Suporte de Sistema Operacional para Redes de Sensores

Lucas Francisco Wanner[1] **e Antônio Augusto Fröhlich**[1]

[1]Laboratório de Integração em Software e Hardware
Universidade Federal de Santa Catarina
88040-900 – Florianópolis, SC, Brasil

{lucas,guto}@lisha.ufsc.br

Abstract. *Several research projects have aimed at solving the problem of system support for sensor networks. However, most of them either fail in correctly dealing with sensor network application requirements, or present prohibitive overhead. This work presents the project and implementation of a runtime support environment for wireless sensor network applications based on the* EPOS *operating system. This environment provides applications with hardware support, configurable communication, power management, and a data acquisition system, and presents significant advantages when compared to other sensor network operating systems.*

1. Introdução

A idéia de uma rede de sensores auto-gerenciada, composta por 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 [Hofmeijer et al. 2004, de Almeida et al. 2004, Barr et al. 2002, Dunkels et al. 2004, Hill et al. 2000, Abrach et al. 2003, Han et al. 2005] se propuseram a tratar o problema de suporte de sistema para redes de sensores sem fios.

Entrentanto, este artigo mostrará que a maioria deles falha em tratar adequadamente os requisitos das aplicações, ou apresenta *overhead* excessivo.

O sistema EPOS [Fröhlich 2001, Marcondes et al. 2006] 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, MIPS e AVR [Marcondes et al. 2006].

Este trabalho apresenta o projeto e implementação de um sistema de suporte a execução de aplicações em redes de sensores sem fios baseado no sistema EPOS. Este ambiente inclui suporte a plataformas de hardware, serviços de gerência de energia, comunicação configurável através do protocolo C-MAC (*Configurable MAC*), e um sistema de aquisição de dados de sensores. Análises comparativas mostram vantagens deste ambiente com relação a outros sistemas operacionais para redes de sensores.

2. Sistemas Operacionais para Redes de Sensores

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, aplicações e pesquisa atuais [Hill et al. 2000], é possível listar uma série de requisitos de *sistemas operacionais* para redes de sensores sem fios:

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 operacional 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 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, uma vez que é nesta camada de rede que encontram-se as maiores oportunidades de impacto no desempenho e consumo de energia.

Operar com recursos limitados 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, QNX, OS-9, WinCE e μClinux 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. Entre os sistemas operacionais desenvolvidos especificamente para redes de sensores sem fios, cabe citar AmbientRT [Hofmeijer et al. 2004], YaTOS [de Almeida et al. 2004], MagnetOS [Barr et al. 2002] e Contiki [Dunkels et al. 2004]. Entretanto, os mais proeminentes, e que mais se aproximam dos requisitos levantados acima são TinyOS [Hill et al. 2000], MANTIS OS [Abrach et al. 2003] e SOS [Han et al. 2005].

2.1. TinyOS

TinyOS [Hill et al. 2000] é um sistema operacional baseado em eventos, organizado como uma coleção de componentes de software. O TinyOS apresenta um modelo de concorrência baseado em tarefas que executam até sua completude, e que só podem ser preemptadas por interrupções. Um modelo tradicional, baseado em *threads* com pilhas próprias exige que cada *thread* reserve espaço para seu contexto de execução na memória. Ao não permitir concorrência entre tarefas, o TinyOS reduz boa parte deste sobrecusto, mas perde as características de um modelo *multithread* tradicional. A restrição de concorrência limita também a capacidade do sistema sistema em tratar tempo real.

A gerência do consumo de energia no TinyOS é implementada por seu escalonador. Quando a fila de tarefas do escalonador está vazia, este 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 (inclusive a desativação de periféirocos) são deixados a cargo da aplicação.

O TinyOS apresenta um projeto de três camadas para abstração de hardware [Handziski et al. 2005]. 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`. 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 ou bibliotecas tenham que completar a funcionalidade dos *drivers* de sensores, o que pode comprometer a portabilidade e uso eficiente de recursos.

A pilha de comunicação do TinyOS é baseada no protocolo B-MAC [Polastre et al. 2004], que é implementado em camadas. Uma camada de controle de hardware permite ajuste de parâmetros básicos da comunicação (e.g. freqüência e potência de transmissão). Por outro lado, adaptações na camada de lógica do protocolo (e.g. ajuste do tamanho do período ativo, uso de algorítmo de *clear channel assessment* e uso de pacotes de confirmação) exigem adaptações do usuário no código fonte do protocolo.

O modelo de componentes e concorrência do TinyOS torna o sistema fiel ao seu nome (*tiny* - minúsculo em português).O sistema 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 [Hill et al. 2000] e modelo de abstração de hardware excessivamente dependente de plataforma fazem com que as aplicações possam ter que completar a funcionalidade do sistema operacional.

2.2. MANTIS OS

O sistema operacional MANTIS (*Multimodal Networks of In-situ Sensors*) [Abrach et al. 2003] é um sistema *multithread* com uma API inspirada em POSIX, adaptada às necessidades das redes de sensores sem fios. 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.

O escalonador do MANTIS OS fornece um subconjunto do pacotes de *threads* POSIX, com escalonamento *round-robin* com prioridades. 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. Estas faz uso de uma chamada `sleep`, que controla apenas o microcontrolador principal do sistema, não havendo controle específico para os periféricos além do fornecido pelos *drivers*.

O MANTIS OS utiliza uma camada de abstração de hardware monolítica (HAL)

em estilo POSIX. Cada função da HAL 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. Não há no MANTIS OS abstrações unificadas para sensoriamento (cada *driver* de sensor tem semântica específica).

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. Abaixo da API de comunicação, o MANTIS OS utiliza *drivers* de dispositivo tradicionais para tratar de suporte de hardware e implementar suporte de MAC. A abordagem monolítica da pilha de comunicações do MANTIS OS pode ocasionar um sobrecusto grande de desempenho e tamanho de código no sistema. Por outro lado, a vantagem de um único ponto de entrada no sistema de comunicação é diminuída pelas chamadas específicas aos métodos de controle de hardware. Como a semântica e parâmetros desses métodos varia entre dispositivos, 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).

O modelo de escalonamento baseado em *threads* do MANTIS OS, bem como sua camada de abstração de hardware monolítica, fazem com que o sistema tenha requisitos maiores do que um modelo mais simples, baseado em eventos. Ainda assim, mesmo com *footprint* tipicamente maior do que uma configuração equivalente do TinyOS, o sistema mostra-se adequado à execução nos protótipos atuais de nodos de redes de sensores.

2.3. SOS

SOS [Han et al. 2005] é um sistema operacional para redes de sensores reconfigurável dinamicamente. O sistema 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 e apresentam interfaces de métodos e passagem de mensagens.

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*. O sistema inclui um alocador dinâmico de memória e um *garbage collector*. Como no TinyOS e MANTIS OS o escalonador coloca o processador em modo de baixo consumo quando não há tarefas para escalonar.

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. Nesta solução, modelos diferentes de hardware com a mesma função podem ser abstraídos de forma similar, ainda que com grande sobrecusto de memória, devido ao registro dinâmico de *drivers*, e de execução, devido aos testes de mensagens necessários para determinar a ação associada a mensagem recebida em tempo de execução.

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 argumentam que este sobrecusto é aceitável para a maioria de aplicações de redes de sensores sem fios [Han et al. 2005].

3. O Sistema EPOS

O sistema EPOS (Embedded Parallel Operating System) [Fröhlich 2001, Marcondes et al. 2006] 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 do projeto de sistemas orientado à aplicação, 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. Esta seção apresenta os principais conceitos relacionados ao sistema EPOS, incluindo a metodologia do projeto de sistemas orientado à aplicação, e seus serviços específicos para redes de sensores sem fios.

3.1. Projeto de Sistemas Orientado à Aplicação

Projeto de Sistemas Orientado à Aplicação (AOSD) é uma metodologia de engenharia de domínio que expande as estratégias de análise de *características comuns* e *variabilidades* do Projeto Baseado em Famílias e Orientação a Objetos, adicionando os conceitos de identificação e separação de aspectos às fases iniciais do projeto [Fröhlich 2001]. 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:

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.

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.

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.

No 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 [Fröhlich 2001].

3.2. Organização e Projeto do Sistema

No EPOS, famílias de abstração representam abstrações tradicionais de sistema operacional. 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* 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.

Processos no EPOS gerenciados pelas abstrações `Thread` e `Task`. Cada `Thread` mantém em sua pilha o seu contexto de execução. A classe `Context` deste mediador define todos os dados a serem armazenados por um fluxo de execução em determinada arquitetura. A abstração `Alarm` é ser usada para gerar eventos que *acordam* uma `Thread` ou chamam uma função previamente registrada. A abstração `Alarm` incorpora ainda um *evento mestre*, que é utilizado para disparar o algoritmo de escalonamento de processos, quando o sistema está configurado com um escalonador ativo.

A família de abstrações `Synchronizer` 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` é inspirado no mecanismo de variáveis de condição, e permite que uma `Thread` suspenda sua execução até que determinado predicado em dados compartilhados torne-se verdadeiro.

No EPOS, detalhes de proteção e tradução de endereços, bem como alocação dinâmica de memória, são tratados pela família de mediadores `MMU`. 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`. Métodos de alocação de memória operam de forma similar à função `malloc` da libc.

O controle de entrada e saída em dispositivos periféricos no EPOS é fornecido pelos mediadores específicos do hardware em questão. O mediador `Machine` guarda informações de localização de memória de periféricos, e trata do registro dinâmico de tratadores de interrupções de maneira independente de plataforma. O mediador `IC` (Con-

trolador de Interrupções) é responsável por habilitar, desabilitar e configurar interrupções individuais. De maneira a tratar interrupções disponíveis em diferentes plataformas e contextos, 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).

3.3. Gerência do Consumo de Energia

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 [Hoeller Junior et al. 2006].

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`. 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`.

Adicionalmente, o sistema implementa algoritmos de gerência ativa de energia através do escalonador de processos. Além de colocar o processador em modo de baixo consumo de energia quando não há tarefas a escalonar, este gerente de energia utiliza contadores de acesso para determinar quando componentes periféricos estão inativos. As heurísticas para determinar quando deve-se desligar um compoenente ou coloca-lo em modo de baixo consumo são “re-plugáveis”, e perimitem às aplicação selecionarem as estratégias mais adequadas às suas necesidades Estudos de caso [Wiedenhoft et al. 2007, Hoeller Junior et al. 2006] mostraram que tanto o gerenciamento de energia dirigido pela aplicação quanto o gerente ativo do EPOS conseguiram resultados significativos de economia do consumo de energia, sem sobrecusto excessivo, e com intervenção mínima do programador de aplicação.

3.4. Sensoriamento

O EPOS fornece suporte de sensoriamento às aplicações através de uma interface de software/hardware que abstrai famílias de dispositivos de sensor de forma uniforme [Wanner et al. 2006]. O sistema define 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

Sensor	TinyOS			MANTIS OS			EPOS		
	Memória (bytes)		Taxa	Memória (bytes)		Taxa	Memória (bytes)		Taxa
	Código	Dados	(Hz)	Código	Dados	(Hz)	Código	Dados	(Hz)
Custo base	10188	455	—	25500	596	—	7046	313	—
ATMega128 ADC	550	4	8084	538	9	3685	64	3	24597
ADXL202	722	4	7657	936	10	3401	266	9	21711
ERT-J1VR103J	1366	12	5766	1050	11	3107	1064	3	10999
Fotocélula CdSe	1366	12	6009	1050	11	3117	1064	3	11121
HMC1002	748	7	7494	910	10	3408	246	9	23024
SHT11	1984	23	6	—	—	—	3206	8	10

Tabela 1. Tamanho de memória e taxa de amostragem dos componentes de sensoriamento

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.

No subsistema de sensoramento 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 calibragem específica para cada 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 preenche a estrutura `Descriptor` da família com valores específicos do sensor. Valores de padrão de 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 realizar operações específicas do dispositivo, ou utilizar a *classe do dispositivo*, e restringir-se às operações definidas por aquela classe. Uma realização meta-programada estaticamente da classe do dispositivo agrega os dispositivos disponíveis em uma configuração do sistema.

A tabela 1 apresenta o tamanho dos componentes de sensoriamento e a taxa máxima de amostragem obtida nestes sensores nos sistemas TinyOS, MANTIS OS e EPOS. O tamanho menor e taxa de amostragem maior obtidos pelo sistema EPOS são resultado direto do projeto do sistema, que minimiza as dependências entre os componentes de sensoriamento e o restante do sistema. Desta forma, um componente do EPOS que abstraia um sensor analógico normalmente terá dependências somente com o conversor analógico digital e o subsistema de entrada e saída de uma plataforma, que por sua

vez são abstraidos por funções *inline* ou operadores meta-programados. Desta forma, no EPOS uma operação de leitura de um sensor em geral traduz-se simplesmente em um pequeno conjunto de instruções necessários para fazer uma leitura do conversor ADC. Uma operação equivalente no TinyOS em geral envolve adicionamente o tratamento de uma interrupção (devido ao modelo baseado em eventos do sistema). No MANTIS OS uma leitura de sensor pode envolver a interpretação de parâmetros para determinar qual sensor deve ser lido, a leitura propriamente dita, e a propagação de resultados da leitura. O modelo do EPOS minimiza o *overhead*, mesmo considerando que o EPOS inclui métodos de conversão e calibragem que não encontram equivalentes em outros sistemas.

3.5. 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, incluindo detecção de pacotes de dados, detecção e tratamento de erros, endereçamento, filtragem de pacotes, tenham que ser implementados em software. Os protocolos de controle de acesso ao meio para redes de sensores fazem um compromisso entre desempenho (latência, vazão) por custo (consumo de energia). Comparações em diferentes cenários mostram que não há um protocolo "ideal" para todas aplicações de redes de sensores [Langendoen and Halkes 2005].

A infra-estrutura de comunicação do EPOS faz uso do C-MAC (Configurable MAC), um *protocolo configurável* de acesso ao meio para redes de sensores sem fios. O C-MAC [Wanner et al. 2007] 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 em tempo de compilação e execução.

O C-MAC utiliza um *framework* metaprogramado para construir um *kernel* de comunicação configurável. A configuração em tempo de execução permite alterar parametros básicos de comunicação (e.g. freqüência e potência de transmissão), e a configuração em tempo de compilação permite alterar amplamente a lógica do protocolo, incluindo período ativo e organização, mecanismos para evitar, detectar e tratar colisões, e uso de pacotes de confirmação. Esta gama de configuração permite ao protocolo *emular* com sobrecusto mínimo o funcionamento de outros protocolos típicos de redes de sensores [Wanner et al. 2007].

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 ela 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. Desta forma, é possível ao C-MAC *emular* a funcionalidade de outros protocolos com tamanho de memória menor e desempenho melhor. Esta característica é ilustrada pela tabela 2, que apresenta uma comparação entre o protocolo C-MAC no EPOS e o protocolo B-MAC [Polastre et al. 2005] no TinyOS, com os dois protocolos configurados de forma identica, em uma rede composta por três nodos [Wanner et al. 2007].

Sistema	Código (bytes)	Dados (bytes)	Pacotes Recebidos (%)	Taxa de dados (bps)
EPOS / C-MAC	3888	108	77,7	10251
TinyOS / B-MAC	8562	205	74,3	9130

Tabela 2. Tamanho de memória e desempenho de comunicação

3.6. Reprogramação

A estrutura de reprogramação em campo do EPOS faz uso de uma estrutura de *proxy* e *agent* disponível no sistema. Nesta estrutura, a invocação de um método de um componente da aplicação previamente configurado com suporte para atualização (através do seu Traits no sistema) do cliente passa pelo *proxy* que envia uma mensagem para o *agent*. Esta mensagem é composta pelos IDs do objeto, método e classe, na qual o Agent usa essas informações para fazer a invocação do método correspondente, associando seus valores à um tabela. Através do objeto, o Agent acessa o endereço em que aquele componente esta na memória. Após a chamada, o valor de retorno é enviado ao proxy.

Esta estrutura cria um nível de indireção entre as chamadas de método e permite um confinamento dos componentes do sistema. Toda a chamada de método dos componentes configurados com suporte a atualização passam obrigatoriamente por esta estrutura. Com isso os componentes podem ser mapeados em qualquer posição da memória. Em uma atualização, simplemeste é feita a atualização do endereço em memória do componente na tabela do Agent. Resultados preliminares mostram que esta estrutura é capaz de lidar adequadamente com reprogramação, adicionando em média um *overhead* de menos de 50 bytes de memória por componente.

4. Conclusões

Este trabalho apresentou o projeto e implementação de suporte de tempo de execução para aplicações de redes de sensores sem fios baseado no sistema EPOS. O sistema EPOS fornece um ambiente de programação amplamente portável para aplicações embarcadas. Seu sistema de gerência do consumo de energia fornece controle autônomo ou dirigido pela aplicação, permitindo que partes do sistema, ou que o sistema como um todo, migre para níveis mais baixos de consumo de energia. Seu subsistema de sensoriamento permite que aplicações coletem dados de sensor sem ter que lidar com detalhes específicos de hardware, e sem sobrecusto excessivo. A implementação e projeto do protocolo C-MAC permitem *emular* funcionalidade de diferentes protocolos de comunicação de acordo com as necessidades das aplicações, com tamanho de memória menor e desempenho competitivos. Finalmente, o mecanismo de reprogramação do EPOS permite alterar funcionalidade de componentes *online*, com sobrecusto mínimo.

Referências

Abrach, H., et al. (2003). Mantis: System support for multimodal networks of in-situ sensors. In *2nd ACM WWSNA*, pages 50 – 59, San Diego, CA.

Barr, R., et al. (2002). On the need for system-level support for ad hoc and sensor networks. *SIGOPS Oper. Syst. Rev.*, 36(2):1–5.

de Almeida, V. C., et.al. (2004). Sistema operacional yatos para redes de sensores sem fio. In *Primeiro Workshop de Sistemas Operacionais*, pages 176–176, Salvador.

Dunkels, A., Grönvall, B., and Voigt, T. (2004). 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.

Fröhlich, A. A. (2001). *Application-Oriented Operating Systems*. GMD - Forschungszentrum Informationstechnik, Sankt Augustin.

Han, C.-C.,et al. (2005). A dynamic operating system for sensor nodes. In *3rd international conference on Mobile systems, applications, and services*, pages 163–176, NY.

Handziski, V., Polastre, J., Hauer, J.-H., Sharp, C., and Culler, A. W. D. (2005). Flexible hardware abstraction for wireless sensor networks. In *Proceedings of the Second European Workshop on Wireless Sensor Networks (EWSN '05)*, Istambul, Turkey.

Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D., and Pister, K. (2000). 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.

Hoeller Junior, A. S., Wanner, L. F., and Fröhlich, A. A. (2006). A Hierarchical Approach For Power Management on Mobile Embedded Systems. In *5th IFIP Working Conf. on Distributed and Parallel Embedded Systems*, Braga, Portugal.

Hofmeijer, T., Dulman, S., Jansen, P., and Havinga, P. (2004). AmbientRT - Real time system software support for data centric sensor networks. In *2nd Int. Conf. on Intelligent Sensors, Sensor Networks and Information Processing, Melbourne*, pp. 61–66,.

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

Marcondes, H., Hoeller Junior, A. S., Wanner, L. F., and Fröhlich, A. A. (2006). Operating Systems Portability: 8 bits and beyond. In *11th IEEE Int. Conf. on Emerging Tech. and Factory Automation*, Prague.

Polastre, J., Hill, J., and Culler, D. (2004). 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. ACM Press.

Polastre, J., Szewczyk, R., and Culler, D. (2005). Telos: Enabling ultra-low power wireless research. In *4th Intl. Conf. on Information Processing in Sensor Networks (IPSN/SPOTS)*, Los Angeles, California.

Wanner, L. F., de Oliveira, A. B., and Fröhlich, A. A. (2007). Configurable Medium Access Control for Wireless Sensor Networks. In *Int. Embedded System Symposium*, pages 401–410, Irvine, CA, USA.

Wanner, L. F., Hoeller Junior, A. S., de Oliveira, A. B., and Fröhlich, A. A. (2006). Operating System Support for Data Acquisition in Wireless Sensor Networks. In *11th IEEE Int. Conf. on Emerging Tech. and Factory Automation*, Prague.

Wiedenhoft, G. R., Junior, A. S. H., and Fröhlich, A. A. (2007). A Power Manager for Deeply Embedded Systems. In *12th IEEE International Conference on Emerging Technologies and Factory Automation*, pages 748–751, Patras, Greece.