Revista Brasileira de Engenharia Biomédica,
v. 23, n. 3, p. 231-244, dezembro 2007

ISSN 1517-3151

Artigo Original
Recebido em 30/10/2006, aceito em 03/12/2007

Uma rede de sensores para monitoração do corpo humano com suporte à programação

A body sensor network with programming support

Francisco Assis de Oliveira Nascimento
Adson Ferreira da Rocha*
Grupo de Processamento Digital de Sinais,
Departamento de Engenharia Elétrica,
Faculdade de Tecnologia / UNB
70910-900 Brasília, DF
E-mail: adson@unb.br

Talles Marcelo Gonçalves de Andrade Barbosa
Iwens Gervásio Sene Junior
Departamento de Computação,
Universidade Católica de Goiás

Hervaldo Sampaio Carvalho
Faculdade de Medicina / UNB

*Autor para correspondência

Resumo

Este trabalho apresenta uma rede de sensores utilizada para monitoração de variáveis fisiológicas do corpo humano com suporte à programação. A tarefa de configuração desse tipo de sistema é usualmente realizada por desenvolvedores especializados, profissionais da computação com grande conhecimento de linguagens de programação. Entretanto, para que essa tecnologia se torne clinicamente viável, é necessário que os próprios profissionais da área de saúde possam fazê-lo. A programação e a reconfiguração à distância de uma rede de sensores sem fios é o principal objetivo deste trabalho. Como contribuição maior é apresentada uma arquitetura de *software* denominada SOAB (*Software Architecture for Body-worn Sensor Networks Project*). Concebida com base em uma abordagem *top-down*, a arquitetura SOAB é constituída por quatro camadas independentes: i) uma interface gráfica direcionada aos profissionais de saúde; ii) *middleware* para interconexão da rede de sensores para monitoração do corpo humano com a Internet; iii) um servidor para execução dos serviços solicitados pelos programadores; e iv) um sistema operacional com suporte para multitarefa que será embutido nos nós-sensores. Esse sistema operacional foi denominado MedOS e visa aumentar a sobrevida dos nós-sensores (tempo de operação), promovendo a redução do consumo de energia elétrica por meio do escalonamento de tarefas com base em políticas adaptadas para aplicações biomédicas. A sistematização dessas políticas foi obtida por meio da utilização de um modelo baseado em autômatos. Para avaliar a arquitetura SOAB foi elaborado e aplicado um teste de carga, cujo objetivo foi quantificar o tempo gasto para programação de um nó-sensor do eletrocardiograma (ECG). Os resultados dos testes mostraram que o sistema proposto tem um bom potencial para se tornar uma ferramenta eficiente para a programação de redes de sensores para monitoração do corpo humano por profissionais não-especializados na área de informática.

Palavras-chave: Redes de sensores, Java, *Web services*, Autômatos, Multitarefa.

Abstract

This paper presents a wireless sensor network that is used for monitoring physiological data in the human body (or Body Sensor Networks – BSN) with configuration support. The configuration of this type of system is usually performed by specialized engineers. However, if this technology is to become clinically useful, it is essential that the healthcare professionals are able to configure the system, and the goal of this work is to make this task possible. The main contribution of this work is the proposal of a software architecture that was named SOAB (Software Architecture for Body-worn Sensor Networks Project). The SOAB architecture has been conceived based on a top-down approach and it is composed of four independent layers: i) a graphical interface oriented to health-

care professionals; ii) middleware for interconnecting BSN's to the Internet; iii) a server for processing clients' requests; and iv) a multitasking operating system that is embedded into the sensor nodes. This operating system was called MedOS and, among other features, it helps to increase the lifetime of batteries by scheduling tasks based on customized policies, designed for taking into account the specificities of biomedical applications. To implement these policies, an automata-based model has been used. For the evaluation of the system, a benchmarking approach has been developed and applied, in order to quantify the time spent for programming ECG sensor nodes. The results of the tests showed that the proposed system has a good potential to become an effective tool for programming body sensor networks by professionals that are not specialists in information technology.
Keywords: *Body sensor networks, Java, Web services, Automata, Multitasking.*

Extended Abstract

Introduction
The goal of the Body-Worn Sensor Networks project (BWSNET) is to build an infrastructure for monitoring the human health through wireless sensor networks embedded in the user's (patient) clothes and even in the body. An application-oriented software architecture, called SOAB (Software Architecture for Body-Worn Sensor Networks Project), has been developed in order to allow programming (at deployment-time) and reconfiguration (at run-time) of sensor networks used to monitor the human body, which can be performed by healthcare personnel.

Materials and Methods
The sensor-nodes were built using the Olimex MSP-P149 kit (Olimex, 2006), with a bluetooth radio (SP, 2006), shown in Figure 2a. This system allows the monitoring of electrocardiographic and electromyographic signals, as well as skin temperature, arterial pressure and galvanic skin resistance (Figure 2b).

In the first layer of the SOAB (Figure 1), we have the BWSNET Configuration Tool. This graphical interface is responsible for providing means by which the healthcare professionals can describe their algorithms in a way that is less time-consuming, less error-prone and more intuitive. This interface also allows the possibility of including new sensors that were not initially specified, without the need to recompile the source code. To achieve that, an interface based on Java Reflection (TJT, 2006) has been used. The BWSNET Configuration Tool has also a simulator. Figure 3 and Figure 4 show instances of the BWSNET Configuration Tool.

In order to provide support for the programming and reconfiguration of the sensor nodes from the Internet, a group of remote procedure calls (RPC) has been implemented in compliance with the recommendations of the W3C Web Services (W3C, 2004; W3C, 2006). Figure 5 shows examples of messages used for RPC implementation.

In the third layer we have the BWSNET Proxy. It is responsible for the translation and execution of the client's requisitions, i. e., it has the responsibility to create the actual image of the system from an abstract model based on automata. In Figure 6, the main components of the BWSNET Proxy are presented. Figure 7 shows an automaton that represents the system image generated based on the configuration presented in Figure 3.

In the last layer of the SOAB, there is an application-oriented operating system, called MedOS. The main objective of the MedOS is to provide support for the behavioral adjustment of the sensor-node at run-time, changing the priority values that can be associated with the tasks provided by the sensor-node. Figure 8 shows the main artifacts of the MedOS and their dependencies.

Results
To evaluate the SOAB, a set of techniques has been developed based on international standards (IEEE, 1993; ISO, 2001). The parameters to be assessed were: functionality, portability and efficiency. For most of these parameters, specific white box tests were applied and the results were incorporated into the source code comments. For efficiency, black box benchmarking has been performed in order to assess the response time of the deployment-time programmability feature. Figure 9 shows the software artifacts and the arrangement of the devices for the benchmarking tests, and Table I summarizes the main results obtained. Deployment cycle duration corresponds to the (rounded) mean response time (in seconds), obtained from 10 interactions between the Httperf (Mosberger and Jin, 1998) and the BWSNET Proxy.

Discussion
The values of the response time (presented in Table 1) tend to increase according to the complexity of the code and to the inherent level of intelligence included in the sensor-nodes. In practice, these values quantify the period of time that the patient will have to wait with an inactive system, while connected to the Proxy through the interface JTAG. In general, the health condition of the patient determines the frequency of system reprogramming. However, eventually the need for reprogramming due to software or hardware faults may also influence this frequency.

In many cases, reprogramming can be achieved by simply performing a slight behavioral adjustment in the software at run-time. The run-time reconfiguration methodology allows the programmers to act not only at the sensor network level but also at the sensor-node level. In the network level, for example, the Sensor QoS Level parameter (see Figure 10) refers to the importance of the information flow collected by a certain sensor-node. At the sensor-node level, a priority value will be associated to each executed task. The changing in the values of the priorities can modify the sequence and, especially, the time-slice during which each task will use the CPU. The programmer will also be able to suspend functionalities that are less interesting in favor of others of greater interest.

Conclusion
This paper presents a model for the programming of BSN's taking into account the fact that the programmers of these systems need tools that are more transparent in order to carry out their activities in such a way that these activities are not an inconvenience to their patients and to themselves.

The main contributions of this work are: i) the proposal of a graphical interface designed and implemented according to the needs of the healthcare professionals; and ii) the utilization of multitasking in BSN's. This concept allows the inclusion of tasks with more complexity and more interactivity among the sensor nodes. Both technological innovations were not described in literature yet.

Future efforts will be concentrated on the implementation of usability tests, mechanisms for authentication and security improvement, and new algorithms based on priorities and application-oriented algorithms that can be used to get more efficient scaling of the functionalities embedded in the sensor-nodes.

Introdução

Uma Rede de Sensores para o Corpo Humano (*Body Sensor Network* – BSN) é um sistema distribuído composto por nós-sensores, usualmente interconectados por um meio de comunicação sem fios e alimentados por baterias. Em geral, o nó-sensor é composto por uma unidade de processamento, memória, interfaces e transdutores, rádio transmissor e receptor, circuito de alimentação e bateria. As BSNs podem ser usadas para o monitoramento ininterrupto e não-obstrutivo da saúde humana. Atualmente existem diversas aplicações desses sistemas para o monitoramento de sinais eletrofisiológicos (Jovanov, 2006; Welsh, 2006; Yang, 2006). No futuro, as redes de sensores sem fios poderão ser embutidas na indumentária humana (*wearable systems* e/ou *wearable robots*) (Paradiso *et al*., 2006; Sankai *et al*., 2005) e até mesmo distribuídas dentro do próprio corpo sob a forma de nanosensores (NASA/NCI, 2006).

Não foram reportadas ainda, na literatura especializada, ferramentas (*software*) adaptadas para que os profissionais de saúde possam configurar aplicações para essas redes. Além disso, o *framework* utilizado pela maioria dos projetos, o TinyOS (TinyOS, 2006), não oferece suporte à multitarefa, dificultando a criação de programas mais complexos e interativos, e, ainda, impõe ao programador a utilização da linguagem de programação nesC (Gay *et al*., 2003). Como resultado, a popularização e a eficiência do uso desses dispositivos podem ser comprometidas, haja vista a necessidade de profundo conhecimento técnico acerca do funcionamento da tecnologia.

Um aspecto original deste trabalho envolve a proposta de mudança do paradigma utilizado atualmente para programação e reconfiguração das BSNs. Como o sistema se baseia em uma arquitetura de *software* orientada pela aplicação, espera-se que os profissionais de saúde possam se tornar os mantenedores das BSNs. O termo "orientado pela aplicação" foi originalmente utilizado para caracterizar um sistema operacional fortemente compromissado com as aplicações (Fröhlich, 2001, p. 44). Uma BSN deve ser projetada com base em requisitos de uma aplicação ou de um conjunto de aplicações similares e, portanto, entende-se que deva ser orientada pela aplicação (Barbosa *et al*., 2006).

Como prova do conceito, foi desenvolvida uma plataforma composta por *hardware* e por uma arquitetura de *software*, que é a contribuição maior deste trabalho. A arquitetura SOAB (*Software Architecture for Body-worn Sensor Networks Project*), ilustrada pela Figura 1, integra conceitos, tecnologias e modelos especificados com base

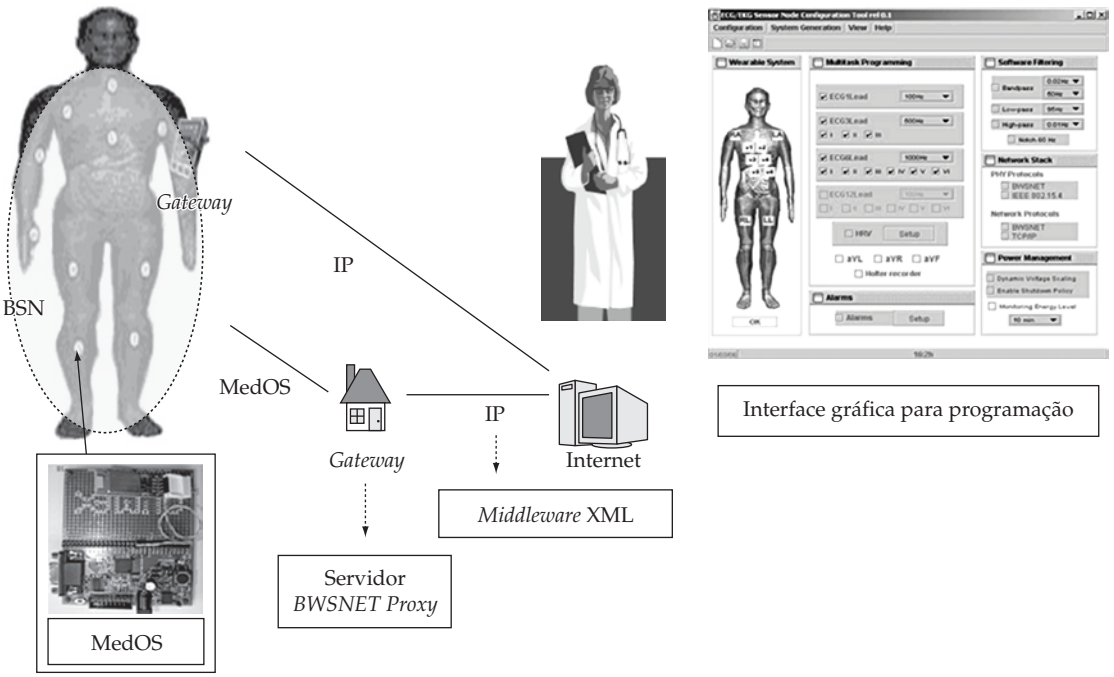

Figura 1. Visão geral do sistema e da arquitetura SOAB (*Software Architecture for Body-worn Sensor Networks Project*). Na imagem, são mostradas as quatro camadas da arquitetura SOAB: i) interface gráfica direcionada aos profissionais de saúde para programação da BSN; ii) *middleware* para interconexão da BSN com a Internet; iii) servidor *BWSNET Proxy*; iv) sistema operacional MedOS. ***Figure 1.*** *General view of the system and of the SOAB (Software Architecture for the Body-worn Sensor Networks Project): i) graphical interface directed to healthcare professionals for programming the BSN; ii) middleware for interconnection of the BSN with the Internet; iii) BWSNET Proxy server; iv) MedOS operating system.*

em requisitos direcionados (orientados) pela aplicação para compor uma solução eficiente.

A arquitetura SOAB é constituída por quatro camadas independentes: i) uma interface gráfica direcionada aos profissionais de saúde; ii) *middleware* para interconexão da rede de sensores do corpo humano com a Internet; iii) um servidor para execução dos serviços solicitados pelos programadores e, finalmente, iv) um sistema operacional com suporte para multitarefa que será instalado nos nós-sensores. Esse sistema operacional foi denominado MedOS e visa a aumentar a sobrevida do sistema, aumentando o tempo de funcionamento dos nós-sensores, por meio do escalonamento de tarefas com base em políticas adaptadas para aplicações biomédicas. O tempo de funcionamento é limitado pela quantidade de energia elétrica armazenada na bateria. Para sistematizar essas políticas, foi utilizado um modelo baseado em autômatos.

Para avaliar a arquitetura SOAB foi elaborado e aplicado um teste de carga, cujo objetivo foi quantificar o tempo gasto para a programação de um nó-sensor do sinal eletrocardiográfico (ECG). Esse teste de carga faz parte de um conjunto de técnicas que vêm sendo desenvolvidas com base em normas internacionais e que servirão para avaliação de sistemas como o apresentado neste artigo.

Materiais e Métodos

A seguir são apresentados os diversos componentes de *hardware* e *software* do sistema projetado para gerenciar à distância uma rede de sensores do corpo humano.

O *hardware*

Os nós-sensores foram construídos usando como base a plataforma Olimex MSP-P149 (Olimex, 2006). O microcontrolador MSP430F149 (TI, 2006) é a base desse sistema e representa a unidade de processamento, de memória e alguns periféricos embutidos em um único *chip*. Para interconexão do sistema utilizou-se o módulo BlueSMiRF v1.0 (SP, 2006). O BlueSMiRF v1.0 possibilita a construção de um *link* serial *full-duplex* entre o nó-sensor e qualquer outro dispositivo que tenha interface de rede *bluetooth*, incluindo os outros nós-sensores. Foram desenvolvidos ainda circuitos eletrônicos para o monitoramento do eletrocardiograma (ECG), do eletromiograma de superfície (EMG), da temperatura cutânea (TC), da resistência galvânica da pele (GSR) e da pressão arterial (PA). Esses circuitos têm a seguinte estrutura básica: i) sensor específico para aquisição de sinal; ii) amplificador do sinal capturado; e iii) filtro analógico. Acerca do consumo de energia elétrica, o microcontrolador MSP430 e os amplificadores utilizados podem operar com níveis muito baixos de potência. Entretanto, o rádio transmissor consome muita energia. A Figura 7 apresenta alguns resultados preliminares obtidos de experimentos para avaliação do consumo de corrente do nó-sensor apresentado neste texto. É importante ressaltar que quando o rádio transmissor está em modo de operação *standby* a drenagem de corrente elétrica pelo nó-sensor é reduzida em aproximadamente 94,64%.

Em função das fontes de dados disponíveis, os nós-sensores podem ser multifuncionais, quando executarem mais de uma funcionalidade (capacidade de sensoriamento), compartilhando o tempo de processamento por meio da multitarefa, ou monofuncionais, caso contrário (Barbosa *et al.*, 2006b). A Figura 2a exibe a arquitetura do nó-sensor desenvolvido, e a Figura 2b, um protótipo para aquisição dos sinais eletrofisiológicos.

SOAB: a interface gráfica

Como parte integrante da SOAB, a *BWSNET Configuration Tool* é uma interface gráfica desenvolvida utilizando tecnologia Java (Sun, 2006) com base em uma modelagem orientada a objetos. Essa interface apresenta dois níveis de abstração: i) no nível mais alto, a rede de sensores e seus parâmetros reconfiguráveis, e ii) no nível mais baixo, os nós-sensores, que podem ser multifuncionais ou dedicados a um único propósito (Barbosa *et al.*, 2006b). Essa interface objetiva facilitar a programação dos nós-sensores em tempo de compilação, durante o ciclo de *deployment*, e ainda permitir a reconfiguração da rede e dos nós-sensores durante a execução da aplicação. Essa interface permite que uma BSN seja programada e reconfigurada local ou remotamente via Internet sem perda de generalidade, ou seja, com a possibilidade de incluir novos sensores que não foram previamente especificados. O modelo de dados que representa cada um dos nós da rede amplia as funcionalidades da interface proposta por Carvalho (2005, p. 31) para um nó-sensor genérico. Esse modelo de dados uniformiza as informações a respeito das capacidades e dos objetivos de cada sensor, facilitando a inserção de novos sensores que não foram inicialmente planejados. Para dar suporte à inclusão de novos sensores sem a necessidade da recompilação do código-fonte, a *BWSNET Configuration Tool* implementa um mecanismo baseado em Java Reflection (TJT, 2006), com suporte para Java Applets, Java Applications e Thinlets (Bajzat, 2006). Esse me-

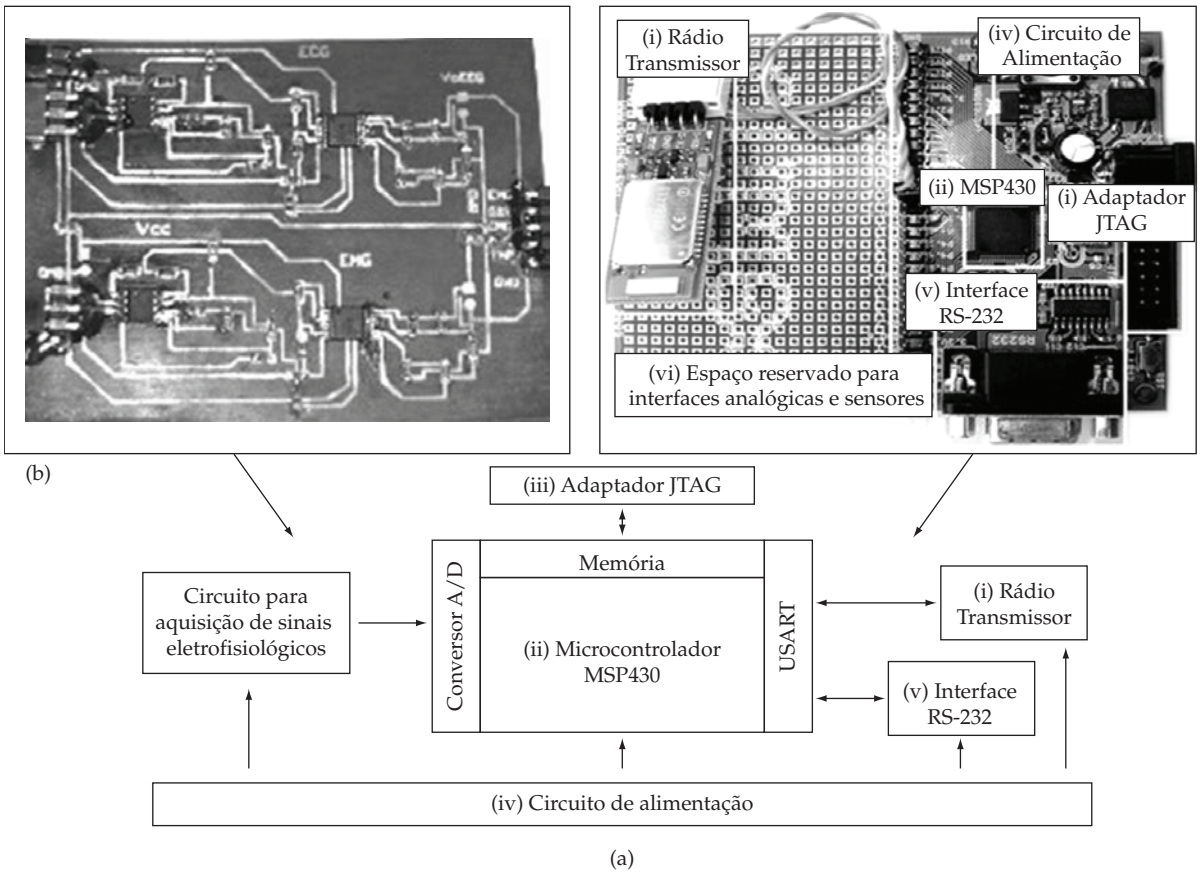

Figura 2. a) arquitetura do nó-sensor composta por i) rádio transmissor BlueSMIRF com tecnologia *bluetooth*; ii) microcontrolador MSP430F149; iii) adaptador JTAG (*Joint Test Action Group*); iv) circuito de alimentação; v) interface de comunicação serial padrão RS232; e vi) espaço reservado para inclusão de sensores e interfaces analógicas; e b) protótipo de um dos circuitos desenvolvidos para aquisição de sinais eletrofisiológicos implementado em uma placa de circuito impresso com dupla face (Barbosa *et al.*, 2006c). Esta placa, em particular, inclui apenas circuitos de amplificadores de sinais eletrocardiográficos e eletromiográficos. ***Figure 2.*** *a) architecture of the sensor node, composed of i) BlueSMIRF radio transmitter with bluetooth technology, ii) MSP430F149 microcontroller, iii) JTAG (Joint Test Action Group) adapter, iv) power supply circuit, v) RS232 serial communication interface, and vi) free space for inclusion of sensors and analog interfaces; and b) prototype of one the circuits that were developed for acquisition of electrophysiologic signals, implemented in a double-face printed circuit board (Barbosa et al., 2006c). This particular board includes only circuits for amplification of electrocardiographic and electromyographic circuits.*

canismo habilita o sistema a carregar em memória um novo objeto que representará o novo sensor, fazer sua introspecção e finalmente executá-lo a partir de uma invocação de métodos descobertos em tempo de execução (Barbosa *et al.*, 2006b). Isso possibilita, por exemplo, que um novo tipo de sensor seja incluso em *BWSNET Configuration Tool* sem que o sistema seja desativado para recompilação. A utilização de Thinlets permite ainda que a interface gráfica do nó-sensor possa ser manipulada, construída e modificada com base em um arquivo de configuração em formato XML (*eXtensible Markup Language*).

A programação da rede normalmente é iniciada pela programação dos nós-sensores. O nível de transparência oferecido aos programadores é configurável, isto é, o programador pode simplesmente selecionar as tarefas executadas por esses dispositivos, bem como o nível de inteligência embutido em cada nó-sensor, por exemplo, pela inclusão de algoritmos para remoção de ruído, pela seleção dos mecanismos para gerenciamento de energia e pela seleção dos alarmes, entre outros. A Figura 3 ilustra a interface de programação de um nó-sensor multifuncional para a captura do ECG.

A título de exemplo, suponha-se que o programador tenha selecionado uma configuração pela seleção dos eletrodos, como mostrado na Figura 3. Nesse caso, o programador poderá dispor de um nó-sensor de

ECG com três diferentes funcionalidades: i) ECG com uma única derivação operando com freqüência de amostragem de 100 Hz; ii) ECG com três derivações operando com freqüência de amostragem de 500 Hz e iii) ECG com seis derivações operando com freqüência de amostragem de 1.000 Hz. Logo que o programador finaliza suas escolhas, um arquivo de texto (em formato XML) com as informações a respeito das configurações selecionadas pelo programador é gerado pelo sistema. Após o comando do programador, o conteúdo desse arquivo é encaminhado para a camada subjacente implementada com *middleware* XML.

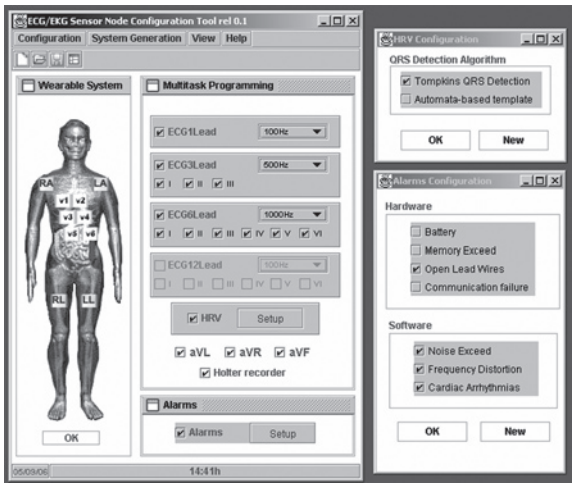

Figura 3. A interface de programação para um nó-sensor de ECG (Barbosa *et al*., 2006b). O nível de transparência oferecido aos programadores é configurável. O programador pode simplesmente selecionar as tarefas a serem executadas pelos nós-sensores (veja janela *Tasks*) como também o nível de inteligência embutido em cada nó-sensor, por exemplo, pela inclusão de algoritmos para remoção de ruído, ou pela seleção dos mecanismos para gerenciamento de energia ou dos alarmes, entre outros. ***Figure 3.*** *The programming interface for an EKG sensor node (Barbosa et al., 2006b). The level of transparency offered to programmers is configurable. The programmer can simply select the tasks to be executed by the sensor nodes (see the window Tasks) as well as the level of intelligence embedded in each sensor-node, for example, by the inclusion of algorithms for removal of noise, or by the selection of mechanism for power management, or of alarms, among others.*

Além da geração automática do sistema a partir da descrição visual das tarefas, a *BWSNET Configuration Tool* prevê que, para cada nó-sensor, seja disponibilizado um simulador, para que o programador possa avaliar a sobrevida (tempo de operação estimado) de cada nó-sensor com base na quantidade de energia estimada e na programação efetuada. Uma ferramenta como essa permite que o programador possa avaliar o desempenho das configurações escolhidas antes que a programação do nó-sensor seja de fato iniciada, evitando que o sistema seja programado desnecessariamente. A Figura 4 mostra um exemplo de resultados gerados pelo simulador para um nó-sensor de ECG. Como entrada para esse sistema tem-se: i) o arquivo de configuração gerado automaticamente pela ferramenta de programação, e ii) o nível de energia do nó-sensor, estimado pelo próprio programador. O principal resultado, o parâmetro "Deadline", corresponde ao tempo de vida máximo estimado para o nó-sensor. Os demais parâmetros são relacionados à eficiência de uso.

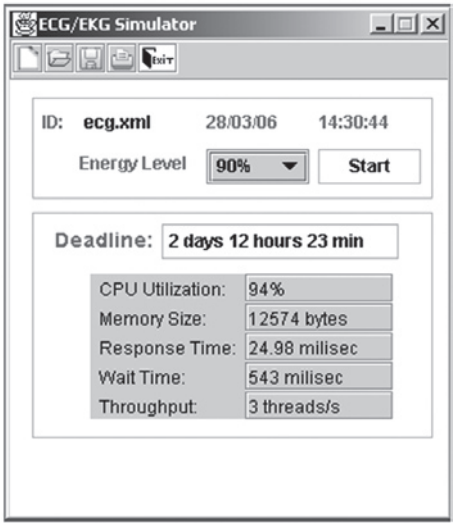

Figura 4. Simulador para o nó-sensor de ECG (Barbosa *et al.*, 2006b). Os parâmetros de entrada para este sistema são o arquivo de configuração, gerado automaticamente pela ferramenta de programação, e o nível de energia do nó-sensor, estimado pelo próprio programador. O principal resultado, o parâmetro "Deadline" corresponde ao tempo de vida máximo estimado para o nó-sensor. Os demais parâmetros são relacionados à eficiência do uso da memória Flash e da execução multitarefa embutida no nó-sensor. ***Figure 4.*** *EKG sensor node simulator (Barbosa et al., 2006b). The input parameters to this system are the configuration file, which is automatically generated by the programming tool, and the energy level of the sensor node, which is estimated by the programmer himself. The main result, the "Deadline" parameter, quantifies the maximum estimated lifetime for the sensor node. The remaining parameters are related to the efficiency in the use of the Flash memory and of the multitasking feature embedded in the sensor node.*

SOAB: *Middleware* XML

Foi desenvolvida uma camada para interconexão da BSN com a Internet. Assim, a BSN passa a ser vista como uma entidade independente e fornecedora de serviços para a camada 1: a *BWSNET Configuration Tool*. Para isso, *middleware* baseado em RPC (*Remote Procedure Call*) foi implementado de acordo com os protocolos especificados pelo Web Services W3C e pela a arquitetura SOA (*Service Oriented Architecture*) (W3C, 2004). Todos os protótipos desenvolvidos foram construídos utilizando tecnologia Java, tanto para criação e interpretação (*parsing*) das mensagens SOAP (*Simple Object Access Protocol*) quanto para encapsulamento dos comandos utilizados para geração dos sistemas que serão instalados nos nós-sensores. Para aumentar o desempenho e facilitar a portabilidade entre plataformas diferentes, não foi utilizado nenhum *framework* durante a implementação, como, por exemplo, o Xerces (Apache, 2006). Todas as mensagens SOAP, bem como as mensagens HTTP (*Hypertext Transfer Protocol*) utilizadas para suporte, foram implementadas manualmente. A Figura 5 apresenta um trecho do código-fonte que implementa as mensagens de resposta para a operação "*deploy*". Essa operação aciona a camada subjacente para a instalação da imagem (arquivo binário) do sistema.

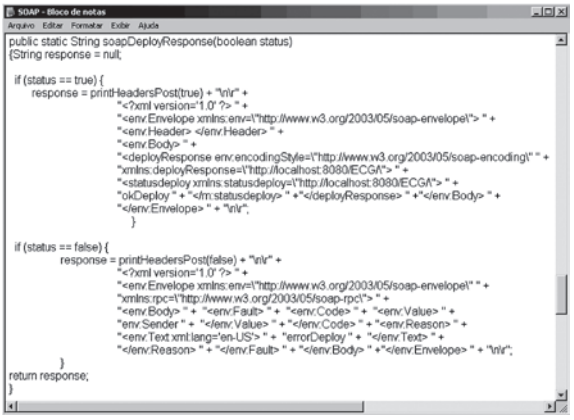

Figura 5. Mensagens SOAP utilizadas para implementação da resposta para a operação "*deploy*". Essa operação pode gerar duas respostas, uma de confirmação positiva, caso a operação seja concluída com êxito, e outra para confirmar que houve falha na execução da operação. A operação "*deploy*" é responsável pela instalação do sistema nos nós-sensores. ***Figure 5.*** *SOAP messages that are used for the implementation of the response to the "deploy" operation. This operation can generate two responses, one of positive confirmation, in case the operation is successfully performed, and one to confirm that the operation failed. The "deploy" operation controls the installation of the system in the sensor nodes.*

A interconexão da rede de sensores por meio de *web services* permite que a mesma possa ser vista como uma entidade que fornece serviços para vários usuários com necessidades diferentes e dinâmicas, independentemente da interface gráfica para programação e reconfiguração. O modelo prevê a incorporação de novas funcionalidades sob a forma de novos serviços que podem ser implementados por terceiros e facilmente adicionados ao sistema. Essa é, de fato, a principal razão para a proposta de uso da linguagem XML e do protocolo SOAP, ambos reconhecidos como padrões Internet, como mecanismos para representar toda a comunicação da aplicação. O acesso ao sistema através de uma interface padrão de alto nível, baseada na linguagem XML e no protocolo SOAP, permite uma interação do tipo aplicação-aplicação (*business-to-business*), a qual é muito mais flexível para as aplicações clientes do que o acesso único e exclusivo por meio de interfaces gráficas pré-definidas, possibilitando ainda uma separação clara das funcionalidades de cada camada. O fato de não se basear em protocolos proprietários, e sim em padrões ubíqüos da Web, como SOAP e XML, fornece ao *middleware* grandes capacidades de interoperabilidade e extensibilidade. Aplicações podem ser escritas em diferentes linguagens de programação e interagir com a rede através do suporte de ferramentas para a geração automática de mensagens SOAP a partir de documentos WSDL. Desenvolvedores de protocolos de comunicação para a rede ou de novos serviços para o *middleware* podem interagir com a rede através dos Esquemas XML e do documento WSDL fornecidos. Neste projeto, utilizou-se a linguagem WSDL (*Web Services Description Language*) versão 2.0 (W3C, 2006) para a criação dos arquivos que descrevem os serviços.

SOAB: *BWSNET Proxy*

Na terceira camada da arquitetura SOAB, tem-se o *BWSNET Proxy*. Essa ferramenta (*software*) vem sendo desenvolvida para operar nos dispositivos de *gateway*. Esses dispositivos podem ser computadores pessoais, *laptops*, computadores de mão (PDAs) ou até mesmo telefones celulares, desde que possuam uma das interfaces de rede compatível com a BSN. O *BWSNET Proxy* é um servidor que traduz as requisições de serviços do formato SOAP em comandos compreensíveis para a programação e para a reconfiguração dos nós-sensores. Além disso, gera o *software* que será instalado e executado pelos nós-sensores. A Figura 6 exibe o *BWSNET Proxy* e o seu principal conjunto de componentes. Outras funcionalidades, como por exem-

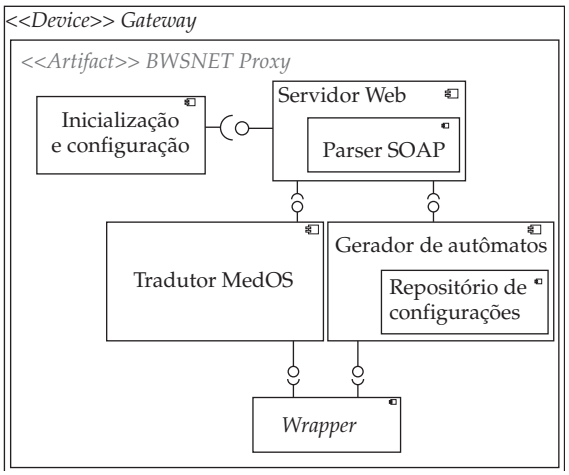

Figura 6. Principais componentes do *BWSNET Proxy*. O Tradutor MedOS é responsável por analisar as mensagens SOAP relativas à reconfiguração em tempo de execução, criar uma mensagem compreensível para a camada subjacente (o MedOS) e acionar o componente *Wrapper* para que a mensagem de reconfiguração seja, então, enviada para cada nó-sensor. O Gerador de autômatos recebe uma seleção de funcionalidades do programador por meio de uma mensagem SOAP. A partir daí determina o comportamento auto-ajustável para cada nó-sensor pela seleção do melhor autômato disponível no repositório de configurações. Em seguida, comanda o *Wrapper* para a compilação, instalação (*deployment*) e inicialização do sistema.
Figure 6. *Main components of the BWSNET Proxy. The MedOS translator is responsible for analyzing the SOAP messages related to the reconfiguration at execution time, for creating a message that is understandable to the underlying layer (the MedOS) and to turn the Wrapper component on so that the reconfiguration message is sent to each sensor node. The automata generator receives a selection of functionalities from the programmer though a SOAP message. After that, the automata generator defines the self-adjusting behavior for each sensor node, by selecting the best automatum available in the repository of configurations. After that, the generator sends an instruction to the Wrapper to perform the compilation, deployment and initialization of the system.*

plo, promover a integração da BSN com sistemas de informação de saúde do paciente ou novos serviços, também podem ser incorporadas como novos componentes ao *BWSNET Proxy*. Para isso, uma estrutura comum, denominada servidor de aplicação, foi desenvolvida. O objetivo dessa estrutura é fornecer um mecanismo leve (com baixa demanda por memória), portável entre diferentes tipos de dispositivos de *gateway* e que seja capaz de receber e tratar requisições no formato padronizado para *web services*. Atualmente os serviços disponibilizados pelo *BWSNET Proxy* resumem-se a dois componentes: o tradutor MedOS e o gerador de autômatos (Figura 6). O tradutor MedOS analisa as mensagens SOAP relativas à reconfiguração em tempo de execução, cria uma mensagem equivalente e compreensível para a camada subjacente e aciona o componente *Wrapper* para que a mensagem de reconfiguração seja, então, enviada para cada nó-sensor. O Gerador de autômatos recebe uma seleção de funcionalidades do programador por meio de uma mensagem SOAP. A partir daí determina o comportamento auto-ajustável para cada nó-sensor e comanda o *Wrapper* para a compilação, instalação e inicialização da imagem do sistema em cada nó-sensor. Esse processo é chamado de programação estática do nó-sensor.

De acordo com a hipótese motivadora para utilização de sistemas auto-ajustáveis (*software*) que serão instalados nos nós-sensores, é possível economizar energia elétrica mudando dinamicamente o monitoramento do estado de saúde do paciente (Carvalho *et al.*, 2003). Para isso é necessário reajustar os parâmetros que governam o funcionamento do sistema, como por exemplo, as interfaces de comunicação, a freqüência de operação e o desligamento de partes do sistema quando não estiverem em uso. Para sistematizar essa abordagem, tem sido utilizada uma modelagem baseada em autômatos. Os autômatos fornecem uma descrição inicial do comportamento do sistema de acordo com o objetivo de cada aplicação. Durante a programação estática, além da seleção efetuada pelo programador, o melhor autômato para uma dada aplicação é selecionado automaticamente pelo sistema para que o tempo de vida dessa aplicação seja maximizado. Para isso, uma biblioteca com diversos autômatos foi desenvolvida para diferentes situações, encontrando-se os mesmos disponíveis para o usuário durante a configuração. A Figura 7 exibe o diagrama de estados de um autômato relativo à seleção apresentada na Figura 3.

SOAB: MedOS

Na quarta e última camada, encontra-se um sistema operacional orientado pela aplicação. O MedOS é composto por: i) um conjunto de instruções (comandos) para reconfiguração do nó-sensor denominado protocolo MedOS; ii) um interpretador de comandos para o protocolo MedOS; iii) um conjunto de algoritmos representados por autômatos que descrevem o comportamento auto-ajustável de cada nó-sensor de acordo com os requisitos de cada aplicação, e iv) um

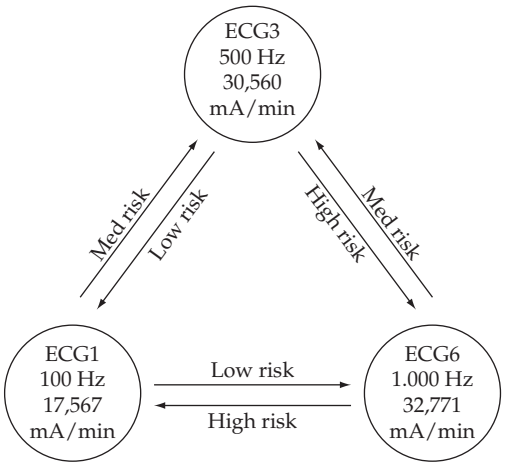

Figura 7. Diagrama de estados de um autômato relativo à seleção (programação) apresentada pela Figura 3. Este diagrama ilustra como o consumo de corrente pode variar durante a execução de cada tarefa (representada pelos estados do autômato). A freqüência de operação da CPU é definida com base na complexidade de cada tarefa (número de canais e taxa de amostragem requisitados). Além disso, uma política para desligamento do rádio-transmissor durante alguns intervalos de tempo é associada a cada tarefa. Os valores de consumo de corrente (em mA/min) apresentados neste diagrama são referentes ao *hardware* (nó-sensor) apresentado na Figura 2. As mudanças de estado são induzidas por eventos gerados por outros autômatos que analisam e classificam o estado de saúde do paciente (risco a que o mesmo está submetido) com base no resultado do próprio eletrocardiograma. Por exemplo, um evento de alto risco (*high risk*) poderia ter sido gerado para o ECG obtido de um paciente infartado em estado grave na unidade de terapia intensiva. Maiores detalhes acerca do significado dos eventos podem ser obtidos em Carvalho *et al.* (2003). ***Figure 7.*** *State diagram of an automatum that is responsible for the selection (programming) shown in Figure 3. This diagram illustrates how the consumption of electrical current can vary during the execution of each task (represented in each state of the automatum) at each instant. The frequency operation of the CPU is defined based on the complexity of the task, and a policy for turning the radio-transmitter off is associated with each task. The current consumption presented in this diagram refers to the sensor node presented in Figure 2. Further details about the meaning of the events are presented in Carvalho et al. (2003).*

conjunto de bibliotecas de código (*device drivers*) responsável pelo acionamento do *hardware*. A Figura 8 exibe o conjunto de artefatos que compõe atualmente o MedOS.

O MedOS disponibiliza bibliotecas de código em linguagem C para facilitar a utilização dos componentes (*hardware*) do nó-sensor. Os *device drivers* oferecem interfaces para programação dos sensores, da interface de comunicação e da CPU. Atualmente, os *drivers* disponibilizados pelo MedOS são utilizados para programação dos sensores, programação da interface de comunicação e seleção dos modos e ajuste da freqüência de operação da CPU para economia de energia. Os algoritmos orientados pela aplicação são implementados pela manipulação das rotinas responsáveis por dar suporte à multitarefa. Para suporte à criação de tarefas têm sido utilizadas as bibliotecas de código do sistema FreeRTOS (Barry, 2006). Essa escolha possibilita que os artefatos do MedOS possam ser desenvolvidos com maior independência do *hardware*, haja vista que existem implementações do FreeRTOS para uma grande variedade de microcontroladores e microprocessadores.

As funcionalidades providas por cada nó-sensor são associadas aos estados do autômato que é escolhido para uma dada aplicação. Para cada estado do autômato é criada uma tarefa no MedOS. Para cada tarefa são associados um estado de execução e uma prioridade. Essas informações são indispensáveis para estabelecer a ordem e a temporalidade determinada pelo autômato. Os algoritmos orientados pela aplicação manipulam os valores de prioridades e os estados de execução das tarefas com o objetivo de cumprir a meta preestabelecida pelo autômato.

O interpretador de comandos é o mecanismo responsável por prover capacidade de ajuste durante a execução do autômato (reconfiguração dinâmica). O interpretador de comandos é implementado como uma tarefa de maior prioridade no MedOS e pode ou não ser incluído durante a programação estática do sistema. A cada comando enviado pelo *BWSNET Proxy*, o interpretador de comandos deve reconhecer a validade desse comando dentro do conjunto de comandos do MedOS (protocolo MedOS) e, então, desencadear uma ação. As ações estão relacionadas ao ajuste das tarefas (manipulação dos valores de prioridade e estados de execução) durante a execução das mesmas.

Testes e Resultados

Muitas vezes a avaliação de um *software* é realizada por métodos empíricos, com base em parâmetros pertencentes ao domínio da própria aplicação, como, por exemplo, uma simples averiguação dos requisitos funcionais definidos durante projeto. Esse tipo de avaliação é impreciso e, em geral, avalia apenas os requisitos funcionais de um sistema e não fornece

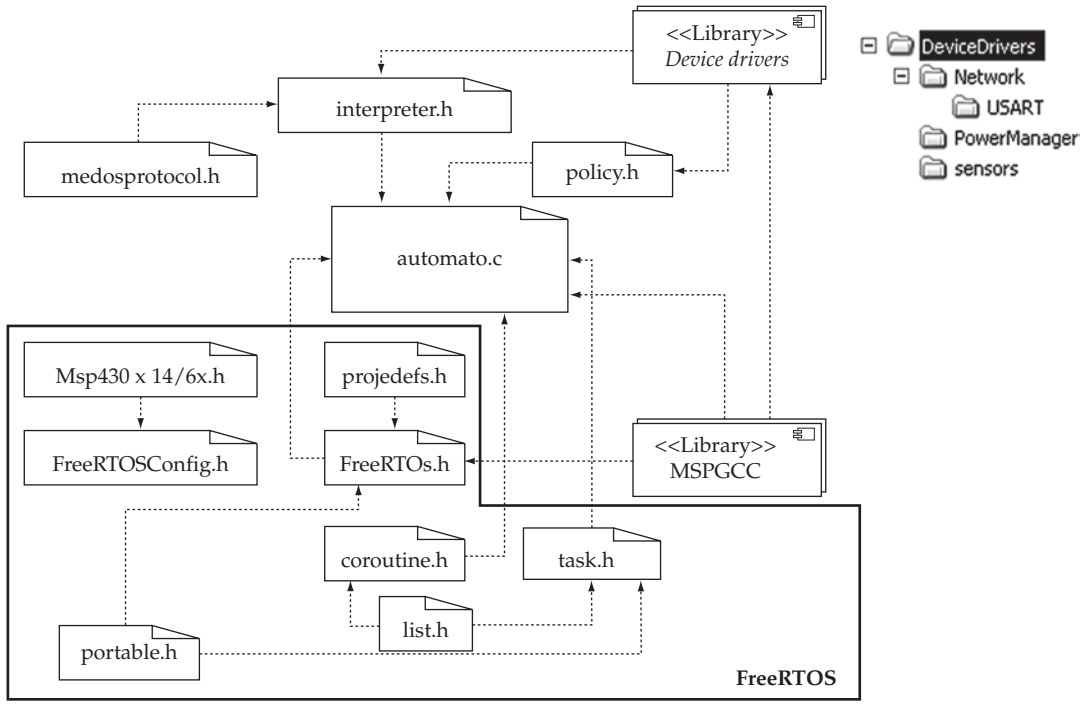

Figura 8. Principais artefatos que compõem o MedOS e suas dependências. Um dos objetivos principais da arquitetura SOAB é a geração automatizada do arquivo "automato.c". Esse arquivo corresponde à imagem de todo o sistema, que será instalada nos nós-sensores. Depois de finalizado, o arquivo "automato.c" é compilado e o arquivo resultante (executável) é instalado na memória do microcontrolador. ***Figure 8.*** *The main components of the MedOS and their dependencies. One of the main goals of the SOAB architecture is the automatic generation of the "automato.c" file. This file corresponds to the image of the entire system, which will be installed in the sensor nodes. After the deployment, the"automato.c" file is compiled and the resulting (executable) file is installed in the memory of the microcontroller.*

resultados que poderão ser utilizados para comparações. Pensando nisso, concebeu-se uma metodologia para avaliação da SOAB. Essa metodologia tem com base as normas IEEE Std. 1061/1992 – IEEE *Standard for Software Quality Metrics Methodology* (IEEE, 1993) e ISO/IEC 9126-1/2001 (ISO, 2001). Essas normas estabelecem parâmetros que fornecem o conceito de qualidade de um *software.* Os parâmetros avaliados até o momento foram a funcionalidade, a portabilidade e a eficiência.

Para avaliar o parâmetro "funcionalidade" foram executados testes de "caixa aberta (ou caixa branca)" (Staa, 2000, p. 529) com o objetivo de averiguar o correto funcionamento das unidades e dos componentes de *software*. A cada nova unidade e novo componente desenvolvido, um conjunto de testes foi executado. Os resultados obtidos desses testes foram satisfatórios quanto à acurácia e à conformidade com a modelagem preestabelecida. Os relatórios desses testes foram incorporados ao código-fonte dos programas como parte dos comentários.

Quanto à portabilidade, a definição de uma arquitetura em camadas funcionais e independentes facilita a modificabilidade, subparâmetro definido pela ISO, em 2001. A utilização de tecnologias de desenvolvimento, como por exemplo, Java, SOAP e FreeRTOS, promove a independência do *hardware* e do *software* e facilita a instalação do sistema. Na prática, foram executados testes em plataformas Windows e Linux para as três primeiras camadas da SOAB. Os resultados obtidos nos testes foram satisfatórios quanto a: i) exatidão dos resultados obtidos em ambas as plataformas; ii) interoperabilidade entre os quatro subsistemas (camadas); iii) independência de plataforma, resguardando a diferença entre os tipos de fontes gráficas disponibilizadas pelo Linux em relação aos tipos disponibilizados pelo Windows®, e iv) facilidade de modificação de elementos de cada camada em função da independência entre as mesmas.

Eficiência é sinônimo de economia de recursos e/ou de tempo (IEEE, 1993; ISO, 2001). Para avaliar esse parâmetro foi aplicado um teste de "caixa fechada (ou caixa preta)" (Staa, 2000, p. 529) com o objetivo de

quantificar o tempo de resposta para a programação estática do sistema. Esse teste é também conhecido pelo jargão da Ciência da Computação como "teste de carga" ou, em inglês, *benchmarking.* Entende-se que o tempo de resposta é a principal métrica relacionada à eficiência de uma metodologia de programação de uma BSN através da Internet. Neste trabalho, o tempo de resposta corresponde ao tempo para completar todo o ciclo de programação do sistema, que inclui o tempo gasto para que o servidor *Proxy* receba a requisição do serviço (mensagem SOAP). Soma-se a isso o tempo de processamento das mensagens de serviço, que corresponde ao processamento das mensagens SOAP com o objetivo de decodificar o autômato definido pelo programador, bem como as operações relacionadas a esse autômato, que podem ser as operações de compilar, instalar e executar.

Para emular o comportamento da *BWSNET Configuration Tool* quando esta interage com o servidor *Proxy* pela Internet foi utilizada a ferramenta Httperf (Mosberger e Jin, 1998), executada em um PC Pentium IV 2,4 GHz com 512 MB de memória RAM. O Httperf é um gerador de requisições capaz de fornecer relatórios estatísticos precisos a respeito do desempenho de aplicações distribuídas, incluindo, por exemplo, parâmetros como o tempo de resposta, a taxa de conexões e a taxa de bloqueios, entre outros. O *BWSNET Proxy* foi executado em um PC Pentium III 700 MHz com 128 MB de memória RAM, com o intuito de emular o desempenho obtido quando essa ferramenta estivesse sendo executada por um dispositivo móvel de menor poder computacional. Para o enlace, foi utilizada uma rede ETHERNET 10/100baseT sem isolamento de tráfego para que a simulação se aproximasse ao máximo do modelo real. A Figura 9 ilustra o ambiente operacional e os artefatos utilizados nos testes.

Para os testes foram criadas três tarefas gerenciadas pelo MedOS: i) ECG com única derivação e freqüência de amostragem de 100 Hz, ii) ECG com três derivações e freqüência de amostragem de 500 Hz, e iii) ECG com seis derivações e freqüência de amostragem de 1.000 Hz. Para cada tarefa foi incluído um filtro FIR (*Finite Impulse Response*) para eliminar a variação da linha de base (*baseline wandering*), a interferência de 60 Hz e o ruído branco (Raju, 2005). Esse filtro acrescenta aproximadamente 1.000 bytes ao tamanho do código de cada tarefa. A Tabela 1 resume os principais resultados obtidos. A duração do ciclo de progra-

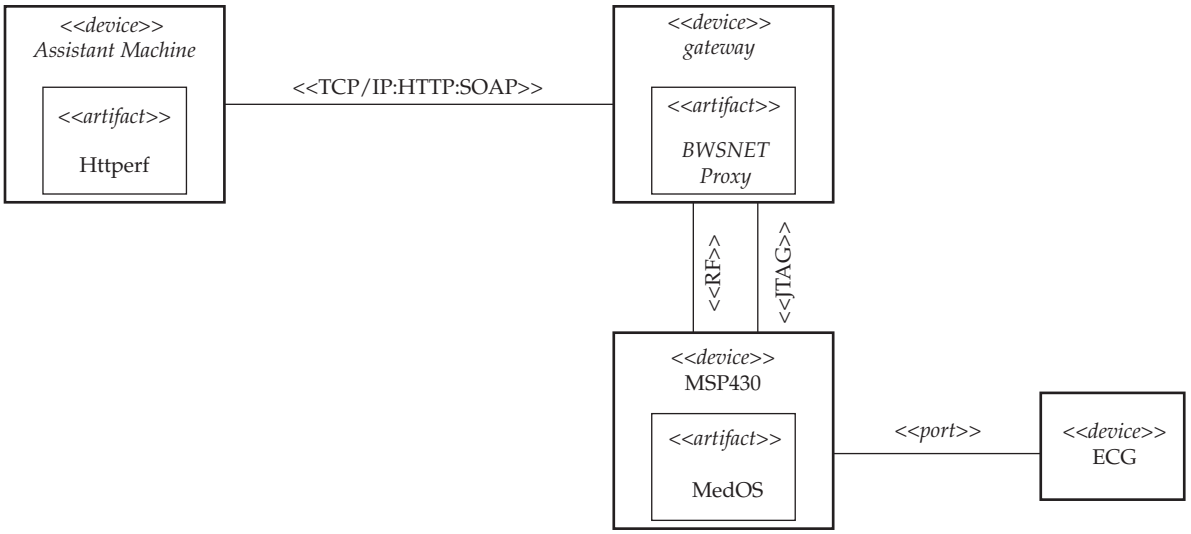

Figura 9. Disposição dos artefatos e dispositivos para a realização do teste de carga. Para emular o comportamento da *BWSNET Configuration Tool* foi utilizada a ferramenta Httperf (Mosberger e Jin, 1998), executada em um PC Pentium IV 2,4 GHz com 512 MB de memória RAM. O *BWSNET Proxy* foi executado em um PC Pentium III 700 MHz com 128 MB de memória RAM, com o intuito de emular o desempenho obtido quando essa ferramenta estivesse sendo executada por um dispositivo móvel de menor poder computacional. Para o enlace, foi utilizada uma rede ETHERNET 10/100baseT sem isolamento de tráfego, de forma que a simulação se aproximasse ao máximo do modelo real. ***Figure 9.*** *Disposition of the artifacts and devices for performing the benchmarking. The Httperf tool (Mosberger and Jin, 1998) was used to emulate the behavior of the BWSNET Configuration Tool, on a PC Pentium IV 2.4 GHz, 512 MB of RAM memory. The BWSNET Proxy run on a PC Pentium III 700 MHz, 128 MB of RAM memory, in order to emulate the performance that is attained during the execution of this tool in a mobile device with lower computation power. An Ethernet 10/100baseT without traffic isolation was used for the data link, so that the simulation would be more realistic.*

Tabela 1. Tempo de resposta obtido para programação do nó-sensor de ECG pela Internet. Para cada tarefa foi incluído um filtro FIR (*Finite Impulse Response*) para eliminar a variação da linha de base, a interferência de 60 Hz e o ruído branco (Raju, 2005). Esse filtro acrescenta aproximadamente 1.000 *bytes* ao tamanho do código de cada tarefa. A duração do ciclo de programação do sistema corresponde ao tempo de resposta médio arredondado (em segundos) obtido mediante 10 interações entre o Httperf e o *BWSNET Proxy.* ***Table 1.*** *Response time obtained for the reprogramming of the EKG sensor node using the Internet. For each task, an FIR filter has been included in order to eliminate the baseline wandering, the 60 Hz interference and undesirable white noise (Raju, 2005). This filter adds approximately 1,000 bytes to the code size of each task. The duration of the programming cycle of the system corresponds to the approximate mean response time (in seconds) obtained after 10 interactions between the Httperf and the BWSNET Proxy.*

Tamanho do código (aproximado)	Duração aproximada do ciclo de programação	Descrição do conteúdo do código binário
5 kB	4 s	Apenas o FreeRTOS
6 kB	5 s	FreeRTOS e MedOS
9 kB	10 s	FreeRTOS, MedOS e três tarefas: ECG1/100 Hz, ECG3/500 Hz ECG6/1.000 Hz

mação do sistema corresponde ao tempo de resposta médio arredondado (em segundos) obtido mediante 10 interações entre o Httperf e o *BWSNET Proxy*.

Discussão

Em relação à eficiência avaliada, os valores para os tempos de resposta tendem a crescer em função da complexidade do código e do nível de inteligência embutido no nó-sensor. Esse fator também pode influir na quantidade de reprogramações diárias que se fazem necessárias. Na prática, os valores apresentados na Tabela 1 devem ser considerados como o período de tempo em que o paciente terá que aguardar com o sistema inativo e conectado ao *BWSNET Proxy* por meio da interface JTAG. Obviamente, com a utilização de sistemas com maior capacidade de processamento (*hardware*) e de tecnologias de rede com maiores taxas de transmissão, os tempos de resposta, em valores absolutos, deverão ser menores.

De forma geral, o estado de saúde do paciente pode definir a freqüência da reprogramação do sistema, embora a possibilidade de reprogramação por falhas de *software* ou do *hardware* possa, eventualmente, influir nesse parâmetro. Em muitos casos, a reprogramação pode resumir-se a um reajuste do *software* efetuado durante a execução das tarefas. Para isso, foi desenvolvida uma abordagem complementar à programação estática e que permitirá a reprogramação sem a necessidade de interrupções no funcionamento do sistema nem de conexões com cabos. A metodologia de reconfiguração em tempo de execução (*runtime reconfiguring methodology*) possibilita que os programadores possam atuar tanto no âmbito da rede de sensores quanto no âmbito dos nós-sensores (Barbosa *et al.*, 2006).

No âmbito da rede, é possível alterar a importância de cada nó-sensor de acordo com as necessidades de cada aplicação. Esse parâmetro é definido como *Sensor QoS Level* (Figura 10), e refere-se à importância da informação coletada por determinado sensor. Nesse caso, o valor 0,8 significa que a informação de interesse advinda de um determinado nó-sensor deve ser pontuada com 80% do valor máximo possível. O ajuste do *Sensor QoS Level* está diretamente ligado ao ajuste da largura de banda e da taxa de transmissão utilizadas pelo nó-sensor a cada instante.

No âmbito do nó-sensor, é associado um valor de prioridade a cada tarefa executada. A alteração dos valores das prioridades pode alterar a seqüência, quando nenhum evento ocorrer, e principalmente o período de tempo (*time-slice*) em que cada tarefa deverá utilizar a CPU. Na Figura 10, são apresentados três valores de prioridades: *Low Risk*, *Med Risk* e *High Risk*, que são mapeados respectivamente em três valores de prioridades pelo MedOS: Priority 2, Priority 3 e Priority 4. Além de alterar os valores das prioridades o programador poderá suspender (*suspend*) tarefas em razão de outras de maior interesse.

Conclusão

Este artigo apresenta um modelo para programação de BSNs que permite que os profissionais não-especializados em tecnologias da informação realizem a tarefa de programação desses sistemas com mais transparência, de forma que essas atividades não sejam inconvenientes aos pacientes ou aos próprios programadores. A transparência aqui apresentada é avaliada por parâmetros obtidos de normas internacionais e não puramente de requisitos não-funcionais obtidos de outros trabalhos. Por exemplo, por meio de um teste de carga, se propôs mensurar quão inconveniente o processo de programação da BSN pode se tornar.

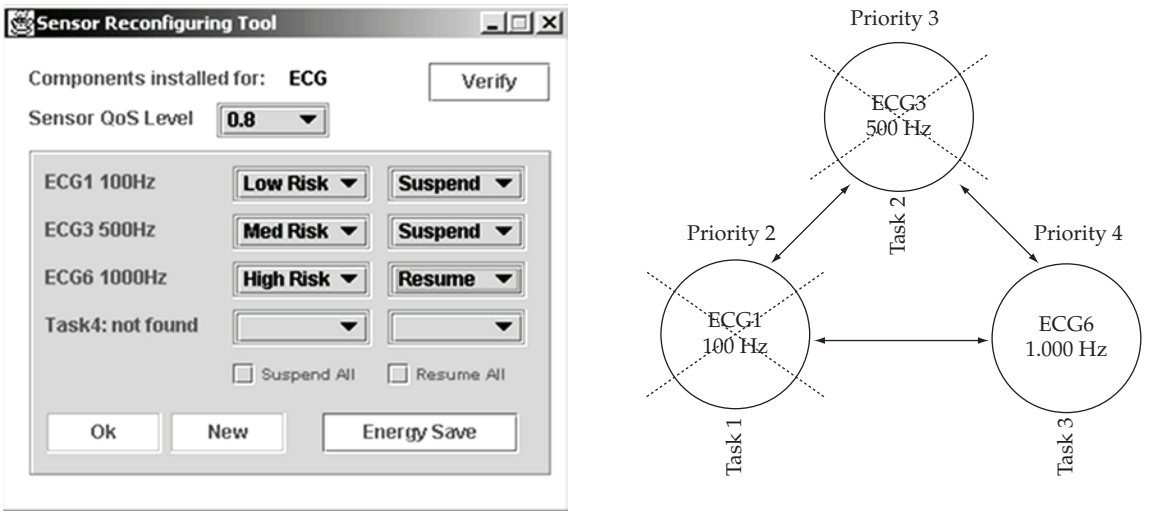

Figura 10. O processo de reconfiguração remota e suas implicações no nó-sensor. Nesta figura são apresentados três valores de prioridades: *Low Risk*, *Med Risk* e *High Risk*, que são mapeados, respectivamente, em três valores de prioridades pelo MedOS: Priority 2, Priority 3 e Priority 4. Além de alterar os valores das prioridade, o programador poderá suspender (*suspend*) e reinicializar (*resume*) as tarefas quando necessário. A alteração dos valores das prioridades pode alterar a seqüência e, principalmente, o período de tempo (*time-slice*) em que cada tarefa deverá utilizar a CPU do nó-sensor.
Figure 10. *The process of remote reconfiguration and its implications on the sensor node. In this figure, three priority values are presented: Low Risk, Med Risk and High Risk, and they are mapped, respectively, into three priority values by the MedOS: Priority 2, Priority 3 and Priority 4. Besides altering the values of the priorities, the programmer can suspend and resume the tasks when needed. The changing in the values of the priorities may alter the sequence and, especially, the time slice during which each task should use the CPU of the sensor node.*

Uma arquitetura de *software* em camadas permite ainda que a transparência possa ser alcançada de maneira modular, resguardando a portabilidade quanto às possíveis modificações das camadas subjacentes, inclusive do próprio *hardware*.

Duas inovações tecnológicas destacam-se neste trabalho. A primeira é a proposta de uma interface gráfica para programação de redes de sensores para o corpo humano. Até o momento, nenhum relato que trate desse assunto foi encontrado na literatura especializada, e essa ferramenta é um primeiro passo na tentativa de mudança de paradigma. Espera-se que os profissionais de saúde possam se tornar os verdadeiros programadores e mantenedores das redes de sensores. Outra inovação é a aplicação do conceito de multitarefa em redes de sensores para o corpo humano. A multitarefa permite que os nós-sensores sejam dotados de funcionalidades mais complexas e, ao mesmo tempo, mais interativas. Com isso, é possível manter e até aumentar a capacidade de operação autônoma dos nós-sensores, diminuindo a necessidade de centralização de decisões nos *gateways* (potenciais gargalos). Além disso, esse modelo de programação assegura a possibilidade de intervenções humanas quando há mudanças de interesses. Até o momento, também não foi encontrado nenhum registro a respeito da utilização da multitarefa em redes de sensores para o corpo humano.

Trabalhos futuros serão concentrados: i) em testes de usabilidade, ii) na implementação de mecanismos para segurança e autenticação, principalmente para as intervenções remotas pela Internet, e iii) em novos algoritmos baseados em prioridades e orientados pela aplicação que possam ser utilizados para escalonamento mais eficiente das funcionalidades embutidas nos nós-sensores.

Agradecimentos

Ao CNPq, pelo financiamento por meio do Programa de Apoio à Pesquisa, Desenvolvimento e Inovação em Tecnologia da Informação, PDI – TI, Chamada Conjunta MCT/SEPIN-FINEP-CNPq 01/2002, e por meio de bolsa de produtividade em pesquisa (processo 301036/2006-3).

Referências

Apache - Xerces Java Parser 1.4.4. (2006), In: http://xerces.apache.org/xerces-j/, acessado em 27 set.

Bajzat, R. (2006), "Thinlet", In: http://thinlet.sourceforge.net/home.html, acessado em 27 set.

Barbosa, T.M.G.A., Sene Jr., I.G., Carvalho, H.S., da Rocha, A.F., Nascimento, F.A.O., Camapum, J.F. (2006a), "Application-oriented programming model for sensor networks embedded in the human body", In: *Proceedings of Annual International Conference IEEE Engineering in Medicine and Biology Society (EMBC)* [*Engineering Revolution In BioMedicine*], Nova Iorque, v. 1, p. 6037-6040, 30 ago-3 set.

Barbosa, T.M.G.A., Sene Jr., I.G., Carvalho, H.S., da Rocha, A.F., Nascimento, F.A.O. (2006b), "Programação de uma rede de sensores para o corpo humano por meio de uma interface gráfica", In: *Anais do X Congresso Brasileiro de Informática em Saúde (CBIS)*, Florianópolis, 14-18 out.

Barbosa, T.M.G.A., Gutiérrez, E.M., França, R.D., Carvalho, H.S., da Rocha, A.F. (2006c), "Desenvolvimento de uma rede de sensores sem fios para o monitoramento biomédico", In: *Anais do X Congresso Brasileiro de Informática em Saúde (CBIS)*, Florianópolis, 14-18 out.

Barry, R. (2006), "FreeRTOS", In: http://www.freertos.org/, acessado em 27 set.

Carvalho, H.S., Zuquim, A., Loureiro, A.A., Coelho-Jr., C., Vieira, M., Vieira, L.F, Vieira, A., Fernandes, A., Silva Jr., D., da Mata, J., Nacif, J. (2003), "Efficient power management in real-time embedded systems", In: *Proceedings of 9th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'03)*, Lisboa, v. 1, p. 496-505, 16-19 set.

Carvalho, H.S. (2005), *Data fusion implementation in sensor networks applied to health monitoring*, Tese de Doutorado, Programa de Ciência da Computação, DCC/UFMG, Belo Horizonte, 158 p., jan.

Fröhlich, A.A. (2001), *Application-Oriented Operating Systems*, Sankt Augustin: GMD - Forschungszentrum Informationstechnik, 200 p.

Gay, D., Levis, P., von Behren, R., Welsh, M., Brewer, E., Culler, D. (2003), "The nesC Language: a holistic approach to network embedded systems", In: *Proceedings of ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation*, San Diego, v. 38, n. 5, 9-11 jun.

IEEE – Institute of Electrical and Electronics Engineers (1993), *Std 1061-1992 IEEE standard for a software quality metrics methodology*, Nova Iorque: IEEE Computer Society Press, 96 p.

ISO – International Organization for Standardization (2001), *9126-1:2001 Software engineering – Product quality – Part 1: quality model*, Suíça: ISO Press, 25 p.

Jovanov, E. (2006), "WHMS – Wearable Health Monitoring System", In: www.ece.uah.edu/~jovanov/whrms/, acessado em 27 set.

Mosberger, D., Jin, T. (1998), "Httperf: a tool for measuring Web server performance", *Performance Evaluation Review*, v. 26, n. 3, In: www.hpl.hp.co.uk/research/linux/httperf/docs.php, acessado em 27 set. 2006.

NASA/NCI – National Aeronautics and Space Administration / National Cancer Institute (2006), "Biomolecular Sensor Development", In: http://nasa-nci.arc.nasa.gov/, acessado em 27 set.

Olimex (2006), "MSP430-P149 Development Board", In: www.olimex.com/dev/index.html, acessado em 27 set.

Paradiso, J., Morris, J.S., Benbasat, Y.A., Pham, H., Oey, J., Dan Lovell, S., Asmussen, E., Bramsen, P. (2006), "Wireless wearable system for gait evaluation", In: www.media.mit.edu/resenv/gaitshoe.html, acessado em 27 set.

Raju, M. (2005), "Heart Rate and EKG Monitor using the MSP430FG439", Texas Instruments Technical Report SLAA280, In: http://focus.ti.com/lit/an/slaa280/slaa280.pdf#search=%22EKG%20%2B%20MSP430%20%2B%20RAJU%22, acessado em 27 set.

Sankai, Y., Suzuki, K., Kawamura, Y., Hayashi, T., Sakurai, T., Hasegawa, Y. (2005), "Intention-based walking support for paraplegia patient", In: *Proceedings of Conference on Systems, Man and Cybernetics*, Waikoloa-Hawaii, p. 2707-2713, v. 3, out.

SP – SPARK FUN electronics (2006), "Bluetooth Modem – BlueSMiRF", In:http://www.sparkfun.com/commerce/product_info.php?products_id=582, acessado em 27 set.

Staa, A. (2000), *Programação modular: desenvolvendo programas complexos de forma organizada e segura*, 1ª ed. Rio de Janeiro: Campus, 690 p.

SUN (2006), "The Source for Java Developers", In: http://java.sun.com/, acessado em 27 set.

TI – Texas Instruments (2006), "MSP430F149" In: http://focus.ti.com/docs/prod/folders/print/msp430f149.html, acessado em 27 set.

TinyOS (2006), In: www.tinyos.net/, acessado em 27 set.

TJT – The JavaTM Tutorials (2006), "Trail: The Reflection API", In: http://java.sun.com/docs/books/tutorial/reflect/index.html, acessado em 27 set.

W3C – Web Services Architecture (2004), In: www.w3.org/TR/ws-arch/, acessado em 27 set.

W3C – Web Services Description Language (WSDL) (2006), Version 2.0 Part 0: Primer, In: www.w3.org/TR/2006/CR-wsdl20-primer-20060327/, acessado em 27 set.

Welsh, M. (2006), "CodeBlue: Wireless Sensor Networks for Medical Care", In: www.eecs.harvard.edu/~mdw/proj/codeblue/, acessado em 27 set.

Yang, G. (2006), *Body Sensor Networks*, Springer, 500 p.