Suporte de Sistema Operacional para Tratamento de Heterogeneidade em Redes de Sensores sem Fios

Lucas Francisco Wanner , Arliones Stevert Hoeller Junior , Fauze Valério Polpeta , Antônio Augusto Fröhlich

[1]Laboratório de Integração Software/Hardware
Universidade Federal de Santa Catarina
C.P. 476 – 88040-900 Florianópolis, SC

{lucas,arliones,fauze,guto}@lisha.ufsc.br

Abstract. *Given the variability of Wireless Sensor Networks hardware in use today, the lack of proper abstraction and encapsulation mechanisms at the operating system level often forces developers to reimplement their sensing applications whenever a different sensor, radio or processor is deployed.*
In this paper we introduce a novel strategy for abstracting Wireless Sensor Networks Hardware, implemented for the EPOS *operating system. It consists in providing application programmers with a high level interface for sensing components, that can latter be bound to pre-existing components that are adapted on demand at system generation time to fulfill application requirements, thus enabling programmers to code portable sensing applications in spite of the hardware diversity and without significant overhead.*

Resumo. *Dada a variabilidade do hardware para Redes de Sensores sem Fios em uso hoje, a falta de mecanismos adequados de abstração e encapsulamento no nível de sistema operacional muitas vezes força os desenvolvedores a reimplementarem suas aplicações de sensoriamento sempre que um novo sensor, rádio ou processador são utilizados.*
Neste artigo introduzimos uma nova estratégia para abstração de hardware de Redes de Sensores sem Fios, implementada para o sistema operacional EPOS*. Esta consiste em fornecer aos programadores de aplicação uma interface de alto nível para componentes de sensoriamento, que pode ser associada a componentes pré-existentes que são adaptados sob demanda para atender aos requisitos da aplicação no tempo de geração do sistema, permitindo assim que os programadores escrevam aplicações de sensoriamento portáveis apesar da diversidade de hardware e sem custo adicional significante.*

1. Introdução

O hardware para Redes de Sensores sem Fios é, por sua própria natureza, heterogêneo e modular. Requisitos específicos de aplicações balizam o projeto de hardware, determinando desde capacidades de processamento até banda de rádio e módulos de sensores. Mesmo em uma mesma família de nodos de sensor (e.g. a família de *motes* de Berkeley) é possível encontrar diferenças arquiteturais que não podem ser trivialmente abstraídas. Neste cenário, uma aplicação de sensoriamento escrita 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

Mote					
Tipo	Rene	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	4 KB	2 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 1: Hardware típico de redes de sensores

plataforma de sensores de maneira adequada. De fato, a abstração adequada de plataformas de sensores torna-se um assunto chave face às modernas soluções SoC (*System on a Chip*), que integram micro-controlador, rádio e módulos de sensores em um único circuito integrado que normalmente pode ser configurado de acordo com as exigências dos usuários [Rabaey et al., 2002, Hill, 2003]. A variabilidade arquitetural do hardware para redes de sensores pode ser observada na tabela 1, que apresenta as características de alguns módulos representativos de nodos de redes de sensor. Somando-se às diferenças arquiteturais entre as plataformas, módulos de sensores (e.g. sensores de luz, temperatura, movimento) apresentam uma gama de variabilidade ainda maior. Módulos de sensor que apresentam a mesma funcionalidade podem apresentar diferenças em suas interfaces de acesso, características e parâmetros operacionais.

Um sistema de suporte a execução devidamente projetado poderia livrar os programadores de aplicação destas diferenças arquiteturais e promover portabilidade da aplicação entre diferentes plataformas de hardware. Dada uma aplicação de sensoriamento, não há um motivo forte pelo qual ela não possa ser reutilizada em outra plataforma que satisfaça seus requisitos (e.g. presença de determinado sensor, capacidade de armazenamento não volátil, etc.). Por exemplo, um sistema operacional poderia entregar uma abstração de sensor de temperatura que seria instanciada pelas aplicações com parâmetros de alcance e escala (e.g. linear, logarítmica, Poisson, etc.). O sistema então garantiria comportamento adequado independente do sensor físico que existe na plataforma.

Neste artigo apresentamos as estratégias atuais para tratamento de heterogeneidade em redes de sensores sem fios, e introduzimos um novo modelo baseado na metodologia de *projeto de sistema orientado à aplicação – application oriented system design* (AOSD) [Fröhlich, 2001]. AOSD introduz os conceitos de *mediadores de hardware* e *abstrações de sistema* que implementam conceitos de sistema *independentes de cenário*. Estas abstrações são exportadas para a aplicação através de *interfaces infladas*, e são adaptadas de acordo com as necessidades da aplicação através de *adaptadores de cenário*. Nós introduzimos o modelo de *abstrações de sensoriamento* utilizado no sistema EPOS, um sistema operacional experimental desenvolvido pelo Laboratório de Integração Software/Hardware, da Universidade Federal de Santa Catarina. Em seguida, mostramos que este modelo é capaz de abstrair adequadamente hardware de redes de sensores sem fios com praticamente nenhum *overhead*, produzindo um *footprint* de sistema muito pequeno.

2. Trabalhos Relacionados

Grande parte do desenvolvimento em sistemas operacionais e abstração de hardware para redes de sensores concentra-se no sistema TINYOS [Hill et al., 2004]. Inicialmente desenvolvido na Universidade da Califórnia em Berkeley, e atualmente um projeto *open-source* mantido por várias instituições, o TinyOS é o sistema operacional mais utilizado em redes de sensores sem fios. O sistema foi escrito em NESC [Gay et al., 2003], uma linguagem de programação de alto nível que emula a sintaxe e funcionalidade de linguagens de descrição de hardware, e provê componentes para comunicação, coordenação entre *threads* e, de nosso especial interesse, abstração de hardware.

Mesmo implementando uma camada de abstração de hardware (*hardware abstraction layer - HAL*) completa e funcional, o TINYOS apresenta uma série de problemas estruturais que podem comprometer a portabilidade de aplicações. Cada plataforma de hardware tem uma implementação completa e separada no TINYOS. Mesmo sendo verdade que diferenças arquiteturais entre plataformas podem exigir implementações separadas, as interfaces do sistema devem permanecer uniformes sempre que for possível. O TINYOS não provê interfaces uniformes para a maioria das abstrações de alto nível de sistema (e.g. sensores, timers, etc.), forçando o programador a entender os detalhes da plataforma de hardware e comprometendo a portabilidade da aplicação.

O sistema Mantis [Abrach et al., 2003], desenvolvido pela Universidade do Colorado, tem como objetivo tornar a tarefa de programar uma aplicação para redes de sensores o mais parecido possível com programar uma aplicação para um PC. Para tanto, implementa uma API *UNIX-like*, com uma camada de abstração de hardware monolítica.

Abstrações de mais alto nível normalmente fazem uso de máquinas virtuais que abstraem o hardware físico em uma arquitetura ideal virtual. Um mecanismo de tradução é responsável por garantir operação correta, independente de detalhes específicos do hardware utilizado. Assim, até mesmo implementações de máquinas virtuais bastante eficientes, como Maté [Levis and Culler, 2002], introduzem *overhead* no sistema. Isto torna-se especialmente crítico para plataformas de hardware com recursos restritos, como nodos de sensores, já que, para encontrarem uso prático, abstrações de alto nível para redes de sensores devem fazer uso eficiente das baixas capacidades de memória, processamento e energia dos nodos.

3. A solução de AOSD

Nesta seção introduziremos os conceitos de *abstrações de sistema* e *mediadores de hardware* como componentes para abstração de hardware reutilizáveis, de alto nível e eficientes para redes de sensores.

O *projeto de sistema orientado à aplicação* (*application oriented system design – AOSD*) [Fröhlich, 2001] foi proposto como uma metodologia multi-paradigma para desenvolvimento de software, e faz uso de diversas técnicas de programação e engenharia de software que podem ser combinadas para gerar sistemas de suporte a execução configurados e otimizados para aplicações específicas. O sistema operacional EPOS foi implementado de acordo com as técnicas de AOSD, e tem sido empregado nos mais diversos cenários de uso [Fröhlich et al., 2000, Polpeta and Fröhlich, 2004].

O principal objetivo do sistema EPOS é permitir que programadores de aplicação escrevam aplicações independentes de arquitetura, e através da análise da aplicação, fornecer suporte de tempo de execução que agregue todos os recursos que esta aplicação específica necessita, e nada mais. Para atingir estes objetivos, o EPOS faz uso dos conceitos

de *interfaces infladas, abstrações de sistema, aspectos de cenário, features configuráveis* e *mediadores de hardware*.

O EPOS faz amplo uso de técnicas *metaprogramação estática* e *programação orientada a aspectos* para implementar seus componentes de software, conferindo a eles uma vantagem significativa com relação as soluções tradicionais de máquinas virtuais e camadas de abstração de hardware. Através destas técnicas é possível adaptar o componente de acordo com o cenário de sua utilização, sem comprometer sua interface, e sem agregar código inútil.

3.1. Mediadores de Hardware

Mediadores de hardware [Polpeta et al., 2004] são propostos como construções de software que mediam a interação entre os componentes do sistema operacional, chamados *abstrações de sistema*, e os componentes de hardware. A principal idéia por traz dos mediadores de hardware não é construir camadas de abstração de hardware universais e máquinas virtuais, mas manter o *contrato de interface* entre o sistema e a máquina.

Diferente do que acontece nas HALs comuns, mediadores de hardware não constroem uma camada monolítica encapsulando os recursos disponíveis na plataforma de hardware. Cada componente de hardware é tratado através de seu próprio mediador, garantindo assim a portabilidade de abstrações que o utilizam, sem causar dependências desnecessárias. Mediadores de hardware devem ser fundamentalmente metaprogramados, podendo dissolver-se nas abstrações assim que o contrato de interface é cumprido. Em outras palavras, um mediador de hardware entrega a funcionalidade de seu hardware correspondente através de uma interface orientada ao sistema.

Os mediadores de hardware são organizados em famílias cujos membros representam as entidades significativas do domínio. Por exemplo, uma família de mediadores de `CPU` apresentaria membros como `ARM`, `AVR8`, e `PPC`. Um mediador de hardware simples para o barramento de GPIO no processador AVR é apresentado na figura 1. Este mediador abstrai as operações em linguagem assembly responsáveis por escrever e ler valores de portas de IO em operadores C++ de alto nível sem nenhum overhead.

3.2. Abstrações de sistema e interfaces infladas

AOSD utiliza *abstrações de sistema independentes de cenário* para implementar os componentes de sistema operacional. Estes componentes definem as funcionalidades do sistema, e sua implementação faz uso de diversas técnicas como *metaprogramação estática* e *programação orientada a aspectos*, permitindo que estas abstrações sejam fortemente configuráveis.

As abstrações de sistema são definidas a partir de um processo de *análise e decomposição de domínio orientada a objetos*. Este processo de análise é bastante similar à *decomposição orientada a objetos*. A principal diferença é que o projeto de sistema orientado à aplicação é uma metodologia multi-paradigma, assim outras entidades, como aspectos e *features* configuráveis devem surgir a partir desta análise.

Cada abstração de sistema é composta por um conjunto de componentes de sistema operacional similares. Estes componentes são organizados de acordo com o paradigma de *projeto baseado em famílias*, e têm suas *semelhanças* e *diferenças* exploradas através de diferentes 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. No EPOS, o framework do sistema seleciona

```
class AVR8_GPIO_Port:
        protected GPIO_Port_Common {
public:
 enum {
   PORTA = 0x39,
   PORTB = 0x36,
   PORTC = 0x33,
   PORTD = 0x30
 }; // ...
 void operator=(unsigned char value) {
   _ddr = (unsigned char)0xff;
   _port = value;
 }
 operator unsigned char() {
   _ddr = (unsigned char)0x00;
   return _pin;
 } // ...
private:
 IO_Register<unsigned char> _pin;
 IO_Register<unsigned char> _ddr;
 IO_Register<unsigned char> _port;
};
```

Figura 1: O mediador de hardware `AVR_GPIO`

automaticamente realizações de interface, levando em consideração o hardware alvo e um modelo de custo para os componentes.

3.3. Aspectos de cenário e *features* configuráveis

Em AOSD, aspectos não funcionais são fatorados como *aspectos de cenário* que podem ser aplicados a membros de famílias conforme for necessário. Por exemplo, famílias como `UART` e `Ethernet` muitas vezes precisam operar em modo de acesso exclusivo. No EPOS isto poderia ser atingido aplicando-se um aspecto de controle de compartilhamento às famílias.

Features configuráveis, por outro lado, representam características dos componentes que podem ser ligadas ou desligadas de acordo com os requisitos ditados pelas abstrações ou aplicações. Uma *feature* configurável não está restrita a uma *flag* que indica se uma determinada característica do hardware deve ser ativada ou não. Normalmente, incorpora também uma implementação na forma de *programação genérica* [Musser and Stepanov, 1989] dos algoritmos e estruturas de dados que são necessários para implementar esta característica quando o hardware não a provê. Um exemplo de *feature* configurável é a geração de códigos CRC em um mediador `Ethernet`.

4. Componentes para redes de sensores no EPOS

O sistema EPOS (*Embedded Parallel Operating System*) tem como objetivo entregar suporte de tempo de execução adequado a aplicações de computação dedicada. Para atingir este objetivo, o EPOS utiliza as técnicas de *projeto de sistema orientado à aplicação* para guiar o desenvolvimento de abstrações de sistema como famílias de componentes de software, cada qual implementando uma abstração de sistema independente de cenário que pode ser adaptada com ajuda de adaptadores de cenário. Componentes de software são coletados em um repositório e exportados para os programadores de aplicação através de *interfaces infladas* que escondem as peculiaridades de cada membro de uma família como

se toda a família fosse um único componente. Esta estratégia, além de reduzir drasticamente o número de componentes exportados, permite que os programadores expressem facilmente os requisitos de suas aplicações no que diz respeito ao sistema operacional.

Para preservar a portabilidade de seus componentes de software, o EPOS faz uso de mediadores de hardware. Em princípio, nenhuma das abstrações de sistema do EPOS interagem diretamente com o hardware, utilizando sim seus mediadores correspondentes. Desta maneira, uma troca de contexto no domínio de uma abstração de `Thread` diz respeito principalmente à decisão de qual *thread* deve ocupar a `CPU` a seguir, deixando a tarefa de salvar e restaurar o contexto para o mediador de `CPU` correspondente. Mediadores de hardware relevantes incluem, além dos módulos tradicionais (e.g. `CPU`, `MMU`, `Timer`, etc.), `Radio` e `Sensor`. Um exemplo mais substancial de abstração de sistema pode ser visto na família de abstrações `Sentient`.

4.1. Sensores e Sensientes

As diferenças entre as arquiteturas de sensores em uso hoje tornam difícil a tarefa de definir uma maneira comum de acessar tais dispositivos. Estes dispositivos variam desde circuitos integrados digitais simples até circuitos complexos e (normalmente) analógicos.

Os dispositivos da primeira categoria são normalmente circuitos digitais integrados simples, normalmente implementados sobre um barramento serial (e.g. I2C, SPI). Neste tipo de sensores, o processo de aquisição de dados é iniciado por um sinal específico ou por operações de leitura e escrita. Outra característica destes sensores é que existe um período regular de tempo no qual novos valores são sensoriados. Exemplos destes sensores incluem a família Texas TMP de sensores de temperatura, a maioria dos magnetometros Honeywell, os acelerômetros STM e a maior parte dos conversores A/D.

Na outra ponta, temos dispositivos muito específicos e, normalmente, analógicos. Exemplos desta categoria são os sensores de temperatura e luz da *Mica Sensor Board* de Berkeley. Estes sensores compartilham o mesmo circuito analógico, tornando seu uso e gerenciamento um assunto complexo. O circuito é composto por dois termistores e um foto-resistor, que são gerenciados por três pinos de GPIO, e que entregam seus resultados através do mesmo pino de um A/D. Mesmo sendo um circuito simples, possui restrições operacionais e de tempo. Outro exemplo é o magnetometro da placa de sensores *Mica*. Ele é implementado por um circuito analógico e digital complexo e, apesar de ser configurado por um potenciômetro I2C simples, tem saída analógica e características semânticas e de tempo complexas.

Mesmo com todas estas diferenças entre o hardware de sensoriamento, é possível afirmar que a operação de sensores segue um *padrão regular*. Este padrão é constituído por um comando "iniciar sensoriamento", seguido por uma fase de configuração opcional. Uma vez configurado, há um período de espera pela fase de aquisição de dados do sensor. Quando a aquisição de dados foi concluída, os dados finalmente são lidos. A rotina de sensoriamento pode então ser parada, ou mantida em um *loop* de "espera por aquisição / lê dados".

De maneira a contemplar todos os possíveis tipos de hardware de sensoriamento, o EPOS implementa as famílias de mediadores e abstrações apresentada a figura 2. A família de mediadores `Bus` implementa acesso uniforme a dispositivos periféricos. A família `ADC` de conversores A/D também implementa acesso uniforme a estes dispositivos. Estas duas famílias básicas de mediadores são a chave para a implementação modular do subsistema de sensoriamento do EPOS, uma vez que permite que a família de mediadores `Sensor` seja composta por componentes altamente configuráveis e reutilizáveis. Alguns dos mediadores implementados são apresentados na figura 2. Qualquer

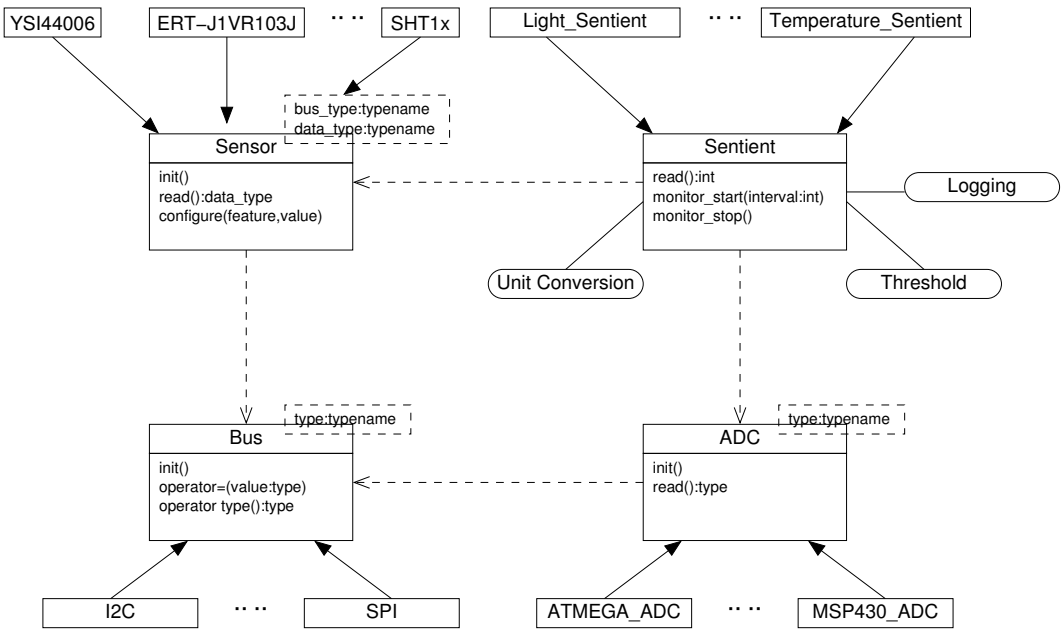

Figura 2: Conjunto de componentes relevantes para redes de sensores.

outra implementação simplesmente precisa estender o componente `Sensor` e fazer sua implementação dependente de hardware.

Estas estruturas de mediadores de hardware permitem o projeto de uma família de abstrações de mais alto nível e independente de arquitetura, que pode ser utilizada pelas aplicações sem afetar sua portabilidade. A família `Sentient` é composta por componentes de software que têm como objetivo abstrair dos sensores a sua finalidade, mas não sua implementação (que é considerada no mediador `Sensor`). Estes componentes implementam não somente acesso transparente a sensores, mas também outras funcionalidades como conversão de unidades e *logging* de dados, que são mapeadas como *features* configuráveis ou aspectos.

Uma vez que a aplicação usa um membro da família `Sentient`, é responsabilidade das regras de composição do *framework* do EPOS manter a coerência do subsistema de sensoriamento, garantindo que, por exemplo, a abstração `Temperature_Sentient` utilize um mediador `Sensor` de temperatura disponível. Esta coerência é garantida pelo processo de configuração e geração do sistema, que leva em consideração a análise da aplicação e a descrição da plataforma de hardware. Uma descrição completa deste processo é apresentada em [Tondello and Fröhlich, 2005].

5. Avaliação

Para testar a expressividade, portabilidade e custo do subsistema de sensoriamento do EPOS, implementamos uma simples aplicação de sensoriamento utilizando três sistemas operacionais diferentes: TinyOS, Mantis e EPOS.

A aplicação exemplo é constituída por um laço que constantemente lê os dados de um sensor de temperatura e redireciona os dados lidos para uma interface `UART`. Sempre que foi necessário expressar código dependente de arquitetura, foi considerada como plataforma destino da aplicação o *mote* Mica2 de Berkeley [Hill et al., 2000], desenvolvido pelo mesmo grupo responsável pelo TinyOS. A figura 3 apresenta as implementações da aplicação em questão nos três sistemas operacionais analisados.

```
/* TinyOS Sensing Application */
configuration SenseToUART {}
implementation {
  components Main, SenseToInt, IntToUART, TimerC,
                               DemoSensorC as Sensor;
  Main.StdControl -> SenseToInt;
  Main.StdControl -> IntToUART;
  SenseToInt.Timer -> TimerC.Timer[unique("Timer")];
  SenseToInt.TimerControl -> TimerC;
  SenseToInt.ADC -> Sensor;
  SenseToInt.ADCControl -> Sensor;
  SenseToInt.IntOutput -> IntToUART;
}
```

```
/* Mantis Sensing Application */
#include <inttypes.h>
#include "led.h"
#include "dev.h"
#include "com.h"
static comBuf send_pkt;
void start (void) {
   send_pkt.size=1;
   while(1) {
     dev_read(DEV_MICA2_TEMP,
              &send_pkt.data[0],1);
     com_send(IFACE_SERIAL,
              &send_pkt);
   }
}
```

```
/* EPOS Sensing Application */

#include <sentient.h>
#include <uart.h>

Temperature_Sentient t;
UART u;

int main()
{
  while(1)
  {
    u.send_byte(t.read());
  }
}
```

Figura 3: Aplicações de sensoriamento exemplo

5.1. Expressividade

Todos os sistemas proveram interfaces capazes de satisfazer os requisitos do problema. Para melhor efeito de comparação de *overhead*, as aplicações implementadas utilizaram o mínimo de recursos de sistema operacional possível, não fazendo uso, por exemplo, das capacidades *multithreading* presentes nos três sistemas.

A aplicação para o TinyOS foi escrita na linguagem de programação `NesC`, e sua implementação consiste simplesmente em conectar entradas e saídas de interfaces (`ADC` e `UART` no caso desta aplicação). Caso fosse necessário, a implementação de algoritmos na aplicação poderia ser feita na própria linguagem `NesC`, com sintaxe semelhante à linguagem `C`. A aplicação para o Mantis foi implementada em `C`, incluindo os cabeçalhos com as APIs de comunicação e acesso a dispositivos do sistema. A aplicação para o EPOS foi implementada em `C++`, importando os cabeçalhos da abstração de sensoriamento `Sentient` e do mediador `UART`.

5.2. Portabilidade

Na plataforma Mica2, o sistema TinyOS reflete o design de hardware e exporta o sensor de temperatura para a aplicação como um 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 funcio-

nalidade da aplicação 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. A leitura do sensor de temperatura 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 uma série de `defines` indicando 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 de temperatura 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.

5.3. Custo

A tabela 2 apresenta os tamanhos de código e dados das aplicações teste geradas. O sistema EPOS apresentou o menor custo em bytes dentre os três sistemas testados, tanto para memória de dados quanto para tamanho de código gerado. Evidentemente, a avaliação de custo baseada simplesmente no tamanho de código e dados da aplicação é incompleta, mas em sistemas com recursos limitados como nodos de sensores, estes dados tem peso muito grande. Próximas avaliações deste trabalho incluirão tamnbém medições de performance e consumo de energia.

	Mantis	TinyOS	EPOS
.text (bytes)	22486	9990	5234
.data (bytes)	74	16	38
.bss (bytes)	711	358	298

Tabela 2: Tamanho das aplicações geradas

6. Conclusões

Neste artigo discutimos o problema da heterogeneidade no hardware para redes de sensores sem fios, e apresentamos novas técnicas para tratar portabilidade de aplicações nestes sistemas. Utilizamos os conceitos *mediadores de hardware* e *abstrações de sistema* para fundamentar o subsistema de sensoriamento do sistema EPOS, que permite que aplicações de sensoriamento sejam portáveis sem modificações entre diferentes plataformas de hardware para redes de sensores sem fios. Uma aplicação exemplo de sensoriamento foi implementada no sistema EPOS e em dois dos principais sistemas operacionais disponíveis para redes de sensores. O sistema EPOS apresentou melhor suporte a portabilidade da aplicação, além de menor custo em termos de tamanho de código e dados. Estes resultados posicionam o EPOS como uma alternativa bastante viável para sistemas operacionais para dispositivos de sensoriamento.

Referências

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

Fröhlich, A. A. (2001). *Application-Oriented Operating Systems*. Number 17 in GMD Research Series. GMD - Forschungszentrum Informationstechnik, Sankt Augustin.

Fröhlich, A. A., Tientcheu, G. P., and Schröder-Preikschat, W. (2000). 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. Springer.

Gay, D., Levis, P., von Behren, R., Welsh, M., Brewer, E., and Culler, D. (2003). The nesc language: A holistic approach to networked embedded systems. In *Proceedings of Programming Language Design and Implementation*, San Diego, CA.

Hill, J. (2003). *System Architecture for Wireless Sensor Networks*. PhD thesis, University of California, Berkeley.

Hill, J., Horton, M., Kling, R., and Krishnamurthy, L. (2004). The platforms enabling wireless sensor networks. *Communications of the ACM*, 47(6):41–46.

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.

Levis, P. and Culler, D. (2002). Maté: 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.

Musser, D. R. and Stepanov, A. A. (1989). Generic programming. In *Proceedings of the First International Joint‘ Conference of ISSAC and AAECC*, number 358 in Lecture Notes in Computer Science, pages 13–25, Rome, Italy. Springer.

Polpeta, F. V. and Fröhlich, A. A. (2004). 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. Springer.

Polpeta, F. V., Fröhlich, A. A., Junior, A. S. H., and D'Agostini, T. S. (2004). Portabilidade em Sistemas Operacionais Baseados em Componentes de Software. In *First Brazilian Workshop on Operating System*, pages 1466–1475, Salvador, Brazil.

Rabaey, J., Ammer, J., Karalar, T., Li, S., Otis, B., Sheets, M., and Tuan, T. (2002). Picoradios for wireless sensor networks: The next challenge in ultra-low-power design. In *Proceedings of the International Solid-State Circuits Conference*, San Francisco, CA.

Tondello, G. F. and Fröhlich, A. A. (2005). On the Automatic Configuration of Application-Oriented Operating Systems. In *3rd ACS/IEEE International Conference on Computer Systems and Applications*, Cairo, Egypt.