. A figura 4.8 apresenta o

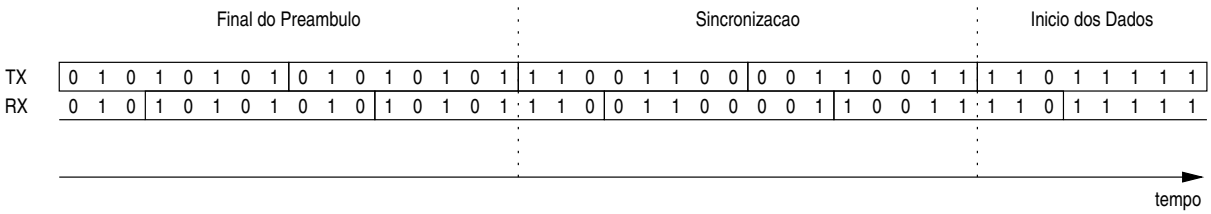


Figura 4.4: Sincronização entre nodos no C-MAC

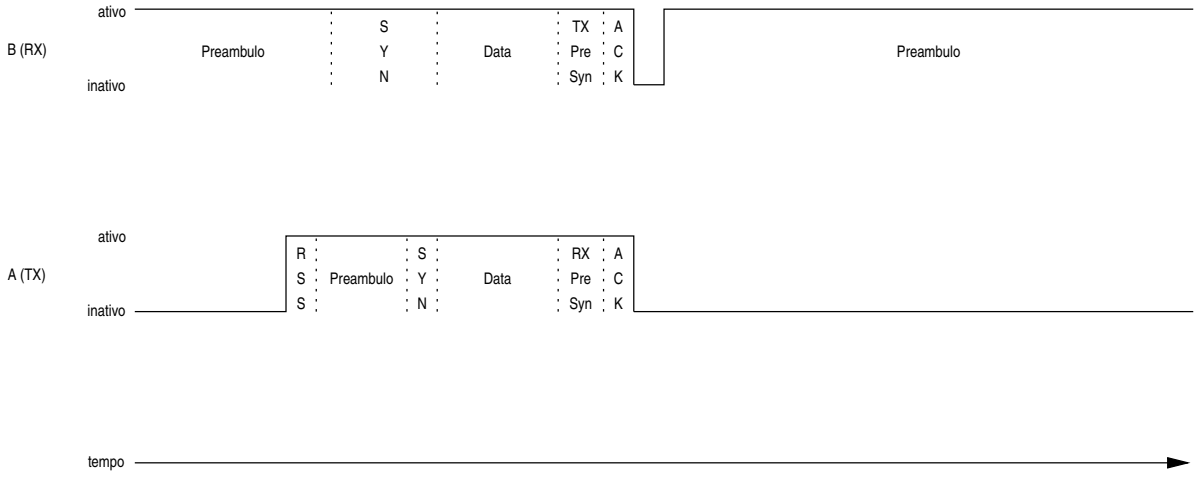


Figura 4.5: Comunicação bem sucedida no C-MAC/CSMA Simples

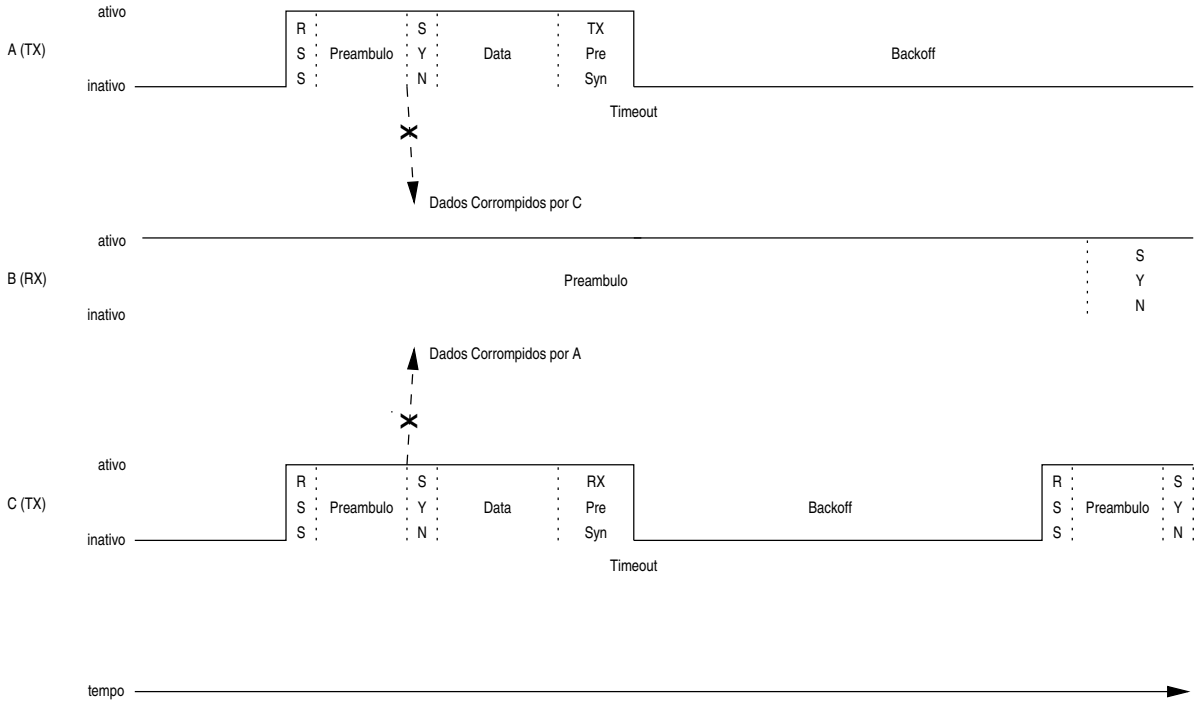


Figura 4.6: Comunicações mal sucedidas no C-MAC/CSMA Simples

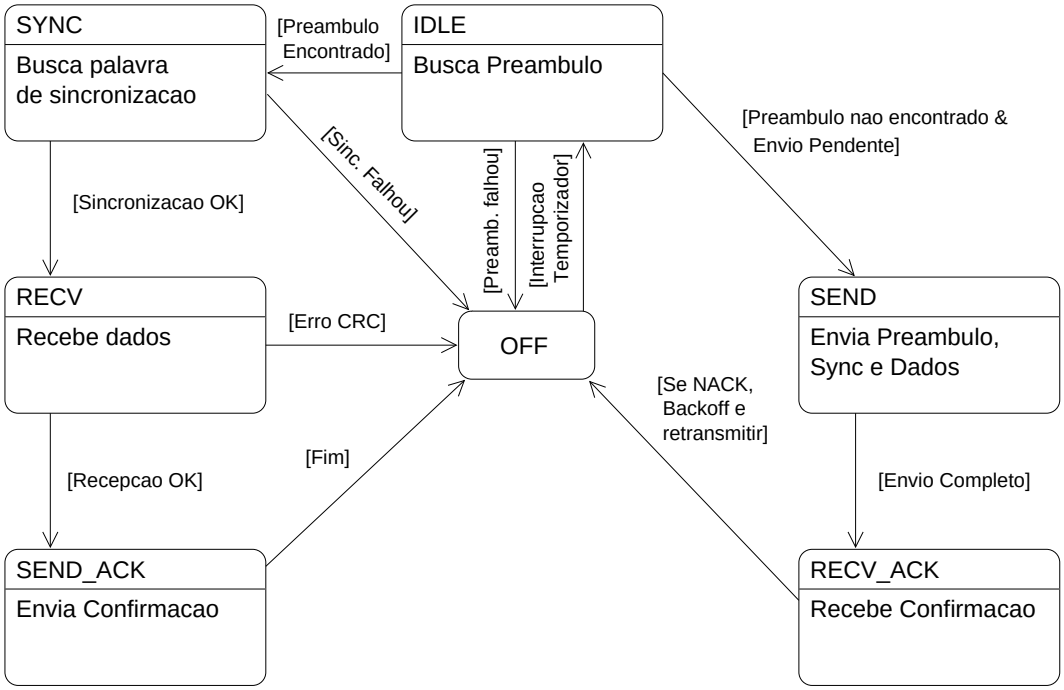


Figura 4.7: Máquina de Estados Simplificada do C-MAC

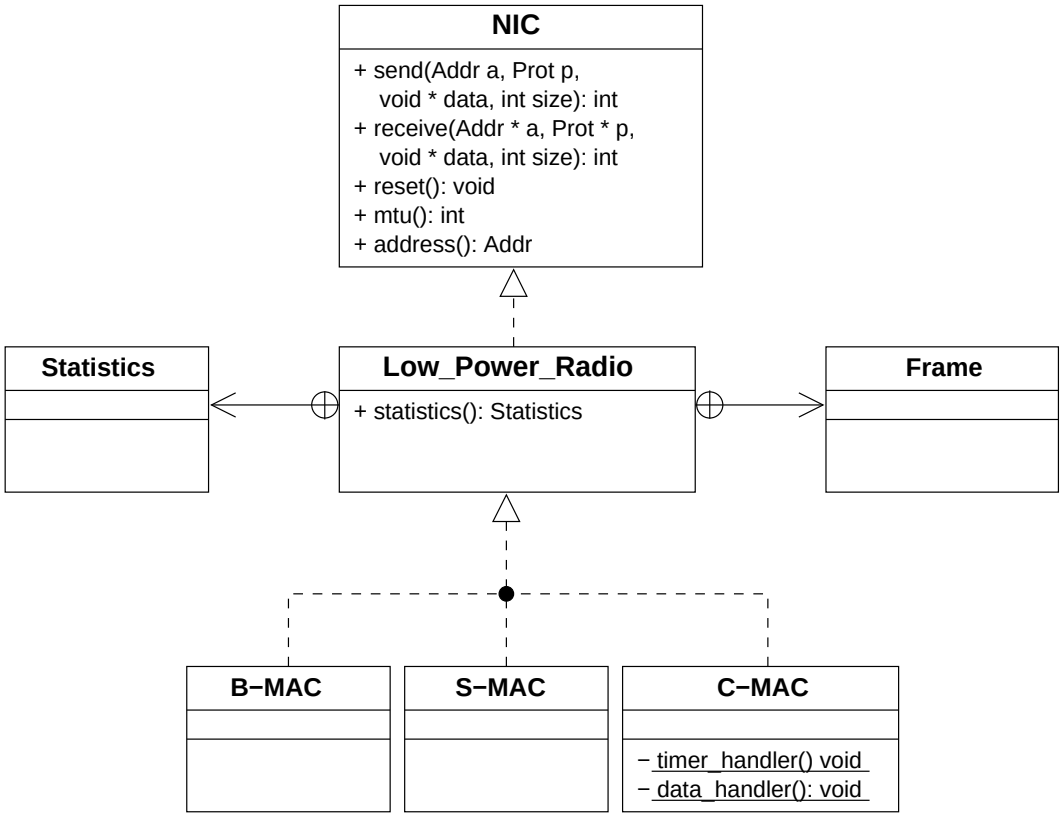


Figura 4.8: Família `Low_Power_Radio`

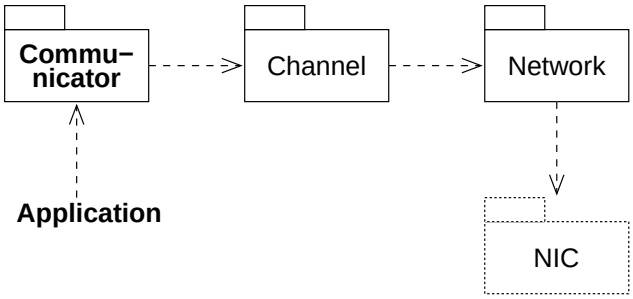


Figura 4.9: Comunicação no EPOS

diagrama de classe da família `Low_Power_Radio`.

4.1.3 Comunicação no EPOS

No EPOS, os processos de aplicação comunicam-se entre si através de um Comunicador (`Communicator`), que age como uma interface para comunicação através de um Canal (`Channel`) implementado sobre uma Rede (`Network`) [Frö01]. A figura 4.9 ilustra as famílias de abstrações envolvidas na comunicação entre processos no EPOS.

Membros da família `Communicator` são pontos de acesso para um canal de comunicação, incluindo interfaces para acesso assíncrono de memória em um nodo remoto (*Asynchronous Remote Memory Segment*), e interfaces para tratadores de mensagens ativas (*Active Message Handler*). Neste último tipo de `Communicator`, as mensagens transportam, além dos dados, uma referência para um tratador que é invocado, no contexto do nodo receptor, para tratar a mensagem na recepção. Os membros da família `Channel` implementam protocolos de comunicação classificados como nível quatro (transporte) no modelo OSI [ISO81], incluindo membros como `Stream` e `Datagram`. Membros da família `Network` fornecem a abstração de uma rede, no sentido que uma rede fornece os meios físicos para construir canais lógicos. Membros desta família abstraem as particularidades de cada tecnologia de rede, de maneira que, do ponto de vista dos membros da família `Channel`, todas as `Networks` são equivalentes.

O trabalho desenvolvido nesta dissertação concentrou-se na comunicação ponto-a-ponto entre nodos em uma rede, que é responsabilidade dos membros da família `NIC` (Seção 4.1.2). Protocolos de transporte e roteamento, além do uso de mensagens ativas para comunicação em redes de sensores sem fios fazem parte dos trabalhos em desenvolvimento e futuros relacionados a esta dissertação (seção 5).

4.1.4 Avaliação

Os rádios de comunicação de baixa potência tipicamente utilizados em redes de sensores sem fios tem funcionalidade reduzida com relação a outros dispositivos de comunicação sem fios como rádios Bluetooth ou IEEE 802.15.11. Por um lado isso dá um amplo grau de adaptabilidade às estratégias de comunicação permitindo, por exemplo, o uso de protocolos de controle de acesso ao meio ajustados às necessidades de diferentes aplicações. Por outro lado, a complexidade do software que abstrai este hardware e implementa estas estratégias de comunicação. Esta seção avalia a complexidade e desempenho dos componentes de software apresentados neste capítulo, em particular do mediador de hardware `CC1000`, responsável por abstrair o hardware de comunicação, e o componente `C-MAC`, que implementa um protocolo de controle de acesso ao meio configurável para redes de sensores sem fios.

Como nos demais testes desta dissertação, a versão do EPOS utilizada neste teste e nos testes subseqüentes foi obtida com um *checkout* do repositório do sistema em 01/05/2006. O compilador utilizado foi o GNU GCC para o AVR, versão 4.0.2, com *flag* de otimização -O2. Os testes com o TinyOS foram realizados com a versão 1.1 do sistema, e os testes com o MANTIS OS, com a versão 0.95 do sistema.

Dada a inexistência de um relógio de tempo real nas plataformas utilizadas, os testes de desempenho foram feitos utilizando o temporizador dedicado do *Time-Stamp Counter* (`TSC`) do EPOS-AVR. Este temporizador foi configurado para uma resolução de 115200 Hz, ou precisão da ordem de 10 microssegundos, na plataforma Mica2. Sempre que não especificado de outra forma, os testes foram executados marcando o tempo antes e depois da execução de uma série de 1000 iterações do procedimento em questão. Este teste é repetido no mínimo cinco vezes, e as médias entre as iterações das cinco repetições são apresentadas.

A tabela 4.1 apresenta o tamanho de código de dados para cada um dos componentes do mediador `CC1000`. O maior componente do mediador é o `Registers`, que abstrai o acesso à interface de configuração do CC1000, e implementa a lógica de uma interface de comunicação de dois fios em software. O componente que faz a leitura do identificador único da plataforma (`Serial_ID`) implementa uma interface de um fio para leitura de dados em série, e também ocupa uma parcela significativa do total dos recursos utilizados pelo componente. Ao componente `Parameters` são adicionados 44 bytes de memória de código para cada configuração de freqüência e potência desejada (mínimo de uma). O componente `CC1000` como um todo utiliza 2184 bytes de memória de código e 13 bytes de memória de dados.

Componente	Código		Dados	
	bytes	%	bytes	%
SPI	20	1	0	0
Serial_ID	700	32	8	62
RSSI	48	2	1	8
Parameters	34	2	4	30
Registers	1180	54	0	0
CC1000	202	9	0	0
Total	2184	100	13	100

Tabela 4.1: Tamanho dos componentes do mediador `CC1000`

Procedimento	Tempo de Execução (ms)	Tempo Máximo Ideal (ms)
Configuração	31.34	70
Wakeup	2.197	2.2
Troca de Modo (RX)	0.37	0.25
Troca de Modo (TX)	0.39	0.27

Tabela 4.2: Tempo de execução de procedimentos do mediador `CC1000`

A tabela 4.2 apresenta os tempos de execução dos principais procedimentos de configuração do CC1000: configuração completa, tempo de ativação a partir de um modo de *stand-by*, e tempo de troca de modo de recepção para envio e vice-versa. O dispositivo sofre uma configuração completa cada vez que seus parâmetros de operação (freqüência, modulador e potência de transmissão) são alterados. O *wakeup* acontece sempre que o dispositivo sai do modo de baixo consumo e passa a receber ou enviar dados. A troca de modo ocorre, por exemplo, quando um nodo que estava recebendo dados passa a enviar uma confirmação. A tabela apresenta também o tempo máximo ideal do procedimento, isto é, o tempo máximo, indicado pela documentação do hardware, em que o dispositivo pode completar o procedimento, descontando qualquer execução de software no controlador. A proximidade entre o tempo de execução medido e o ideal dá-se principalmente por dois fatores: o baixo sobrecusto do sistema, que não gasta tempo de processamento considerável executando seus procedimentos internos e de controle, e a implementação baseada em *polling* dos métodos, que dispensam o uso de um temporizador e permitem que o rádio passe a operar o mais

Parâmetro	Valor
Período Ativo	100%
Potência de Transmissão	5 dBm
Modulador	19.2 kbps, Codificação Manchester
Atraso aleatório de transmissão	0 ms
Detecção e Tratamento de Colisões	Nenhum
Taxa de dados máxima (teórica)	16.4 kbps

Tabela 4.3: Parâmetros de comunicação utilizados

Sistema	Código (bytes)	Dados (bytes)
EPOS	3888	108
TinyOS	8562	205

Tabela 4.4: Sobrecusto da implementação do protocolo nos diferentes sistemas

rapidamente possível.

Dada a natureza altamente modular do projeto das estratégias de comunicação apresentadas nesta dissertação, não é possível fazer uma comparação direta entre o mediador `CC1000` e outros *drivers* para este hardware presentes em outros sistemas, já que não há neles um componente equivalente. Para permitir comparações do C-MAC com soluções existentes em outros sistemas, este foi configurado de maneira a funcionar analogamente ao protocolo B-MAC, apresentado na seção 2.3.1 e presente no sistema TinyOS. Quando da realização dos testes, a implementação do protocolo B-MAC no sistema MANTIS OS encontrava-se desatualizada com relação ao restante do sistema, e não pode ser testada. A tabela 4.3 resume a configuração utilizada nos sistemas e a tabela 4.4 apresenta o sobrecusto de memória de código e dados das implementações do protocolo nos diferentes sistemas. Diferenças entre estes sobrecustos dão-se principalmente pelo modelo de componentes do sistema, e dependências inexistentes entre componentes artificialmente inseridas por falhas de projeto.

A figura 4.10 representa a perda de pacotes do protocolo em diferentes distâncias. Para este teste, dois nodos foram colocados em linha de visão em um campo aberto, afastados aproximadamente um metro do chão, e afastados entre si de 5 a 100 metros. Um dos nodos passa a enviar 1000 pacotes de dados para o outro nodo, que ao final de um período de tempo reporta quantos pacotes recebeu corretamente. O teste foi repetido três vezes para cada distância avaliada, e uma média

dos pacotes perdidos é apresentada. Um teste análogo foi realizado em um ambiente interior, onde os nodos estavam afastados entre si por 20 metros, e foi variada a potência de transmissão. A figura 4.11 apresenta os resultados deste teste.

A figura 4.12 apresenta a perda de pacotes na rede com um número de nodos variando de dois a quatro, nos sistemas avaliados. Neste teste, um nodo foi posicionado no centro de um círculo imaginário com raio de dois metros, e os demais nodos foram posicionados na borda deste círculo, eqüidistantes entre si. Todos os nodos estão ao alcance e potencialmente interferem nas comunicações dos demais. Cada um dos nodos na borda do círculo envia 1000 pacotes de dados para o nodo no centro, e este mede o número total de pacotes recebidos corretamente. A figura 4.13 apresenta a vazão da rede nesta mesma configuração. As diferenças entre os sistemas configurados de forma idêntica podem ser ocasionadas por uma série de fatores, incluindo variações no ambiente de testes e heterogeneidade entre os nodos (e.g. antenas, pequenas diferenças na carga de bateria); perda de interrupções; e sobrecusto dos procedimentos internos e de controle do sistema. Cabe notar que as repetições dos testes tendem a diminuir o impacto da variação ambiental e heterogeneidade.

O C-MAC apresentou desempenho equivalente aos protocolos existentes nos outros sistemas, com um sobrecusto de memória menor. Esta vantagem é ampliada pelo sistema de configuração do C-MAC, que permite a criação de protocolos específicos conforme a necessidade das aplicações, adicionando somente o sobrecusto associado às funcionalidades selecionadas. O projeto modular do protocolo permite ainda que diferentes transceptores de rádio sejam utilizados sem alterações no protocolo.

4.2 Suporte Para Sensoriamento

Os dispositivos sensores formam, juntamente com os subsistemas de processamento e comunicação, o núcleo de um nodo de redes de sensores sem fios. O uso de um sensor específico é determinado pelos requisitos das aplicações (e.g. monitorar temperatura, detectar campos magnéticos), e os nodos de sensor contemporâneos foram projetados para permitir ampla modularidade de dispositivos sensores, permitindo a troca, inclusão ou remoção de sensores de acordo com estes requisitos.

Do ponto de vista das aplicações, a operação com sensores deve ser bastante simples. Em geral, aplicações de sensoriamento monitoram leituras de um determinado sensor e realizam diferentes ações de acordo com os valores obtidos nestas leituras. Entretanto, conforme ilustrado na

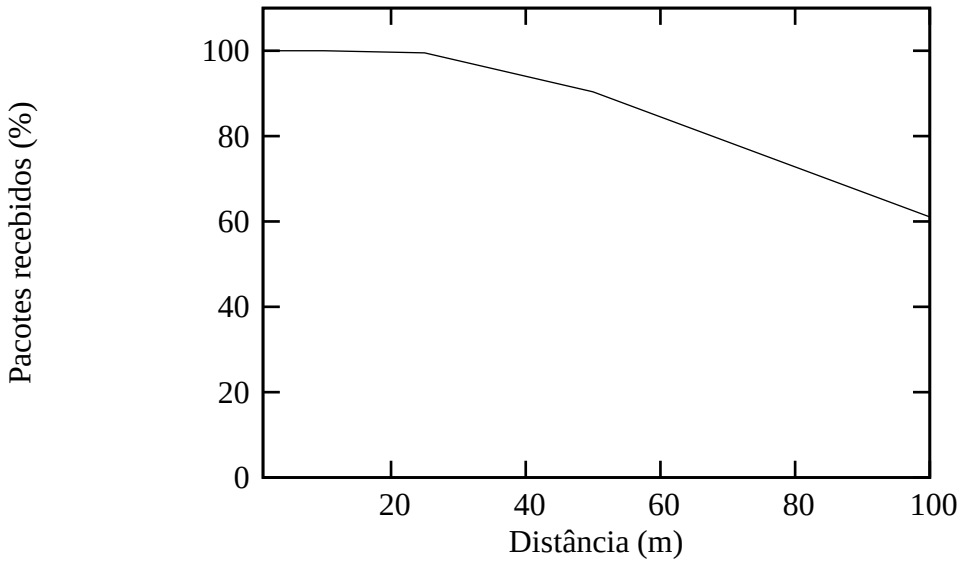


Figura 4.10: Taxa de pacotes recebidos variando a distância (C-MAC)

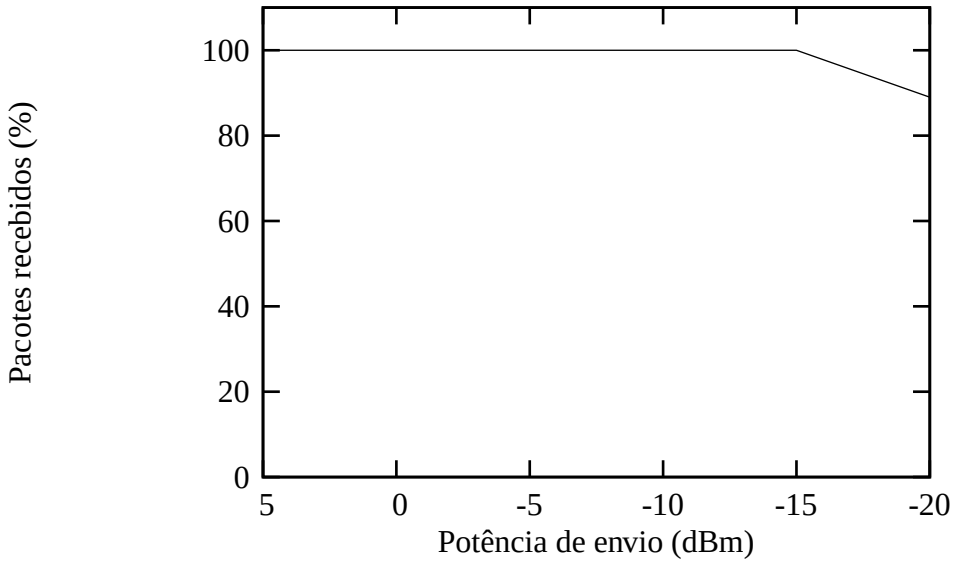


Figura 4.11: Taxa de pacotes recebidos variando a potência de envio (C-MAC)

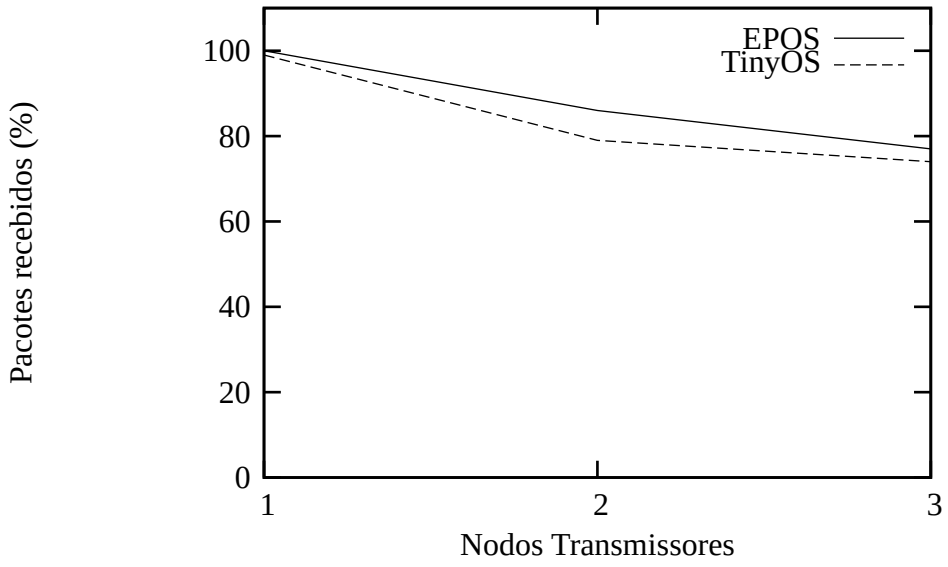


Figura 4.12: Taxa de pacotes recebidos variando o número de nodos (B-MAC e C-MAC)

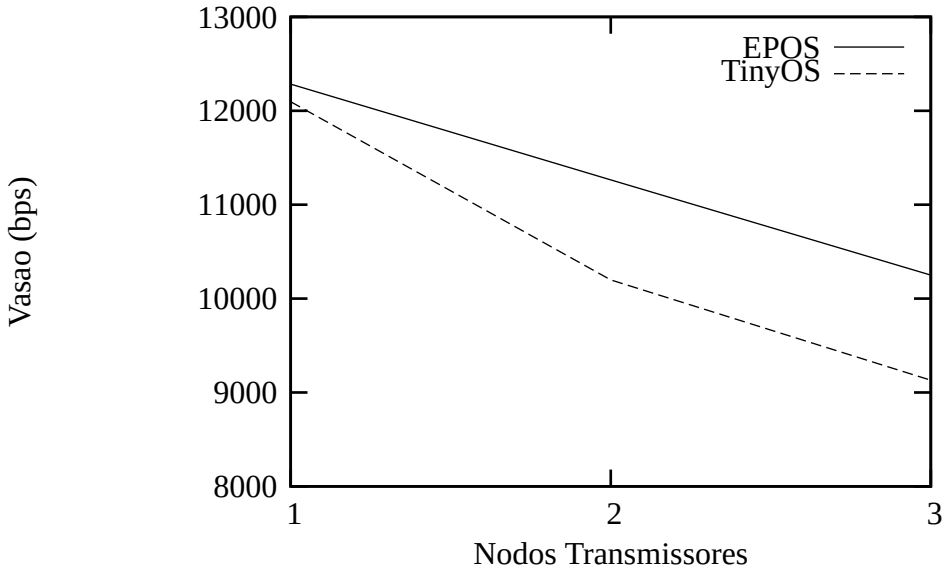


Figura 4.13: Vasão da rede variando o número de nodos (B-MAC e C-MAC)

seção 2.4, diferentes módulos de dispositivos sensores que apresentam a mesma funcionalidade entre si (e.g. medir temperatura ou campo magnético) muitas vezes variam em suas interfaces de acesso e características e parâmetros operacionais. Neste cenário, uma aplicação de sensoriamento desenvolvida para uma certa plataforma raramente será portável para uma plataforma diferente, a menos que o sistema de suporte de execução destas plataformas entregue às aplicações mecanismos que encapsulem a plataforma de sensores de maneira adequada.

O capítulo 3 desta dissertação apresentou as soluções de sensoriamento de diferentes sistemas operacionais para redes de sensores sem fios. No sistema TinyOS, não há abstrações independentes de plataforma específicas para dispositivos sensores além da interface de canal de ADC. Isto faz com que as aplicações tenham que completar a funcionalidade dos *drivers* de sensores, comprometendo a portabilidade das mesmas. O sistema MANTIS OS utiliza funções em estilo POSIX para abstração de hardware. Cada função recebe como parâmetro o dispositivo a ser tratado, e uma tabela de ponteiros para funções redireciona em tempo de execução as chamadas gerais às chamadas específicas. Os parâmetros das funções são específicos para cada dispositivo, e cada *driver* de sensor tem semântica específica. O sistema SOS utiliza módulos carregáveis para abstrair dispositivos de hardware. Através destes, um sensor analógico *conecta-se* a um canal de ADC e registra um tipo de sensor (e.g. `PHOTO`). Quando a aplicação requisita dados de um tipo de sensor, o núcleo do sistema envia o pedido para o *driver* registrado e recebe a leitura de ADC apropriada. Esta solução aproxima-se de uma abstração consistente, já que modelos diferentes de hardware com a mesma função podem ser abstraídos de forma similar, ainda que com grande sobrecusto. O registro de *drivers* ocasiona sobrecusto de memória, já que o sistema operacional tem que armazenar uma tabela de ponteiros de função indexada por tipo de sensor. A troca de mensagens entre módulos ocasiona sobrecusto de execução, já que o módulo deve testar o tipo da mensagem recebida e determinar a ação apropriada em tempo de execução.

O padrão IEEE 1451, apresentado na seção 2.4.7, tem como objetivo padronizar o modo e semântica de acesso a diferentes *transdutores*, como sensores e atuadores. O padrão prevê que um transdutor seja associado a um *data sheet eletrônico do transdutor* (TEDS – *Transducer Electronic Data Sheet*). O TEDS é armazenado em memória não-volátil e contém campos que descrevem o tipo, atributos, operação e calibração do transdutor. O TEDS permite auto-descrição de transdutores, e elimina erros humanos associados com a entrada manual de parâmetros. Apesar das vantagens evidentes de uma interface padrão para transdutores de diferentes tipos, o padrão IEEE 1451, que começou a ser definido em 1997, em 2006 ainda não tem grande aplicação na indústria de transdutores. Este fato se deve, em parte, ao alto custo que seria associado à implementação de

uma lógica de auto-descrição para cada dispositivo, especialmente no caso de sensores simples e de baixo custo.

Uma interface de software ideal para dispositivos sensores deve atender exatamente aos requisitos típicos das aplicações de sensoriamento. Devem existir operações para ligar e desligar o sensor, obter recursos de hardware (e.g. conversor analógico-digital), iniciar leituras e ajustar parâmetros (e.g. taxa de amostragem). Para permitir portabilidade, estas operações devem ter semântica independente de dispositivo e, para tanto, operações como conversões de valores brutos de sensor para unidades físicas padronizadas e aplicação de fatores de calibragem devem ocorrer de forma transparente para a aplicação.

Há várias questões que dificultam a implementação de uma interface como a descrita neste trabalho: diferenças na funcionalidade, semântica de resultados, temporização, escalas de conversão, compartilhamento de recursos únicos (e.g. conversores analógico-digitais, temporizadores) por diferentes dispositivos, entre outros. Em particular, métodos de calibragem de sensores são extremamente dependentes de dispositivo e cenário de aplicação e, em geral, exigem interferência humana direta. É evidente que sensores com funcionalidades consideravelmente diferentes (e.g. um sensor de pH e um acelerômetro de dois eixos) dificilmente poderão ser abstraídos por uma interface idêntica. Por outro lado, se os sensores forem organizados em famílias de dispositivos de acordo com a sua funcionalidade (e.g. medir aceleração ou temperatura), espera-se que seja possível manter uma única interface e semântica de uso para toda a família, e que as particularidades de cada dispositivo dentro da família possam ser abstraídas por construções de software. Esta *virtualização* dos sensores não pode ter sobrecusto excessivo, para não prejudicar a taxa de amostragem do sensor, e não consumir memória excessiva na plataforma.

Este capítulo apresenta uma interface de software/hardware que é capaz de abstrair dispositivos sensores uniformemente. Definimos classes de dispositivos sensores baseado em sua finalidade (e.g. medir aceleração ou temperatura), e estabelecemos um substrato comum para cada classe. Cada dispositivo individual armazena suas propriedades e parâmetros operacionais, de maneira simular ao *data sheet eletrônico de transdutor* do padrão IEEE 1451. Uma camada fina de software adapta dispositivos individuais (e.g. converte leituras de ADC em valores contextualizados, aplica fatores de calibragem) para adequá-lo às características mínimas da sua classe de sensores. Desta forma, um termistor simples é exportado para a aplicação exatamente da mesma forma que um sensor de temperatura digital complexo.

Esta interface foi implementada para o sistema EPOS, e serve como base para as abstrações "sencientes" (`Sentient`) do sistema. Estas abstrações fornecem um ponto de acesso comum

para uso de sensores: periodicamente (de acordo com um temporizador ou interrupção do dispositivo sensor) o nodo verifica o valor medido por um sensor contra uma função configurada pelo usuário. De acordo com o resultado, uma função tratadora pode ser acionada. Desta forma, uma abstração "Alarme de Temperatura" poderia ser construída configurando uma abstração senciente para periodicamente verificar um sensor de temperatura, e se a temperatura detectada for maior do que um determinado valor, acionar um alarme através da função tratadora.

O restante do capítulo está organizado da seguinte forma: A seção 4.2.1 apresenta o projeto e implementação do subsistema de sensoriamento do EPOS. A seção 4.2.2 avalia as soluções propostas, comparando sua semântica de uso, sobrecusto e desempenho com outras soluções encontradas em sistemas operacionais para redes de sensores sem fios.

4.2.1 Projeto e Implementação

A figura 4.14 apresenta uma visão geral simplificada do subsistema de sensoriamento do EPOS. Métodos comuns a todos dispositivos de sensoriamento são definidos pela interface `Sensor_Common`. O método `get()` provê leituras para um único sensor em um único canal (i.e. habilita o dispositivo, espera os dados estarem disponíveis, lê o sensor, desabilita o dispositivo e retorna a leitura convertida em unidades físicas previamente determinadas). Os métodos `enable()`, `disable()`, `data_ready()` e `get_raw()` permitem que o sistema operacional e as aplicações realizem controle de grão fino sobre leituras de sensores (e.g. realizar leituras seqüenciais, obter dados não convertidos de sensores). O método `convert(int v)` pode ser utilizado para converter valores não processados de sensores em unidades científicas ou de engenharia. O método `calibrate()` executa um método de calibragem específico para plataforma e dispositivo sensor, que pode exigir interação com o usuário, dependendo do sensor.

Cada família de sensor pode especializar a interface `Sensor_Common` para abstrair adequadamente características específicas da família. A família `Magnetometer`, pode adicionar, por exemplo, métodos para realizar leituras em diferentes eixos de sensibilidade. A família `Thermistor`, por outro lado, provavelmente não precisará estender a interface comum básica. Cada família também define uma estrutura `Descriptor` específica, que define campos como precisão, dados para calibração e unidades físicas.

Cada dispositivo sensor implementa uma das interfaces definidas e pode fornecer métodos específicos para calibração, configuração e operação. Além disso, cada dispositivo preenche a estrutura `Descriptor` da família com valores específicos do sensor. Valores de padrão de

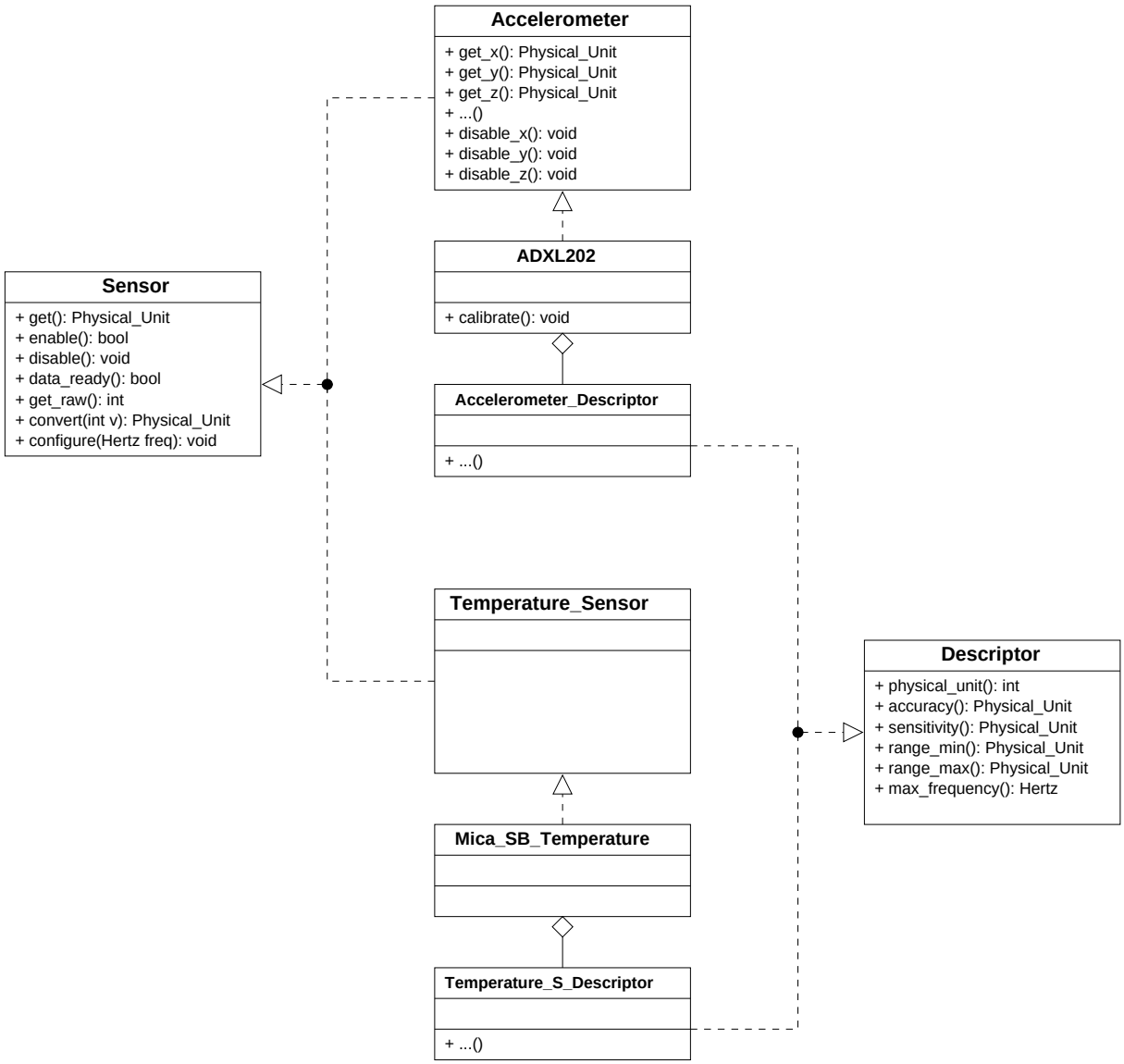


Figura 4.14: Visão Geral do Subsistema de Sensoriamento do EPOS

configuração para cada dispositivo (e.g. freqüência, ganho, etc.) são armazenados em uma estrutura de *traits de configuração*.

Sempre que o sistema operacional ou uma aplicação precisam fazer referência a um dispositivo de sensoriamento, estes podem utilizar um *dispositivo específico* (e.g. `MicaSB_Temperature`) e realizar operações específicas do dispositivo, ou utilizar a *classe do dispositivo*, e restringir-se às operações definidas por aquela classe. A estrutura de *traits de configuração* lista todos dos dispositivos de uma dada classe que estão presentes em uma dada configuração do sistema. Uma realização meta-programada estaticamente da classe do dispositivo agrega todos os dispositivos listados pelos *traits de configuração*. Esta realização é concreta quando todos os dispositivos de uma classe são do mesmo tipo, e polimórfica quando tipos diferentes de sensores existem em uma mesma classe.

4.2.1.1 Família exemplo de dispositivos: Acelerômetros

A família de dispositivos `Accelerometer` estende a interface `Sensor` básica adicionando métodos para leitura de diferentes eixos de sensibilidade (ver figura 4.14). Dispositivos específicos implementam a interface `Accelerometer` completamente ou parcialmente (e.g. um acelerômetro de dois eixos não implementará métodos para o eixo `z`). A família também define uma classe `Descriptor` (parcialmente apresentada na figura 4.15). Nesta estrutura estão definidos, por exemplo, campos que indicam a unidade física utilizada pelo mediador, sensibilidade do sensor em diferentes eixos, alcance, freqüência de operação e precisão do sensor. Uma estrutura deste tipo é armazenada em memória não-volátil para cada dispositivo da família presente no sistema. Dispositivos específicos podem ter estruturas próprias para calibração e *traits de configuração* definidos pelo usuário.

O mediador `ADXL202` é um dos membros da família `Accelerometer`. Este componente preenche uma estrutura descriptor os valores adequados ao dispositivo (por exemplo, na plataforma `Mica2`, cm/s^2 – ou `CENTIMETERS_PER_SECOND_SQ` – como unidade física, sensibilidade típica de 2 cm/s^2, alcance máximo de 19613 cm/s^2). O dispositivo define ainda uma estrutura que armazena seus fatores de calibragem em memória não-volátil, e um método para atualizar estes valores. Este método, como a maioria dos métodos de calibragem, é específico para o dispositivo, e depende de interação humana. No caso do `ADXL202`, o método envolve apontar os diferentes eixos de sensibilidade do sensor em direção à superfície da terra, medir a saída, e ajustar os fatores de calibragem de maneira a igualar a saída do sensor à aceleração da força da gravidade.

```
class Accelerometer_Descriptor {

  int physical_unit();
  Physical_Unit accuracy();
  Physical_Unit sensitivity();
  Physical_Unit range_min();
  Physical_Unit range_max();
  Hertz max_frequency();
  // ...
  struct Calibration;

};
```

(a) Descritor da Família

```
template <> struct Traits<Mica2_Accelerometer>
{
  // Concrete   (One Device)
  typedef LIST<ADXL202> SENSORS;
};
template <> struct Traits<Mica2_Accelerometer>
{
  // Concrete (Two devices of the same type)
  typedef LIST<ADXL202,ADXL202> SENSORS;
};
template <> struct Traits<Mica2_Accelerometer>
{
  // Polymorphic (Different Device Types)
  typedef LIST<ADXL202,ADXL103> SENSORS;
};
```

(b) Lista de Dispositivos Exemplo

```
template <> struct Traits<ADX202>
{
  static const unsigned long CLOCK      = Traits<Machine>::CLOCK;
  static const unsigned long IRQ        = IC::IRQ_ADC;
  static const unsigned char CHANNEL_X  = 3;
  static const unsigned char CHANNEL_Y  = 4;
  static const unsigned char PORT       = Traits<Machine>::IO::PORTC;
  static const unsigned char PORT_DIR   = Traits<Machine>::IO::DDRC;
  static const unsigned char ENABLE     = 0x10;
};
```

(c) Traits de Configuração Exemplo

Figura 4.15: Visão geral da família de Acelerômetros

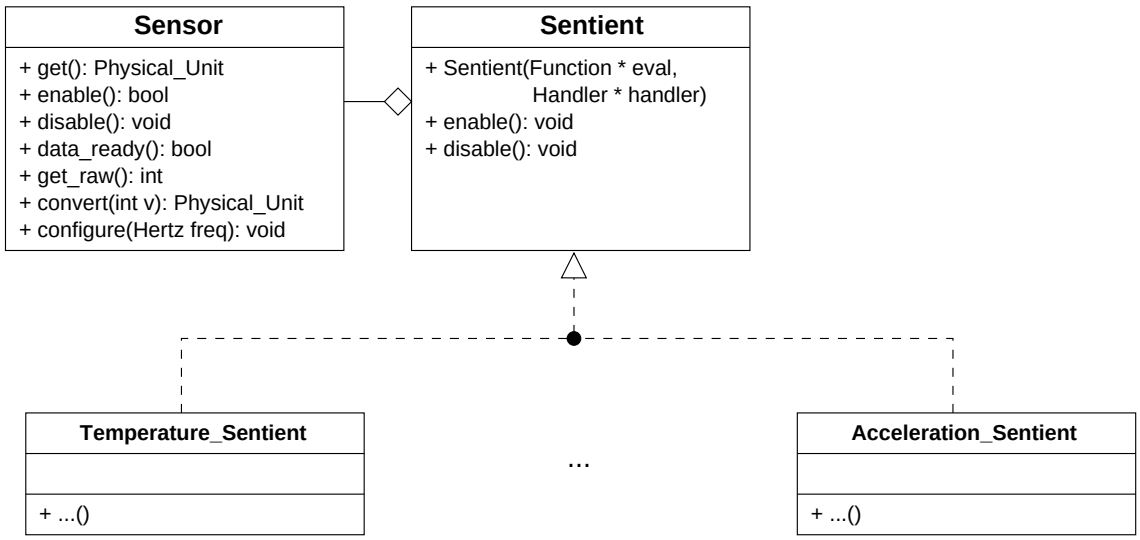


Figura 4.16: Visão Geral da Família `Sentient`

Como os fatores de calibragem são armazenados em memória não-volátil, o sensor só precisa ser recalibrado quando houverem mudanças no ambiente em que este está inserido.

A família é realizada por um *wrapper* meta-programado. Uma lista de dispositivos é definida por cada *machine* (e.g. `Mica2`). Se todos os dispositivos na lista são do mesmo tipo, a realização `Accelerometer` será concreta. Em outro caso, será polimórfica (Figura 4.15). A lista de dispositivos também define a ordem para o construtor `Accelerometer(int unit)`.

A aplicação pode utilizar tanto a realização `Accelerometer` quanto a implementação específica de um dispositivo. No primeiro caso, o programador de aplicação está restrito aos métodos definidos pela classe geral, mas obtém uma interface completamente portável. Métodos específicos de dispositivos podem ser utilizados somente através da implementação destes (e.g. `ADXL202`). Entretanto, a aplicação pode usar a classe `Accelerometer` e utilizar a estrutura `Descriptor` para obter dados específicos dos dispositivos.

4.2.1.2 Sentients

Senciente é definido como algo “que percebe pelos sentidos; que recebe impressões” [Hou01]. No EPOS, a família de abstrações `Sentient` (Figura 4.16) representa um ponto de acesso comum para uso de sensores. Tipicamente, uma aplicação de sensoriamento verifica o valor de um sensor e, de acordo com este valor, executa alguma ação. A família `Sentient` abstrai este modelo de uso, e permite que aplicações registrem funções avaliadoras e tratadoras para determinada classe de sensores.

A aplicação configura os parâmetros de sensoriamento (e.g. freqüência de amostragem do sensor) e instancia um `Sentient`, passando como parâmetro a função avaliadora e a função tratadora. A abstração `Sentient` ativa o sensor, e a cada interrupção de amostragem completa deste, executa a função avaliadora. Caso o resultado desta execução indique que uma ação deve ser tomada, o tratador registrado é acionado. De outra forma, a leitura é descartada pela abstração.

A abstração Sentient permite a construção de aplicações de sensoriamento totalmente independentes de plataforma e livres de contexto, uma vez que a aplicação lida somente com valores de dados já contextualizados pelos componentes de nível mais baixo, e nunca diretamente com os sensores. O sobrecusto de execução adicionado pela abstração é desprezível, e equivalente a uma chamada indireta de função a cada interrupção de sensor, e uma chamada indireta de função cada vez que a função avaliadora indicar que uma ação deve ser executada.

4.2.2 Avaliação e Perspectivas

Para testar a expressividade, portabilidade e sobrecusto do subsistema de sensoriamento do EPOS, foram implementadas uma série de aplicações usando diferentes sensores disponíveis para a plataforma Mica2, usando as abstrações de sensoriamento presentes nos sistemas MANTIS OS, TinyOS e EPOS. O estágio atual de desenvolvimento do sistema SOS não permitiu a realização de testes significativos com este sistema.

Em todos os testes, a aplicação de sensoriamento continuamente solicita amostragens do sensor e envia os dados obtidos pela rede. Em cada teste são medidos a taxa máxima de amostragem obtida, e o sobrecusto de memória de código e dados da abstração ou *driver* em questão. Ao final, uma avaliação qualitativa das aplicações e dados obtidos é apresentada.

A figura 4.17 apresenta a estrutura geral das aplicações de sensoriamento nos três sensores. Nas aplicação do TinyOS, MANTIS OS, e EPOS, `DemoSensorC`, `DEV_MICA2_TEMP` e `Temperature_Sensor` são respectivamente substituídos pelos *drivers* ou abstrações em teste. Para medir o sobrecusto de memória dos componentes de sensoriamento no EPOS e TinyOS, as aplicações foram primeiro compiladas com uma abstração vazia de um sensor, que representa o custo base do sistema. O tamanho total de memória da aplicação teste de cada sensor foi então diminuído deste custo base. Como a aplicação de sensoriamento não varia entre os testes, esta diferença representa o custo efetivo do componente em teste. Dada a natureza monolítica do MANTIS OS, este método não seria significativo para este sistema. Desta forma, para o sistema MANTIS OS, o custo dos componentes em teste foi aferido a partir do custo dos drivers envolvidos

na abstração do sensor em questão.

A tabela 6.1 resume as características dos sensores testados. Para todos os testes, a abstração de mais alto nível disponível no sistema foi utilizada, de maneira que, no sistema EPOS, as aplicações fazem sempre referência à família do sensor (e.g. `Magnetometer`) e não ao componente específico (e.g. HMC1002). No TinyOS e MANTIS OS, a abstração de mais alto nível disponível é o próprio componente ou *driver* do sensor.

A tabela 4.6 resume o sobrecusto dos componentes de sensoriamento testados, com o custo base de cada sistema. A tabela 4.7 apresenta a taxa máxima de amostragem obtida para cada sensor em cada um dos sistemas testados, com a taxa de amostragem máxima indicada pela documentação do sensor. Para obter este dado, o tempo de execução de 10000 leituras de cada eixo do sensor foi medido em cinco repetições. Cabe notar que, para os sensores analógicos e alguns modelos de sensores digitais, a taxa de leitura máxima obtida pode ser maior do que a taxa máxima de amostragem do sensor, uma vez que esta última representa a taxa máxima na qual o sensor pôde ser lido, e a primeira representa a freqüência máxima de reação do sensor a estímulos.

O baixo sobrecusto dos componentes de sensoriamento do EPOS são resultado do projeto, que minimiza as dependências destes componentes com o restante do sistema. No EPOS, um componente que abstrai um sensor analógico em geral depende apenas do componente que abstrai o conversor analógico-digital da plataforma e do subsistema de I/O, que por sua vez é composto por funções *inline* e meta-programadas, o que diminui substancialmente o sobrecusto, mesmo incluindo as funções de conversão de valores, que tipicamente consomem uma parcela significativa de recursos em plataformas de 8-bits. Estas mesmas características são também responsáveis pelas taxa de amostragem superiores alcançadas pelo EPOS.

Para permitir portabilidade, estas operações devem ter semântica independente de dispositivo e, para tanto, operações como conversões de valores brutos de sensor para unidades físicas padronizadas e aplicação de fatores de calibragem devem ocorrer de forma transparente para a aplicação.

Os sistemas testados não atingem o mesmo nível de abstração de sensor apresentado pelo EPOS. Na plataforma Mica2, o sistema TinyOS reflete o projeto de hardware e exporta, por exemplo, um sensor de temperatura para a aplicação como um canal de ADC (na plataforma física o sensor é analógico e está ligado ao ADC do microcontrolador). Esta dependência entre sistema operacional e hardware certamente trará implicações para a portabilidade da aplicação quando, por exemplo, ela for portada para uma plataforma na qual o sensor de temperatura é digital e está conectado diretamente a pinos de IO do microcontrolador. Ainda que a funcionalidade da aplicação

```
configuration SenseToRfm {}
implementation
{
  components Main, SenseToInt, IntToRfm, TimerC,
                   DemoSensorC as Sensor;

  Main.StdControl -> SenseToInt;
  Main.StdControl -> IntToRfm;

  SenseToInt.Timer -> TimerC.Timer[unique("Timer")];
  SenseToInt.TimerControl -> TimerC;
  SenseToInt.ADC -> Sensor;
  SenseToInt.ADCControl -> Sensor;
  SenseToInt.IntOutput -> IntToRfm;
}
```

(a) TinyOS

```
static comBuf send_pkt;

void send_thread ();

void start (void)
{
   send_pkt.size = 2;

   while(1) {
     dev_read (DEV_MICA2_TEMP, &send_pkt.data[0], 1);
     com_send(IFACE_RADIO, &send_pkt);
   }
}
```

(b) MANTIS OS

```
int main()
{
  int msg;
  NIC nic;
  Temperature_Sensor sensor;

  while(1) {
    msg = sensor.get();
    nic.send(NIC::BROADCAST, 0, &msg, sizeof(msg));
  }
}
```

(c) EPOS

Figura 4.17: Aplicações de Sensoriamento

Sensor	Função	Interface Utilizada	Componente		
			TinyOS	MANTIS OS	EPOS
ATMega128 ADC	ADC	Digital	`HPLADCM`	`AVR_ADC`	`ATMega128_ADC`
ADXL202	Acelerômetro	Analógica	`AccelM`	`MICA2_ACCEL`	`ADXL202`
ERT-J1VR103J	Temperatura	Analógica	`PhotoTempM`	`MICA2_TEMP`	`MicaSB_Temperature`
Fotocélula CdSe	Luz	Analógica	`PhotoTempM`	`MICA2_LIGHT`	`MicaSB_Light`
HMC1002	Magnetômetro	Analógica	`MagM`	`MICA2_MAGNET`	`HMC1002`
SHT11	Umidade	Digital	`TempHumM`	—	`SHT11`

Tabela 4.5: Sensores Testados

Sensor	Sobrecusto em bytes					
	TinyOS		MANTIS OS		EPOS	
	Código	Dados	Código	Dados	Código	Dados
Custo base	10188	455	25500	596	7046	213
ATMega128 ADC	550	4	538	9	64	3
ADXL202	722	4	936	10	266	9
ERT-J1VR103J	1366	12	1050	11	1064	3
Fotocélula CdSe	1366	12	1050	11	1064	3
HMC1002	748	7	910	10	246	9
SHT11	1984	23	—	—	3206	8

Tabela 4.6: Sobrecusto dos componentes de sensoriamento

Sensor	Taxa de amostragem em Hertz			
	TinyOS	MANTIS OS	EPOS	Hardware
ATMega128 ADC	8084	3685	24597	147456
ADXL202	7657	3401	21711	2000
ERT-J1VR103J	5766	3107	10999	N/D
Fotocélula CdSe	6009	3117	11121	N/D
HMC1002	7494	3408	23024	10
SHT11	6	—	10	2

Tabela 4.7: Taxa máxima de amostragem obtida

permaneça a mesma, esta deverá ser alterada levando em conta os detalhes da plataforma de hardware na qual será executada.

Um problema semelhante acontece na implementação para o sistema operacional MANTIS OS. A leitura do sensor é feita através de uma função para leitura de dispositivos, que toma como parâmetro o dispositivo físico que a aplicação deseja ler. Evidentemente o modelo físico de sensor varia de plataforma para plataforma, e a aplicação não será portável entre plataformas diferentes, mesmo quando mantém a mesma funcionalidade. Este problema poderia ser parcialmente resolvido no Mantis através de mecanismos de substituição textual que indicassem que, por exemplo, na plataforma Mica, o símbolo `Temperature_Sensor` denota `DEV_MICA2_TEMP`. Esta continuaria, entretanto, sendo uma solução bastante deselegante e ineficiente, tendo em vista que o método `dev_read` agrega código para leitura de todos os dispositivos disponíveis na plataforma, mesmo quando alguns destes não são utilizados.

A implementação para o sistema EPOS não apresenta nenhuma dependência para com o hardware de destino, a não ser a exigência de que este possua um sensor do tipo utilizado disponível, sendo perfeitamente portável entre plataformas que satisfaçam este requisito. A seleção dos mediadores de hardware adequados é feita pelo *framework* do sistema, levando em consideração as interfaces utilizadas pela aplicação e a plataforma alvo indicada pelo programador.

Capítulo 5

Conclusão e trabalhos futuros

Este trabalho apresentou um estudo das tecnologias e aplicações de redes de sensores sem fios e, a partir deste, levantou uma série de requisitos de sistema operacional. Confrontando estes requisitos com as características dos diversos projetos de pesquisa que se propuseram a tratar o problema de suporte de sistema para redes de sensores sem fios, constatou-se que a maioria deles falha em tratar principalmente dois dos requisitos levantados: abstração unificada e eficiente de hardware de sensoriamento e configuração transparente do canal de comunicação.

No TinyOS, o sistema operacional mais utilizado para redes de sensores sem fios na atualidade, não há abstrações independentes de plataforma específicas para dispositivos sensores além da interface de canal de ADC. Isto faz com que as aplicações tenham que completar a funcionalidade dos *drivers* de sensores, comprometendo a portabilidade das mesmas. O MANTIS OS, um sistema inspirado em UNIX para redes de sensores sem fios, utiliza funções em estilo POSIX para abstração de hardware. Cada função recebe como parâmetro o dispositivo a ser tratado, e uma tabela de ponteiros para funções redireciona em tempo de execução as chamadas gerais às chamadas específicas. Os parâmetros das funções são específicos para cada dispositivo, e cada *driver* de sensor tem semântica específica. O SOS, um sistema dinâmicamente reconfigurável para redes de sensores, utiliza módulos carregáveis para abstrair dispositivos de hardware. Através destes, um sensor analógico *conecta-se* a um canal de ADC e registra um tipo de sensor (e.g. `PHOTO`). Quando a aplicação requisita dados de um tipo de sensor, o núcleo do sistema envia o pedido para o *driver* registrado e recebe a leitura de ADC apropriada. Neste sistema, a abstração do subsistema de sensoriamento é independente de plataforma, mas o registro de *drivers* e a troca de mensagens entre módulos no SOS ocasionam sobrecusto de execução incompatível com as características de hardware dos nodos de redes de sensores.

No que diz respeito à configuração do canal de comunicação, a implementação do protocolo de controle de acesso ao meio B-MAC para o TinyOS permite configuração dinâmica e estática dos parâmetros básicos de comunicação (e.g. frequência, potência de transmissão). A configuração dinâmica implica na inclusão de algorítmos complexos para cálculos de valores de registradores de hardware. A configuração estática depende de parâmetros específicos de compilação ou interferência direta do usuário no sistema. No mesmo sistema, o protocolo S-MAC apresenta um projeto monilítico e pouco configurável, otimizado para determinadas cargas de trabalho. É possível trocar uma pilha de comunicação por outra mas, dadas as diferenças entre serviços e interfaces das diferentes soluções, aplicações projetadas com uma solução dificilmente serão diretamente portáveis para outra. Além disto, este tipo de configuração exige interferência do usuário na estrutura do sistema. Os sistemas MANTIS OS e SOS, apesar de permitirem o uso de diferentes implementações de protocolos, não apresentam preocupação específica com a configuração do canal de comunicação, além dos parâmetros básicos de freqüência, modulação e potência de transmissão.

A principal contribuição desta dissertação é o projeto e implementação de um ambiente de suporte à execução de aplicações em redes de sensores sem fios, baseado no sistema operacional EPOS. O desenvolvimento deste ambiente incluiu um porte do sistema EPOS para a plataforma de sensoriamento Mica2, o projeto e implementação do protocolo de controle de acesso ao meio C-MAC (*Configurable MAC*) e um sistema de aquisição de dados de sensores. O projeto e implementação modular do protocolo C-MAC permitem que aplicações configurem o canal de comunicação de acordo com suas necessidades. O protocolo agrega diferentes serviços (e.g. sincronização, detecção de dados, sinais de confirmação, contenção, envio e recepção), implementados sob diferentes estratégias. Aplicações podem configurar diferentes parâmetros de configuração em tempo de compilação e execução. Testes do C-MAC no EPOS apresentaram resultados de vazão e taxa de recepção na rede equivalentes ou melhores do que o protocolo B-MAC no TinyOS, quando os dois protocolos foram configurados de maneira idêntica. O sistema de aquisição de dados de sensor desenvolvido é capaz de abstrair famílias de dispositivos sensores de maneira uniforme, sem ocasionar sobrecusto excessivo, e apresenta vantagens significativas com relação a outras soluções encontradas em outros sistemas operacionais para redes de sensores.

Concomitante à fase final deste trabalho deu-se o desenvolvimento do sistema TinyOS 2.0, que apresenta um novo projeto para o sistema e incorpora novas soluções para abstração de hardware e, em particular, um novo subsistema de sensoriamento, que apresenta uma interface independente de plataforma similar à apresentada neste trabalho. O sistema MANTIS OS, por sua vez,

iniciou a migração para um novo sistema de compilação, que permite configuração específica para aplicações individuais, diminuindo a quantidade de código objeto redundante às aplicações. Estas novas direções de desenvolvimento de ambos sistemas estão alinhadas aos princípios descritos neste trabalho, e corroboram o projeto apresentado nesta dissertação.

As próximas etapas de desenvolvimento do C-MAC deverão permitir um grau maior de configuração, incluindo possibilidade de sincronização global na rede, e período ativo adaptativo. O C-MAC deverá também ser integrado à estruturas de roteamento no EPOS. A implementação do sistema de localização HECOPS [RF06a, RF06b] utiliza os sistemas de comunicação e sensoriamento desenvolvidos nesta dissertação para estimar as posições de nodos em uma rede de sensores sem fios, através da força de sinal obtida na comunicação entre os nodos. O sistema de gerencia do consumo de energia proposto por Hoeller [HWF06b, HWdO^{+}06, HWF06a] adiciona métodos para controle de energia aos componentes apresentados nesta dissertação permitindo, por exemplo, desligar ou colocar em modo de baixo consumo de energia um componente ou subsistema.

Referências Bibliográficas

[ABC+03] H. Abrach, S. Bhatti, J. Carlson, H. Dai, J. Rose, A. Sheth, B. Shucker, J. Deng, and R. Han. Mantis: System support for multimodal networks of in-situ sensors. In *2nd ACM International Workshop on Wireless Sensor Networks and Applications*, pages 50 – 59, San Diego, CA, 2003.

[Ana05] Analog Devices. *Low-Cost +/- 2 g Dual-Axis Accelerometer with Duty Cycle Output (ADXL202E) Datasheet*. Analog Devices, 2005.

[ASD+06] Muneeb Ali, Umar Saif, Adam Dunkels, Thiemo Voigt, Kay Romer, Koen Langendoen, Joseph Polastre, and Zartash Afzal Uzmi. Medium access control issues in sensor networks. *SIGCOMM Comput. Commun. Rev.*, 36(2):33–36, 2006.

[ASSC02] Ian F. Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. A survey on sensor networks. *IEEE Communications Magazine*, 40(8):102–114, August 2002.

[Atm04] Atmel Corporation. *ATMega128L Datasheet*. San Jose, CA, Nov 2004.

[Atm06] Atmel Corporation. AVR 8-Bit RISC. http://www.atmel.com/products/AVR/, 2006.

[BBD+02] Rimon Barr, John C. Bicket, Daniel S. Dantas, Bowei Du, T. W. Danny Kim, Bing Zhou, and Emin Gun Sirer. On the need for system-level support for ad hoc and sensor networks. *SIGOPS Oper. Syst. Rev.*, 36(2):1–5, 2002.

[BCD+05] Shah Bhatti, James Carlson, Hui Dai, Jing Deng, Jeff Rose, Anmol Sheth, Brian Shucker, Charles Gruenwald, Adam Torgerson, and Richard Han. MANTIS OS: An embedded multithreaded operating system for wireless micro sensor platforms. *MONET*, 10(4):563–579, 2005.

[BKM+04] Jan Beutel, Oliver Kasten, Friedemann Mattern, Kay Römer, Frank Siegemund, and Lothar Thiele. Prototyping wireless sensor network applications with btnodes. In *1st European Workshop on Wireless Sensor Networks (EWSN)*, pages 323–338, Berlin, Germany, January 2004. Springer-Verlag.

[Blu06] Bluetooth SIG. Specification of the bluetooth system. Technical report, Bluetooth SIG, 2006.

[Boo94] Grady Booch. *Object-Oriented Analysis and Design with Applications*. Addison-Wesley, 2nd edition, 1994.

[BYAH06] Michael Buettner, Gary Yee, Eric Anderson, and Richard Han. X-MAC: A short preamble mac protocol for duty-cycled wireless sensor networks. Technical Report CU-CS-1008-06, University of Colorado, Boulder, Colorado, USA, May 2006.

[CBBD98] Tim Cummins, Eamonn Byrne, Dara Brannick, and Dennis A. Dempsey. An IEEE 1451 standard transducer interface chip with 12-b ADC, two 12-b DAC's, 10-kb Flash EEPROM, and 8-bit microcontroller. *IEEE Journal of Solid-State Circuits*, 33(12):2112–2120, December 1998.

[CEM01] Charles Chien, Igor Elgorriaga, and Charles McConaghy. Low-power direct-sequence spread-spectrum modem architecture for distributed wireless sensor networks. In *ISLPED '01: Proceedings of the 2001 international symposium on Low power electronics and design*, pages 251–254, New York, NY, USA, 2001. ACM Press.

[Chi04a] Chipcon. *CC1000 Single Chip Very Low Power RF Transceiver Datasheet*. Oslo, Norway, 2004.

[Chi04b] Chipcon. *CC2420 2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver Datasheet*. Oslo, Norway, 2004.

[CHW98] James Coplien, Daniel Hoffman, and David Weiss. Commonality and Variability in Software Engineering. *IEEE Software*, 15(6):37–45, November 1998.

[Cro05a] Crossbow Technology. *MPR/MIB Mote Hardware User's Manual*. Crossbow Technology, Inc, San Jose, CA, September 2005.

[Cro05b] Crossbow Technology. *MTS/MDA Sensor and Data Acquisition Board User's Manual*. Crossbow Technology, Inc, San Jose, CA, April 2005.

[dAVV+04] Vinícius Coelho de Almeida, Luiz Filipe Menezes Vieira, Breno Augusto Dias Vitorino, Marcos Augusto Menezes Vieira, José Augusto Nacif, Antônio Otávio Fernandes, Diágenes Cecílio da Silva Jr., and Claudionor J. N. Coelho Jr. Sistema operacional yatos para redes de sensores sem fio. In *Proceedings of the 1st Operating Systems Workshop (WSO'04)*, pages 176–176, Salvador, BA, Brazil, August 2004. Tec Art. ISBN 85-88442-9.

[DGV04] Adam Dunkels, Björn Grönvall, and Thiemo Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *Proceedings of Proceedings of the First IEEE Workshop on Embedded Networked Sensors*, Tampa, Florida, USA, 2004.

[DHCC06] Prabal K. Dutta, Jonathan W. Hui, David C. Chu, and David E. Culler. Securing the deluge network programming system. In *IPSN '06: Proceedings of the fifth international conference on Information processing in sensor networks*, pages 326–333, New York, NY, USA, 2006. ACM Press.

[DL03] T. van Dam and K. Langendoen. An adaptive energy-efficient MAC protocol for wireless sensor networks. pages 171–180, Los Angeles, CA, November 2003.

[dS05] Thiago Robert Claudino dos Santos. Um Sistema de Comunicação Orientado à Aplicação. Master's thesis, Federal University of Santa Catarina, Florianpolis, 2005. M.Sc. Thesis.

[EH02] A. El-Hoiydi. Aloha with preamble sampling for sporadic traffic in ad hoc wireless sensor networks. In *IEEE International Conference on Communications (ICC)*, New York, April 2002.

[Emb06] Embedded Linux/Microcontroller Project. uClinux. http://www.uclinux.org/, 2006.

[ETH05] ETH Zurich. *BTnode rev3 Hardware Reference*. Eidgenössische Technische Hochschule Zürich, Zurich, Switzerland, April 2005.

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

[Frö06] Antônio Augusto Fröhlich. On operating systems for reconfigurable computing. Technical Report, Software/Hardware Integration Laboratory, 2006.

[FSP00] Antônio Augusto Fröhlich and Wolfgang Schröder-Preikschat. Operating Systems: are we finally ready to move forward after 30 years of stagnation? In *9th International Conference on Architectural Support for Programming Languages and Operating Systems Wild and Crazy Ideas Session*, page 1, Cambridge, U.S.A., November 2000.

[FSP01a] Antônio Augusto Fröhlich and Wolfgang Schröder-Preikschat. Component-Based Communication Support for Parallel Applications Running on Workstation Clusters. In *Fourth International Workshop on Advanced Parallel Processing Technologies*, pages 25–33, Ilmenau, Germany, September 2001.

[FSP01b] Antônio Augusto Fröhlich and Wolfgang Schröder-Preikschat. On Component-Based Communication Systems for Clusters of Workstations. In *First IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 640–645, Brisbane, Australia, May 2001.

[FTSP00] Antônio Augusto Fröhlich, Gilles P. Tientcheu, and Wolfgang Schröder-Preikschat. EPOS and Myrinet: Effective Communication Support for Parallel Applications Running on Clusters of Commodity Workstations. In *8th International Conference on High Performance Computing and Networking Europe*, volume 1823 of *Lecture Notes in Computer Science*, pages 417–426, Amsterdam, The Netherlands, May 2000. Springer.

[GDS+03] N. Golmie, R. E. Van Dyck, A. Soltanian, A. Tonnerre, and O. Rébala. Interference evaluation of Bluetooth and IEEE 802.11b systems. *Wirel. Netw.*, 9(3):201–211, 2003.

[GLAT99] J. J. Garcia-Luna-Aceves and Asimakis Tzamaloukas. Reversing the collision-avoidance handshake in wireless networks. In *MobiCom '99: Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, pages 120–131, New York, NY, USA, 1999. ACM Press.

[GLAT02] J. J. Garcia-Luna-Aceves and Asimakis Tzamaloukas. Receiver-initiated collision avoidance in wireless networks. *Wirel. Netw.*, 8(2/3):249–263, 2002.

[GLCB03] David Gay, Philip Levis, David Culler, and Eric Brew. *nesC Language Reference Manual*, 2003.

[GLvB+03] David Gay, Phil Levis, Rob von Behren, Matt Welsh, Eric Brewer, and David Culler. The nesc language: A holistic approach to networked embedded systems. In *Proceedings of Programming Language Design and Implementation*, San Diego, CA, June 2003.

[HC04] Jonathan W. Hui and David Culler. The dynamic behavior of a data dissemination protocol for network programming at scale. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 81–94, New York, NY, USA, 2004. ACM Press.

[HDJH04] T.J. Hofmeijer, S.O. Dulman, P.G. Jansen, and P.J.M. Havinga. Ambientrt - real time system software support for data centric sensor networks. In *2nd Int. Conf. on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), Melbourne, Australia*, pages 61–66, Los Alamitos, California, December 2004. IEEE Computer Society Press.

[HH04] L. van Hoesel and P. Havinga. A lightweight medium access protocol (LMAC) for wireless sensor networks. Tokyo, Japan, June 2004.

[Hil92] Dan Hildebrand. An architectural overview of QNX. In *Proceedings of the Workshop on Micro-kernels and Other Kernel Architectures*, pages 113–126, Berkeley, CA, USA, 1992. USENIX Association.

[Hil03] Jason Hill. *System Architecture for Wireless Sensor Networks*. PhD thesis, Univerisy of California, Berkeley, 2003.

[HKS+05] Chih-Chieh Han, Ram Kumar, Roy Shea, Eddie Kohler, and Mani Srivastava. A dynamic operating system for sensor nodes. In *MobiSys '05: Proceedings of the 3rd international conference on Mobile systems, applications, and services*, pages 163–176, New York, NY, USA, 2005. ACM Press.

[Hon05] Honeywell Solid State Electronics Center. *HMC1001/1002 1- and 2-Axis Magnetic Sensors Datasheet*. Honeywell, Plymouth, MN, 2005.

[Hou01] Antônio Houaiss. *Dicionário Houaiss da Língua Portuguesa*. Objetiva, 2001.

[HPH+05] Vlado Handziski, Joseph Polastre, Jan-Hinrich Hauer, Cory Sharp, and Adam Woliszand David Culler. Flexible hardware abstraction for wireless sensor networks. In *Proceedings of the Second European Workshop on Wireless Sensor Networks (EWSN '05)*, Istambul, Turkey, 2005.

[HSW+00] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister. System architecture directions for networked sensors. In *Proceedings of the ninth international conference on Architectural support for programming languages and operating systems*, pages 93–104, Cambridge, Massachusetts, United States, 2000.

[HWdO+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*, Campo Grande, Brazil, July 2006.

[HWF06a] Arliones Stevert Hoeller Junior, 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*, Braga, Portugal, October 2006.

[HWF06b] Arliones Stevert Hoeller Junior, Lucas Francisco Wanner, and Antônio Augusto Fröhlich. Um Método Hierárquico para Gerenciamento do Consumo de Energia em Sistemas Profundamente Embarcados. In *32th Latin-American Conference on Informatics*, Santiago, Chile, August 2006.

[Int02] Intersil. *Prism Driver Programmer's Manual*. Intersil, February 2002.

[ISO81] International Organization for Standardization. *Open Systems Interconnection - Basic Reference Model*, August 1981. ISO/TC 97/SC 16 N 719.

[ITU05] International Telecommunication Union. *Internet indicators: Hosts, Users and Number of PCs*, 2005.

[KAH+04] Ralph Kling, Robert Adler, Jonathan Huang, Vincent Hummel, and Lama Nachman. Intel mote: using bluetooth in sensor networks. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 318–318, New York, NY, USA, 2004. ACM Press.

[KKP99] J. M. Kahn, R. H. Katz, and K. S. J. Pister. Next century challenges: Mobile networking for ”smart dust”. In *International Conference on Mobile Computing and Networking (MOBICOM)*, pages 271–278, Seattle, Washington, USA, 1999.

[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.

[LBV06] K.G. Langendoen, A. Baggio, and O.W. Visser. Murphy loves potatoes: Experiences from a pilot sensor network deployment in precision agriculture. In *14th Int. Workshop on Parallel and Distributed Real-Time Systems (WPDRTS)*, April 2006.

[LC02] Philip Levis and David Culler. Mate: A tiny virtual machine for sensor networks. In *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, San Jose, California, October 2002.

[Lee00] Kang Lee. Ieee 1451: A standard in support of smart transducer networking. In *Proceedings of the IEEE Instrumentation and Measurement Technology Conference*, pages 525–528, Baltimore, MD, 2000.

[LGC05] Philip Levis, David Gay, and David Culler. Active sensor networks. In *Proceedings of the Second USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI 2005)*, May 2005.

[LH05] K. Langendoen and G. Halkes. *Embedded Systems Handbook*, chapter Energy-Efficient Medium Access Control. CRC press, 2005.

[LKR04] G. Lu, B. Krishnamachari, and C. Raghavendra. An adaptive energy-efficient and low-latency MAC for data gathering in sensor networks. In *Int. Workshop on Algorithms for Wireless, Mobile, Ad Hoc and Sensor Networks (WMAN)*, Santa Fe, NM, April 2004.

[LMG+04] Philip Levis, Sam Madden, David Gay, Joseph Polastre, Robert Szewczyk, Alec Woo, Eric Brewer, and David Culler. The emergence of networking abstractions and techniques in tinyOS. In *First Symposium on networked system design and implementation (NSDI04)*, pages 1–14, San Francisco, California, USA, 2004.

[LN79] Hugh C. Lauer and Roger M. Needham. On the duality of operating system structures. *SIGOPS Oper. Syst. Rev.*, 13(2):3–19, 1979.

[LSPS05] 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.

[MAN06] MANTIS Project. MANTIS. http://mantis.cs.colorado.edu/, 2006.

[Mau06] Alessandro Barreiros Maurici. *Suporte a redes CAN para Aplicações Emabarcadas*. B.sc. thesis, Federal University of Santa Catarina, Florianpolis, 2006.

[MB04] S. Mahlknecht and M. Boeck. CSMA-MPS: A minimum preamble sampling MAC protocol for low power wireless sensor networks. In *IEEE Int. Workshop on Factory Communication Systems*, pages 73–80, Vienna, Austria, September 2004.

[MCP+02] Alan Mainwaring, David Culler, Joseph Polastre, Robert Szewczyk, and John Anderson. Wireless sensor networks for habitat monitoring. In *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 88–97, Atlanta, Georgia, USA, September 2002.

[MHW+06] Hugo Marcondes, Arliones Stevert Hoeller Junior, Lucas Francisco Wanner, Rafael Luiz Cancian, Danillo Moura Santos, and Antônio Augusto Fröhlich. EPOS: Um Sistema Operacional Portável para Sistemas Profundamente Embarcados. In *Third Brazilian Workshop on Operating System*, Campo Grande, Brazil, July 2006.

[MHWF06] 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*, Prague, Czech Republic, September 2006.

[Mic06] Microsoft Corporation. WindowsCE. http://www.microsoft.com/windows/, 2006.

[Mot05] Moteiv. *Tmote Sky Datasheet*. Moteiv Corporation, San Francisco, CA, February 2005.

[Ous96] J. K. Ousterhout. Why threads are a bad idea (for most purposes). Invited talk at the 1996 Usenix Annual Technical Conference, 1996.

[Par76] David Lorge Parnas. On the Design and Development of Program Families. *IEEE Transactions on Software Engineering*, SE-2(1):1–9, March 1976.

[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.

[PFHD04] Fauze Valério Polpeta, Antônio Augusto Fröhlich, Arliones Stevert Hoeller Junior, and Tiago Stein D'Agostini. Portabilidade em Sistemas Operacionais Baseados em Componentes de Software. In *First Brazilian Workshop on Operating System*, pages 1466–1475, Salvador, Brazil, August 2004.

[PHC04] Joseph Polastre, Jason Hill, and David Culler. Versatile low power media access for wireless sensor networks. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 95–107, New York, NY, USA, 2004. ACM Press.

[PK00] G. J. Pottie and W. J. Kaiser. Wireless integrated network sensors. *Communications of the ACM*, 43(5):51–58, May 2000.

[Pol05] Joseph Polastre. *A Unifying Link Abstraction for Wireless Sensor Networks*. PhD thesis, University of California, Berkeley, 2005.

[Por13] Noah Porter, editor. *Webster's Revised Unabridged Dictionary*. G & C. Merriam Co., 1913.

[PSC05] Joseph Polastre, Robert Szewczyk, and David Culler. Telos: Enabling ultra-low power wireless research. In *The Fourth International Conference on Information Processing in Sensor Networks: Special track on Platform Tools and Design Methods for Network Embedded Sensors (IPSN/SPOTS)*, Los Angeles, California, April 2005.

[Rad01] RadiSys. *Microware OS-9 Booklet*. RadiSys, Hillsboro, OR, USA, 2001.

[RF 06] RF Monolithics Inc. *TR1000 916.50 MHz Hybrid Transceiver Datasheet*, 2006.

[RF06a] Ricardo Reghelin and Antônio Augusto Fröhlich. A Decentralized Location System for Sensor Networks Using Cooperative Calibration and Heuristics. In *9th ACM/IEEE International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, Torremolinos, Malaga, Spain., October 2006.

[RF06b] Ricardo Reghelin and Antônio Augusto Fröhlich. HECOPS: A Location System using cooperative calibration. In *3rd IEEE International Workshop on Wireless Ad-hoc and Sensor Networks*, New York, U.S.A., June 2006.

[RR05] M. Ringwald and Kay Roemer. BitMAC: a deterministic, collision-free, and robust MAC protocol for sensor networks. In *Proc. IEEE European Workshop on Wireless Sensor Networks (EWSN) 2005*, pages 57–69, Istanbul, Turkey, January 2005.

[SdMCF06] Danillo Moura Santos, Roberto de Matos, Rafael Luiz Cancian, and Antônio Augusto Fröhlich. Desenvolvimento de Sistemas Embarcados com Suporte a Tempo-real Seguindo o Projeto de Sistemas Orientados a Aplicação. In *8th Brazilian Workshop on Real-Time Systems*, pages 119–122, Curitiba, Brazil, June 2006.

[Sen05] Sensirion. *SHT1x / SHT7x Humidity and Temperature Sensor Datasheet*. Sensirion, Stäfa, Switzerland, May 2005.

[SGAP00] K. Sohrabi, J. Gao, V. Ailawadhi, and G.J. Pottie. Protocols for self-organization of a wireless sensor network. *IEEE Personal Communications*, 7(5):16–27, Oct 2000.

[Sha38] Claude Elwood Shannon. A symbolic analysis of relay and switching circuits. In *Proceedings of the American Institute of Electrical Engineers*, volume 57, pages 713–723, March 1938.

[SHE03] Thanos Stathopoulos, John Heidemann, and Deborah Estrin. A remote code update mechanism for wireless sensor networks. Technical report, November 26 2003.

[SMCF06a] Danillo Santos, Roberto Matos, Rafael Luiz Cancian, and Antônio Augusto Fröhlich. Advantages and Disadvantages of Application-Oriented System Design in Embedded Systems Design. In *4th International IEEE Conference on Industrial Informatics*, Singapore, August 2006.

[SMCF06b] Danillo Santos, Roberto Matos, Rafael Luiz Cancian, and Antônio Augusto Fröhlich. Application-Oriented System Design as an Embedded Systems Deve-

lopment Strategy: a critical analisys. In *11th IEEE International Conference on Emerging Technologies and Factory Automation*, Prague, Czech Republic, September 2006.

[Soc03] IEEE Computer Society. Ieee standard 802 part 15.4: Wireless medium access control (mac) and physical layer (phy) specifications for low-rate wireless personal area networks (lr-wpans). Technical report, The Institute of Electrical and Electronics Engineers, 2003.

[SOS06] SOS Project. SOS. http://nesl.ee.ucla.edu/projects/sos/, 2006.

[Sta05] Pascal Stang. Bootloaders for the Atmel AVR series. Technical report, http://hubbard.engr.scu.edu/embedded/avr/bootloader/, September 2005.

[TAO05] TAOS. *TSL2550 Ambient Light Sensor with SMBus Interface Datasheet*. Texas Advanced Optoelectronic Solutions, Plano, Texas, 2005.

[Ten00] David Tennenhouse. Proactive Computing. *Communications of the ACM*, 43(5):43–50, May 2000.

[Tin06] TinyOS Project. TinyOS. http://www.tinyos.net/, 2006.

[Tur37] Alan Turing. On computable numbers, with an application to the entscheidungsproblem. In *Proceedings of the London Mathematical Society*, volume 42, London, England, 1937.

[UDT02] UDT Sensors. *Photodiode Characteristics*. UDT Sensors, Inc, Hawthorne, CA, 2002.

[vBCB03] J. Robert von Behren, Jeremy Condit, and Eric A. Brewer. Why events are a bad idea (for high-concurrency servers). In *HotOS*, pages 19–24, 2003.

[VdCdM03] Marcos Augusto Menezes Vieira, Diógenes Cecílio da Silva Jr., Claudionor J. N. Coelho Jr., and José M. da Mata. Survey on wireless sensor network devices. In *Proceedings of the 9th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'03)*, Lisbon, Portugal, September 2003.

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

[Wan03] Lucas Francisco Wanner. *The EPOS System Supporting Wireless Sensor Networks Applications*. B.sc. thesis, Federal University of Santa Catarina, Florianópolis, 2003.

[WC01] Alec Woo and David E. Culler. A transmission control scheme for media access in sensor networks. In *Proceedings of the 7th annual international conference on Mobile computing and networking*, pages 221–235, Rome, Italy, 2001.

[WHPF05a] Lucas Francisco Wanner, Arliones Stevert Hoeller Junior, Fauze Valério Polpeta, and Antônio Augusto Fröhlich. Operating System Support for Handling Heterogeneity in Wireless Sensor Networks. In *10th IEEE International Conference on Emerging Technologies and Factory Automation*, Catania, Italy, September 2005.

[WHPF05b] Lucas Francisco Wanner, Arliones Stevert Hoeller Junior, Fauze Valério Polpeta, and Antônio Augusto Fröhlich. Suporte de Sistema Operacional para Tratamento de Heterogeneidade em Redes de Sensores sem Fios. In *Second Brazilian Workshop on Operating System*, pages 3110–3120, São Leopoldo, Brazil, July 2005.

[Win03] Wind River Systems. *VxWORKS 5.x Datasheet*. Wind River, Alameda, CA, USA, 2003.

[YH04] Wei Ye and John Heidemann. Medium access control in wireless sensor networks. pages 73–91, 2004.

[YHE02] W. Ye, J. Heidemann, and D. Estrin. An energy-efficient MAC protocol for wireless sensor networks. In *21st Conference of the IEEE Computer and Communications Societies (INFOCOM)*, volume 3, pages 1567–1576, June 2002.

[ZMS05] Frank A. Zdarsky, Ivan Martinovic, and Jens B. Schmitt. On lower bounds for mac layer contention in csma/ca-based wireless networks. In *DIALM-POMC '05: Proceedings of the 2005 joint workshop on Foundations of mobile computing*, pages 8–16, New York, NY, USA, 2005. ACM Press.