UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

UMA ARQUITETURA DE REDES DE SENSORES DO CORPO HUMANO

TALLES MARCELO GONÇALVES DE ANDRADE BARBOSA

ORIENTADOR: ADSON FERREIRA DA ROCHA
CO-ORIENTADOR: HERVALDO SAMPAIO CARVALHO

TESE DE DOUTORADO EM ENGENHARIA ELÉTRICA

PUBLICAÇÃO: PPGENE.TD – 021/08
BRASÍLIA/DF: FEVEREIRO – 2008

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

UMA ARQUITETURA DE REDES DE SENSORES DO CORPO HUMANO

TALLES MARCELO GONÇALVES DE ANDRADE BARBOSA

TESE DE DOUTORADO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA, COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM ENGENHARIA ELÉTRICA.

APROVADA POR:

Prof. Adson Ferreira da Rocha, PhD (ENE-UnB)
(Orientador)

Prof. Ricardo Zelenovsky, DSc (ENE-UnB)
(Examinador Interno)

Prof. Geovany Araújo Borges, Docteur (ENE-UnB)
(Examinador Interno)

Prof. Antonio Alfredo Ferreira Loureiro, PhD (DCC-UFMG)
(Examinador Externo)

Prof. José Olímpio Ferreira, DSc (DC-UCG)
(Examinador Externo)

BRASÍLIA/DF, 14 DE FEVEREIRO DE 2008.

FICHA CATALOGRÁFICA

BARBOSA, TALLES MARCELO GONÇALVES DE ANDRADE
Uma Arquitetura de Redes de Sensores do Corpo Humano [Distrito Federal] 2008.
188 p., 210 x 297 mm (ENE/FT/UnB, Doutor, Engenharia Elétrica, 2008).
Tese de Doutorado – Universidade de Brasília. Faculdade de Tecnologia. Departamento de Engenharia Elétrica.
1. Rede de Sensores 2. Corpo Humano 3. Arquitetura de Software
I. ENE/FT/UnB II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

BARBOSA, T. M. G de A. (2008). Uma Arquitetura de Redes de Sensores do Corpo Humano. Tese de Doutorado em Engenharia Elétrica,
Publicação PPGENE.TD-021/08, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 188 p.

CESSÃO DE DIREITOS

AUTOR: Talles Marcelo Gonçalves de Andrade Barbosa
TÍTULO: Uma Arquitetura de Rede de Sensores do Corpo Humano.
GRAU: Doutor ANO: 2008

_____

Talles Marcelo Gonçalves de Andrade Barbosa
Rua T-44, 358. Setor Bueno
74210-150 Goiânia-GO. Brasil.

DEDICATÓRIA

Aos meus pais, José Roldão e Ivonnyr

"Há pessoas que desejam saber só por saber, e isso é curiosidade; outras, para alcançarem fama, e isso é vaidade; outras, para enriquecerem com a sua ciência, e isso é um negócio torpe; outras, para serem edificadas, e isso é prudência; outras, para edificarem os outros, e isso é caridade."

São Tomás de Aquino

AGRADECIMENTOS

Ao professor Adson Ferreira da Rocha pela dedicação, amizade, paciência e generosidade. Durante a nossa convivência pude conhecer um grande exemplo de humildade, perseverança e lealdade. Agradeço-o também pelas inúmeras orientações. Elas contribuíram muito para o desenvolvimento deste trabalho e para minha formação profissional.

Ao professor Hervaldo Sampaio Carvalho, o líder da área de pesquisa em Redes de Sensores do Corpo Humano na UnB, por sua valiosa orientação, e pela oportunidade de trabalhar com um tema vibrante, e de muitas possibilidades, integrando-me ao seu grupo de pesquisa. Também, por possibilitar que este trabalho tenha se apoiado no projeto MiLAN (*Middleware Linking Applications and Networks*) iniciado por ele e pela professora Wendi Heinzelman, na Universidade de Rochester. Além disso, agradeço pelas muitas orientações que possibilitaram interfacear a área da Saúde, a Computação e a Engenharia Elétrica.

Agradeço aos professores Geovany Araújo Borges e Antonio Alfredo Ferreira Loureiro pelas sugestões para melhoria do trabalho. Isso contribuiu decisivamente.

Agradeço também aos professores José Olímpio Ferreira e Ricardo Zelenovsky por se disponibilizarem a participar como membros da banca de avaliação deste trabalho.

Ao professor Francisco Assis de Oliveira Nascimento, pela acolhida no departamento de Engenharia Elétrica e pelo despertar para a área de Processamento de Sinais.

À Universidade Católica de Goiás, pelo apoio financeiro durante os três primeiros anos de doutoramento. Em especial, ao professor José Nicolau Reck (pró-reitor de pós-graduação e pesquisa) e ao professor Luiz Carlos de Sousa (diretor do departamento de computação à época) pelo empenho em viabilizar este projeto.

Agradeço aos colegas da Universidade Católica de Goiás (UCG) pelo apoio durante todo o período de doutoramento. Em especial, aos meus amigos Piero Martelli, José Olímpio Ferreira, Olegário Corrêa da Silva Neto, Cláudio Martins Garcia e Iwens Gervásio Sene Júnior, companheiro durante toda jornada.

A todos os alunos que participaram deste projeto. Da Universidade Católica de Goiás, Alexandre Bellezi José, Alexandre da Silva, Daniel Sanches, Paula Melo e Kátia Kelvis. Da Universidade de Brasília, Liana Castro, Eduardo Gutiérrez, Bruno Martinello, Leandro Barbosa, Bruno Barbosa e Daniel Melo.

Agradeço ainda ao meu irmão Marco pelas visitas e telefonemas de apoio.

Agradeço a minha namorada Fernanda pelo carinho e pela paciência durante todos esses anos. Sem isso, não teria alcançando a vitória.

Aos meus pais José Roldão e Ivonnyr pelo amor e dedicação durante toda minha vida.

Sobretudo, agradeço a Deus por estar sempre conduzindo meus passos.

RESUMO

UMA ARQUITETURA DE REDE DE SENSORES DO CORPO HUMANO

Autor: Talles Marcelo Gonçalves de Andrade Barbosa
Orientador: Adson Ferreira da Rocha
Co-orientador: Hervaldo Sampaio Carvalho
Programa de Pós-Graduação em Engenharia Elétrica
Brasília, 14 fevereiro de 2008

As Redes de Sensores do Corpo Humano (RSCH) devem ser projetadas para operar de maneira autônoma. Por outro lado, devem oferecer mecanismos que remetam o controle aos profissionais da saúde. Um grande desafio no projeto de RSCH é oferecer de forma adequada (com transparência) acesso às configurações internas da rede e dos sensores, sem excluir a capacidade de operação autônoma desses sistemas. A tarefa de configuração de RSCH é usualmente realizada por desenvolvedores especializados, profissionais da Computação. Entretanto, para que essa tecnologia se torne clinicamente viável, é necessário que os próprios profissionais da área de saúde possam fazê-la. Esses profissionais são os responsáveis legais pelas decisões acerca do monitoramento dos pacientes, mesmo que essas decisões tenham sido geradas pelo próprio sistema. Esta tese discute os pressupostos supracitados e propõe um modelo capaz de acomodar essas necessidades. Dois conceitos relacionados à programação de RSCH são tratados neste trabalho: *(i)* programação (em tempo de compilação) e *(ii)* configuração (em tempo de execução). A programação refere-se à definição dos artefatos de software e algoritmos que são embutidos dentro dos sensores. Em RSCH a inclusão dessa funcionalidade requer uma interface para programação adequada aos profissionais da saúde e também de compiladores inteligentes. O compilador inteligente é um conceito novo apresentado nesta tese. Tem como objetivo aumentar a eficiência no uso dos sensores, considerando os requisitos da aplicação, os recursos do hardware e, principalmente, o conhecimento especialista para formulação das políticas que organizam o funcionamento do sistema. Como exemplo, a inclusão de mecanismos e políticas para tratar da economia de energia podem ser ajustadas por essas estruturas. A configuração refere-se à capacidade de ajuste do sistema sem a necessidade de reiniciar o hardware. Esse conceito possibilita maior interatividade entre o profissional da saúde e o sistema. Como requisito, os sensores precisam de mecanismos que possibilitem maior controle acerca das tarefas executadas. Uma possível solução é a utilização de estruturas de dados que possibilitem a aplicação do conceito de multitarefa nos sensores. Como contribuição maior é apresentada uma arquitetura de software denominada SOAB (*Software Architecture for Body-worn Sensor Networks Project*). A arquitetura SOAB é constituída por quatro camadas independentes: *(i)* uma interface para programação; *(ii)* um *middleware* para interconexão da RSCH com a Internet; *(iii)* um servidor para execução dos serviços solicitados pelos usuários e *(iv)* um sistema operacional com suporte para multitarefa, que será embutido nos nós sensores. Esse sistema operacional foi chamado MedOS. Tem como uma de suas funcionalidades aumentar o tempo de funcionamento dos nós sensores, promovendo a redução do consumo de energia elétrica por meio do escalonamento de tarefas com base em políticas adaptadas para aplicações biomédicas.

Palavras-chaves: redes de sensores do corpo humano, programação, configuração, compilador inteligente, multitarefa.

ABSTRACT

A BODY SENSOR NETWORKS ARCHITECTURE

Author of the doctoral dissertation: Talles Marcelo Gonçalves de Andrade Barbosa
Advisor: Adson Ferreira da Rocha
Co-advisor: Hervaldo Sampaio Carvalho
University of Brasilia (Programa de Pós-Graduação em Engenharia Elétrica)
Brasilia (Brazil), February 14th, 2008

Body Sensor Networks (BSNs) must be designed to work autonomously. On the other hand, they must provide mechanisms that allow their control by healthcare personnel, as the clinical assessment of these professionals should always be the basis for the programming strategy. A great challenge in BSN software design is to provide, in a transparent way, access to the internal configurations of the networks and their sensor nodes without excluding their capability for autonomous operation. In order to provide such feature, specific models are needed in the design of software architectures for BSN's, which include programmability as one of their functional requisites. The success of this approach can lead to a paradigm shift, since healthcare professionals will be able to act as the actual programmers and maintainers of the BSN. The programming 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 program the system. This thesis discusses the hypothesis abovementioned and proposes a model that will lead to the achievement of these objectives. Two important concepts related to programmability in BSNs are treated in this work: (i) deployment-time programmability and (ii) runtime set-up. The deployment-time programmability refers to the definition of software artifacts and algorithms that are embedded in the sensor node. In BSN's, the inclusion of this functionality requires a programming interface that is suitable for healthcare personnel, as well as intelligent compilers. Intelligent compilers is a new concept presented in this thesis and its main purpose is to increase the effectiveness of the system's use, considering the application's requirements, the hardware possibilities and mainly the specialist's knowledge to increase the applications lifetime. Consequently, it is possible to maintain the capability for autonomous operation of the BSN and still offer tools that can be used for people with little grasp on programming languages for programming of these systems. As an example, the inclusion of mechanisms and policies for energy saving could be treated by these structures. The runtime set-up refers to the capability for adjustments in run-time. The BSN should provide interactivity between the healthcare professional (the BSN manager) and the system. As a requisite, sensor nodes need mechanisms that allow a better control of the tasks that are being run. A possible solution is the use of data structures that allow preemptive multitasking. The main contribution of this work is the proposal of a software architecture named SOAB (Software Architecture for Body-worn Sensor Networks Project). The SOAB architecture is composed of four independent layers: (i) a programming interface; (ii) middleware for interconnecting BSN's to the Internet; (iii) a server for processing users' requests; (iv) a multitasking operating system. 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.

Keywords – body sensor networks, deployment time programmability, runtime set-up, intelligent compilers, multitasking.

SUMÁRIO

LISTA DE FIGURAS

Pg.

LISTA DE TABELAS

LISTA DE ABREVIAÇÕES

4G	*Fourth Generation*
A/D	Analógico/Digital
API	*Application Programming Interface*
B3B	*Beyond the third generation*
BNEP	*Bluetooth Networking Encapsulation Protocol*
BWSNET	*Body-worn Sensor Networks*
CMOS	*Complementary Metal Oxide Semiconductor*
CPLD	*Complex Programmable Logic Device*
CPU	*Central Processing Unit*
DC-DC	*Direct Current – Direct current*
DCO	*Digitally Controlled Oscillator*
DPM	*Dynamic Power Management*
DSV	*Dynamic Scaling Voltage*
ECG	EletroCardioGrama
EEG	EletroEncefaloGrama
EMG	EletroMioGrama
EPOS	*Embedded Parallel Operating System*
EPSM	*Extended Power State Machine*
FIFO	*First In First Out*
FPGA	*Field Programmable Gate Array*
GPIO	*General Purpose Input/Output*
GPL	*General Public License*
GSR	*Galvanic Skin Resistance*
HAL	*Hardware Abstraction Layer*
HAN	*Home Area Network*
HIS	*Healthcare Information System*
HL7	*Health Level 7*
HRV	*Heart Rate Variability*
HTTP	*Hypertext Transfer Protocol*
IDE	*Integrated Development Environment*
JTAG	*Joint Test Action Group*

LPU	*Local Processing Unit*
MiLAN	*Middleware Linking Applications and Networks*
MIPS	*Million Instruction per Second*
NASA	*National Aeronautics and Space Administration*
NCI	*National Cancer Institute*
OSM	*Object State Model*
PC	*Personal Computer*
PDA	*Personal Digital Assistant*
PS	*Personal Server*
PSM	*Power State Machine*
QoS	*Quality of service*
RAM	*Random Access Memory*
RF	Rádio Freqüência
RISC	*Reduced Instruction Set Computer*
RPC	*Remote Procedure Call*
RSCH	Redes de Sensores para monitoramento do Corpo Humano
RSSF	Redes de Sensores Sem Fios
SDP	*Session Description Protocol*
SLP	*Service Location Protocol*
SO	Sistema Operacional
SOA	*Software Oriented Architecture*
SOAB	*Software Architecture for Body-Worn Sensor Networks Project*
SOAP	*Simple Object Access Protocol*
SoC	*System-on-a-chip*
TCB	*Task Control Block*
TCP/IP	*Transmission Control Protocol /Internet Protocol*
WBAN	*Wireless Sensor Body Area Network*
WHMS	*Wearable Health Monitoring Systems*
WSDL	*Web Service Description Language*
WWRF	*Wireless World Research Forum*
XML	*eXtensible Markup Language*

1 – INTRODUÇÃO

Neste capítulo são apresentados os motivadores para o desenvolvimento de uma arquitetura de redes de sensores do corpo humano. A programação e a configuração desses sistemas foram os objetos de estudo deste trabalho de doutorado. São destacadas também as contribuições científicas, a estrutura e o conteúdo dos capítulos restantes.

1.1 – AS REDES DE SENSORES

Nos últimos anos o grande desenvolvimento dos sistemas de comunicação, a miniaturização e a evolução tecnológica dos sistemas de hardware permitiram o desenvolvimento de novas aplicações. As redes de sensores são exemplos dessas novas aplicações. As aplicações que utilizam redes de sensores se caracterizam pela presença de sensores (nós) interconectados por uma rede de comunicação, usualmente sem fios, com limite de alcance do sinal e limitação de fonte de energia, por serem operadas por baterias. Trata-se de um sistema distribuído com sérias restrições para implementação (CARVALHO, 2005).

As Redes de Sensores Sem Fios (RSSF), em inglês *Wireless Sensor Networks* (WSN), exigem técnicas para tratamento de falhas e que promovam adaptação a condições ambientais diversas para que o tempo de funcionamento (sobrevida) desses sistemas seja o mais longo possível. Por exemplo, em muitos cenários a substituição ou a recarga das baterias é inviável, em termos de tempo (pode haver centenas de sensores) e acesso físico (sensores podem estar implantados no corpo humano). Além disso, é desejável que uma RSSF possa crescer em escala de forma incremental, isto é, pela adição de novos nós sensores e/ou novas funcionalidades (*extensibility*), ou pelo aumento do volume da informação manipulada (*scalability*). Assim, um possível aumento da concorrência por recursos do sistema não pode levar a perda da eficácia.

A comunidade científica vem trabalhando no desenvolvimento de sistemas que suportem tais características. Entretanto, uma RSSF é um sistema dependente da aplicação, ou seja, os requisitos que devem influenciar o projeto e a implementação desses sistemas advêm principalmente da função a que se destinam. Por exemplo, as Redes de Sensores para o

monitoramento do Corpo Humano (RSCH) [1] representam um domínio de possíveis aplicações para as RSSF.

O monitoramento ininterrupto e minimamente obstrutivo da saúde humana por meio de RSCH é uma solução inovadora e de grande potencial econômico e social. Permitirá a monitoração de hábitos de vida e a detecção precoce de anormalidades antecipando, assim, ao aparecimento de doenças. De acordo com Lymberis (LYMBERIS & DITTMAR, 2007), a utilização desses sistemas possibilitará a redução de custos hospitalares, pela redução de internações e procedimentos ambulatoriais desnecessários. Também, a redução do erro médico porque a interação entre o paciente e o profissional da saúde estará disponível a qualquer hora e em qualquer lugar que se encontrem.

Atualmente, esses sistemas estão disponíveis comercialmente para captura de algumas informações fisiológicas, como, por exemplo, os marcadores de freqüência cardíaca utilizados por atletas. No futuro, essa tecnologia poderá ser embutida na indumentária por meio de sistemas vestíveis (em inglês, *wearable systems*) e, até mesmo, distribuída dentro do próprio corpo sob forma de microrobôs e nanosensores.

Em geral, as RSCH são projetadas para operar de maneira autônoma, isto é, sem intervenções humanas. Ainda, devem economizar a energia armazenada nas baterias para garantir maior sobrevida ao sistema (aumentar o tempo de funcionamento) e, em conseqüência, às aplicações. Entretanto, uma RSCH é ainda um sistema experimental, com muitas limitações e passível de falhas que devem ser sanadas. Também, em caso de dúvidas acerca de um diagnóstico médico, a avaliação clínica é sempre soberana. Por exemplo, o médico que acompanha o monitoramento de um paciente por meio de uma RSCH pode em algum momento, apesar de contrariar uma política de consumo de energia pré-definida, aumentar o nível do monitoramento do eletrocardiograma (ECG) porque concluiu que o sinal capturado apresentava algumas características que deixavam dúvidas sobre a situação do paciente (BARBOSA *et al*., 2007).

[1] O termo "rede de sensores do corpo humano" é uma tradução do termo original, em inglês, *Body Sensor Networks* (http://vip.doc.ic.ac.uk/bsn/m621.html). Esta tradução foi utilizada pela primeira vez por SENE JR em (SENE JR *et al*, 2005).

O atual estágio do desenvolvimento das RSCH não oferece ainda ferramentas (software) que possibilitem aos médicos e demais profissionais da saúde modificarem o funcionamento desses sistemas. Em contrapartida, é exigido profundo conhecimento técnico acerca da tecnologia quando, por exemplo, é necessária a modificação dos parâmetros de monitoração de um paciente. Neste trabalho argumenta-se que sem a presença de mecanismos que permitam aos médicos e aos demais profissionais da saúde definir e modificar os parâmetros de monitoração, a viabilidade clínica das RSCH pode ser comprometida.

Por outro lado, é indispensável que as RSCH possam tratar falhas, tomar decisões e, até mesmo, atuar sobre o controle de uma aplicação, independentemente de comandos do usuário. Por exemplo, esses sistemas impõem grandes restrições quanto à gerência de recursos, principalmente, da energia armazenada nas baterias. De acordo com Baldus *et al*. (2004), uma RSCH deve operar de maneira autônoma, entretanto, deve estar sempre ao controle de um profissional da saúde.

Atualmente, o software utilizado em RSCH pode ser classificado de duas formas:

i. Por sistemas proprietários construídos em função do hardware e de uma única aplicação. São sistemas pouco modulares, de difícil entendimento e modificação. Em geral, não utilizam multiprogramação[2] porque cada nó sensor oferece apenas uma única funcionalidade para uma aplicação. Assim, os nós sensores são vistos apenas como fontes de dados e a maior parte do processamento é centrado num *gateway* (elemento que interconecta a RSCH a outros sistemas) ou num *Local Processing Unit* (LPU), elemento que retransmite os dados capturados dos sensores. Isso implica em nós sensores de "tamanho único", ou seja, não projetados para mudanças na programação do software pré-instalado. Além disso, toda a gerência da aplicação deve-se concentrar num único ponto (no *gateway* ou no LPU). Isso cria um "gargalo" para o crescimento da RSCH. Exemplos desses sistemas são apresentados em (ASADA *et al*., 2003), (VALDASTRI *et al*., 2004), (LINZ *et al*., 2006), (KARA *et al.,* 2006) e (CHAKRAVORTY, 2006);
ii. Baseado num modelo de programação genérico e de propósito geral. Esse modelo tem como base, por exemplo, a linguagem de programação NesC (GAY *et al*., 2003),

[2] Multiprogramação é a uma abstração desempenhada pelo software que permite a existência de vários programas em execução competindo por um mesmo processador e demais recursos

o sistema operacional TinyOS (HILL, 2003) e um *middleware* [3] para suporte a programação da rede, o Deluge (HUI, 2004). Todos esses sistemas são de uso gratuito e foram desenvolvidos na Universidade da Califórnia, em Berkeley. O CodeBlue (WELSH *et al.*, 2006), o WHMS (JOVANOV *et al*., 2006) e o UbiMon (ICL, 2006) são exemplos de projetos que utilizam esse modelo. Outros exemplos são apresentados em (BALDUS *et al.*, 2004) e (FARSHCHI *et al*., 2006).

Entretanto, aplicações desenvolvidas com base no TinyOS apresentam as seguintes limitações para emprego em RSCH:

- A linguagem de programação NesC impõe sintaxe peculiar, baseada em conceitos emergentes da engenharia de software. Sem o conhecimento da lógica de programação e a *expertise* para manipular componentes de software é praticamente impossível modificar ou desenvolver novas aplicações. Como conseqüência disso, alterar parâmetros relativos ao funcionamento de uma aplicação NesC é uma atividade que deve ser executada apenas por programadores experientes;
- O modelo de multiprogramação oferecido pelo TinyOS é pouco interativo. Oferece muito pouco controle sobre as possíveis funcionalidades (tarefas) executadas pelos nós sensores, porque não há troca de contexto [4]. Tarefas não podem ser imediatamente interrompidas ou substituídas para atender a uma política definida por uma instância superior da aplicação ou, simplesmente, para atender um comando de usuário. Ainda, segundo Han *et al*. (2005), o modelo implementado pelo TinyOS dificulta a existência de tarefas que consumam muito tempo para execução. Isso aumenta a possibilidade de ocorrência de bloqueios, e esses, resultam na paralisação total ou parcial dos sensores.

Assim, um grande desafio se apresenta aos projetistas de RSCH. Aumentar com transparência o acesso às configurações internas dos nós sensores e da rede, sem excluir a possibilidade de operação autônoma desses sistemas. Transparência refere-se a uma

[3] *Middleware*, nesse contexto, refere-se a um conjunto de bibliotecas de código que fornece uma abstração ao programador de aplicações. Essa abstração tem como objetivo aumentar a transparência disponibilizada ao programador. Para isso, é criada uma camada de software intermediária entre o sistema operacional e os programas que representam a aplicação.

[4] A troca de contexto prevê o armazenamento do estado de execução de uma tarefa (informações relativas a tarefa) para que a execução da mesma possa ser reiniciada a partir da última linha de código executada anteriormente. A multitarefa é um modelo de multiprogramação que possibilita a troca de contexto.

propriedade de sistema. Um sistema é mais transparente quando ao programador e/ou ao usuário são escondidos detalhes de implementação e do funcionamento desse sistema.

1.2 – OBJETIVOS DO TRABALHO

O objetivo principal deste trabalho é propor uma arquitetura de RSCH. Essa arquitetura comporta sistemas implementados por software e hardware. Esses sistemas oferecem suporte a um modelo de programação e configuração de RSCH, que é a principal inovação deste trabalho.

A arquitetura de RSCH proposta neste trabalho foi validada pela implementação de protótipos (hardware e software) para provas de conceito. Também, na execução de testes sobre esses protótipos.

1.3 – SOLUÇÃO PROPOSTA E METODOLOGIA

Nos últimos anos tem sido desenvolvido no Laboratório de Tecnologia Biomédica do Hospital Universitário da Universidade de Brasília um projeto denominado de *Body-worn Sensor Networks* (BWSNET). Esse projeto prevê o desenvolvimento de um sistema de monitoração da saúde humana com o intuito de monitorar a saúde e estados de doença. Para tanto, divide a monitoração em três segmentos: *(i)* monitoração de sinais fisiológicos, *(ii)* monitoração de sintomas e *(iii)* captura de informações com o uso de bibliotecas digitais baseadas em evidências. É importante frisar que o sistema é interligado ao Prontuário Eletrônico do Paciente, ao seu médico assistente, a serviços de atendimento de urgência e ao serviço de saúde de referência.

O projeto BWSNET propõe a utilização da plataforma *wearable* baseada em um sistema vestível e, também, a utilização de uma possível rede de nanosensores distribuídas em diversas partes do corpo. O uso das duas plataformas de hardware visa integrar completamente a monitoração da saúde do indivíduo. A interface de hardware utilizada para comunicação com o indivíduo, com seus familiares e com os profissionais de saúde é um computador de mão, o PDA (*Personal Digital Assistant*), e/ou um aparelho celular. A Figura 1.1 mostra uma tela do módulo de monitoração de sintomas. Por meio desta tela, pode-se realizar a revisão de sistemas, uma parte da anamnese.

Num exemplo de aplicação desse sistema, se o módulo de monitoração contínua do ECG (eletrocardiograma) detectar a presença de uma arritmia cardíaca associada à queda da pressão arterial e o módulo de monitoração de sintomas capturar que o paciente está apresentando tontura e turvação visual (por exemplo, por meio de comandos de voz emitidos pelo próprio paciente), o sistema pode integrar essas informações e chegar à conclusão que o indivíduo está apresentando uma arritmia grave e que o estado do mesmo é grave. Em virtude desta conclusão, o próprio sistema pode fazer uma comunicação automática ao serviço de ambulância de urgência, médico assistente, ou unidade de saúde. Estes devem ter sido previamente cadastrados como as referências do usuário do sistema.

Figura 1.1 – Coleta automatizada de sintomas em um aparelho celular (JOSÉ *et al.*, 2007)

Como uma das estruturas do projeto *Body-Worn Sensor Networks*, uma arquitetura de software de sistema chamada de SOAB (*Software Architecture for Body-Worn Sensor Networks Project*) foi desenvolvida. Uma arquitetura de software de sistema é uma estrutura dos componentes de um programa/sistema, seus inter-relacionamentos, princípios e diretrizes guiando o projeto e a evolução ao longo do tempo (MENDES, 2002, p. 5).

A SOAB é constituída por quatro camadas: *(i)* aplicação, *(ii)* interconexão, *(iii)* serviços da RSCH e *(iv)* serviços do hardware. Cada camada é responsável por oferecer um conjunto de serviços (funcionalidades), considerando requisitos específicos das RSCH e de seus usuários. Em termos de implementação, as camadas da SOAB correspondem respectivamente a: *(i)* uma ferramenta para programação e configuração de RSCH, *(ii)* um *middleware* baseado no conceito de serviços e tecnologias *Web*, *(iii)* um servidor de aplicações, chamado *BWSNET Proxy* e *(iv)* um sistema operacional para nós sensores

utilizados em RSCH, chamado MedOS. A Figura 1.2 ilustra as camadas da arquitetura SOAB e como estas interagem com seus principais usuários.

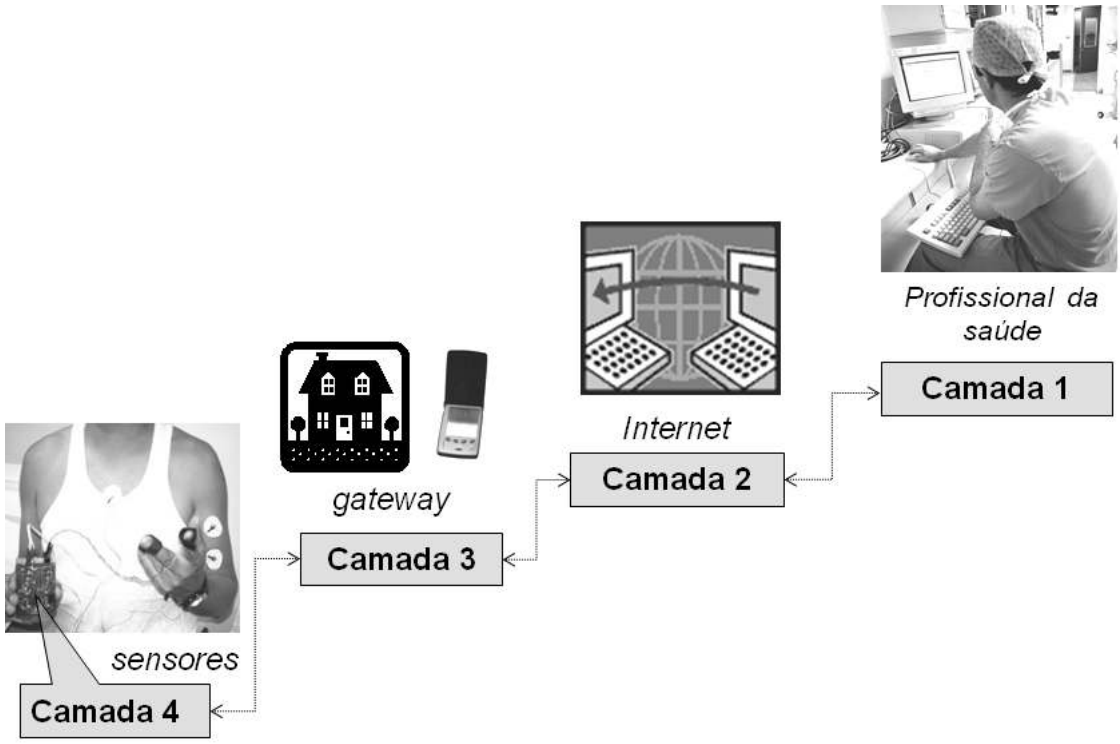


Figura 1.2 – Camadas da Arquitetura SOAB

Neste trabalho foram considerados inicialmente dois perfis de usuário: o paciente e o profissional da saúde. O paciente é quem interage diretamente com sensores, assistentes pessoais e demais dispositivos de monitoramento. Por meio dos dispositivos são efetuadas as coletas dos sinais eletrofisiológicos, dos sintomas e de outras informações de saúde do próprio paciente. Entretanto, o paciente é um agente passivo no que se refere à programação e a configuração dos sistemas de monitoramento, isto é, a ele não é atribuída nenhuma ação referente a essa atividade. Os médicos e demais profissionais da saúde representam o segundo perfil de usuário do sistema. Recebem e manipulam as informações obtidas pelo monitoramento e, principalmente, são os responsáveis por definir e modificar parâmetros referentes ao monitoramento de seus pacientes. Em última instância, são os responsáveis pela decisão, mesmo que esta tenha sido tomada pelo próprio sistema (sistema autônomo). No transcorrer do texto, o termo “usuário” será utilizado para designar o médico ou, de forma genérica, os profissionais da saúde.

Uma arquitetura de software em camadas possibilita que a transparência seja aumentada de forma incremental, promovendo a portabilidade, isto é, facilitando modificações, até mesmo, do próprio hardware. A seguir, são apresentados maiores detalhes acerca dos sistemas que implementam a arquitetura SOAB.

BWSNET Configuration Tool é uma ferramenta para programação e configuração de redes de sensores do corpo humano. Representa uma implementação da camada 1 da SOAB. Essa ferramenta oferece aos usuários um meio pelo qual podem definir a programação e efetuar a configuração de uma RSCH. Em síntese, possibilita modificar os parâmetros relativos ao monitoramento de pacientes, alterando a programação e a configuração dos sensores.

A programação de uma rede de sensores pode ocorrer antes que a rede entre em operação ou durante a execução de alguma aplicação. Para descrever o primeiro caso, usa-se o termo programação estática, programação em tempo de compilação ou, em inglês, *deployment-time programmability*. Para o segundo caso, usa-se o termo reconfiguração dinâmica, configuração em tempo de execução ou, em inglês, *runtime set-up*. A Figura 1.3 mostra as atividades relativas ao modelo de programação de redes de sensores do corpo humano proposto neste trabalho. Esse modelo é composto por dois ciclos distintos e independentes de atividades: a Programação Estática (referida ao longo deste texto por "programação") e a Reconfiguração Dinâmica (referida ao longo deste texto por "configuração"). Algumas das atividades do modelo são executadas pelos usuários e o restante é automatizado pelo próprio sistema.

A programação implementa o processo responsável pela definição dos mecanismos e políticas, sob forma de artefatos de software (programas de computador), que devem ser instalados nos nós sensores para suporte a uma determinada aplicação. A configuração é responsável pela definição e/ou ajustes das aplicações com base nas fontes de dados selecionadas (sensores) e no desempenho requerido do sistema (Qualidade de Serviço). Pode ser executada manualmente, no âmbito dos nós sensores, ou automatizada pelo sistema após o recebimento dos parâmetros enviados pelo usuário.

BWSNET Configuration Tool disponibiliza uma interface gráfica (visual). Com isso, o processo de programação e configuração de uma RSCH torna-se menos cansativo, menos

propício a erros, mais intuitivo e mais fácil. Principalmente, por não exigir conhecimento específico de modelos computacionais e de lógica de programação.

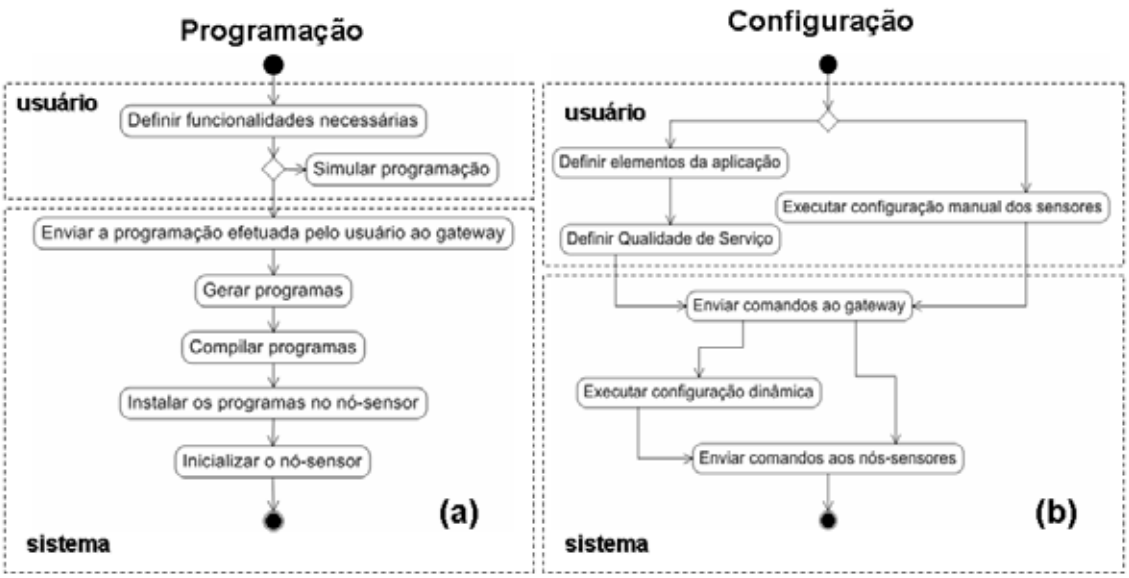


Figura 1.3 – Diagramas de atividades relativo ao modelo para programação das redes de sensores para monitoramento do corpo humano

Inicialmente, foram desenvolvidos dois tipos de sensores para serem programados por meio de *BWSNET Configuration Tool*: o nó sensor de ECG e o nó sensor genérico. O nó sensor de ECG é programado para o monitoramento do eletrocardiograma, enquanto que, o nó sensor genérico pode ser programado para monitorar vários tipos de sinais, como, por exemplo, a pressão arterial, o eletromiograma não-invasivo, a oximetria de pulso e outros. Entretanto, pode haver situações em que o profissional da saúde deseje utilizar um novo sensor, como, por exemplo, um sensor recém lançado no mercado. Neste caso, pressupõe-se que o novo sensor será comercializado com o software específico para a programação. Pensando nisso, foi desenvolvido um mecanismo que possibilita a inclusão e a remoção do novo sensor. Esse mecanismo habilita a *BWSNET Configuration Tool* a carregar em memória o novo programa (relativo ao novo sensor), a fazer a inspeção e, então, a executá-lo a partir de uma invocação de elementos do próprio programa, por exemplo, métodos, descobertos durante a execução.

BWSNET Configuration Tool propõe que, para cada nó sensor, seja disponibilizado um simulador. Isso possibilita que o usuário possa avaliar o desempenho da programação antes

que ela seja de fato iniciada, minimizando a possibilidade de erros. Além disso, o simulador pode ser utilizado no treinamento de novos usuários para programação de RSCH.

Se desconsiderarmos a necessidade de reprogramação por falhas do próprio sistema, então, o ambiente onde o paciente está inserido e, principalmente, o estado de saúde do paciente podem requerer que os sensores sejam reprogramados com muita freqüência. Em muitos casos, essa necessidade pode ser suprida por um reajuste das configurações do software já instalado. Como resultado, obtém-se maior rapidez e menor risco de falhas porque não é necessário que o hardware seja reinicializado.

Para isso, e como parte de *BWSNET Configuration Tool*, foi desenvolvida a ferramenta para configuração (reconfiguração dinâmica) da rede e dos nós sensores. Essa ferramenta possibilita que as atividades mostradas na Figura 1.3b possam ser desempenhadas. Assim, os usuários poderão configurar suas aplicações, selecionando os sensores com as respectivas funcionalidades (definir elementos da rede), associando a eles os respectivos níveis de desempenho requeridos a cada instante (definir Qualidade de Serviço). Também, é permitida aos usuários a manipulação de valores de prioridade e estados de execução relacionados às tarefas executadas pelos nós sensores (executar configuração manual dos sensores). As tarefas são mapeadas em funcionalidades para facilitar a compreensão dos usuários, enquanto que, os valores de prioridade são associados a possíveis estados de saúde do paciente a cada instante, como, por exemplo, alto, médio e baixo risco.

O objetivo da camada 2 da SOAB é possibilitar que os usuários possam desempenhar suas atividades pela Internet. Para isso, além das interconexões físicas da rede é necessário um meio lógico (abstração em software). Este deve facilitar modificações, quando necessárias e, ainda, de forma independente das demais camadas da SOAB. Para isso, uma especificação de *middleware* foi proposta e um conjunto de chamadas remotas a procedimentos foi implementado de acordo com as recomendações do W3C Web Services (W3C, 2006). A interconexão da rede de sensores com outras redes por meio de *web services*[5] possibilita que a RSCH seja vista como uma entidade que fornece serviços para usuários com necessidades diferentes e dinâmicas. Ainda, possibilita que usuários possam

[5] *Web services* podem ser definidos como programas modulares, independentes e autodescritivos que podem ser descobertos e invocados através da Internet ou de uma Intranet corporativa.

acessar serviços independentemente do uso da ferramenta para programação e configuração (camada 1 da SOAB).

Neste momento, surgem dois novos possíveis perfis de usuário do sistema: *(i)* programadores especializados e *(ii)* outros sistemas de informação, por exemplo, um sistema de informação de saúde (*Heathcare Information System – HIS*). Ambos, que no transcorrer do texto serão chamados de "terceiros", de fato, não necessitam da primeira camada da SOAB para acessar serviços providos pela RSCH. Precisam descobrir quais serviços estão sendo disponibilizados e como acessá-los por meio das interfaces de programação.

A terceira camada da SOAB é implementada pelo *BWSNET Proxy*. Este software (um servidor) foi projetado e desenvolvido para operar nos dispositivos de *gateway*. Os dispositivos de *gateway* podem ser computadores pessoais, *laptops*, PDAs ou, até mesmo, um telefone celular. Para isso, devem possuir interface de rede física compatível com a tecnologia usada na RSCH.

O *BWSNET Proxy* é responsável por traduzir as requisições do formato utilizado pelo *middleware* (camada 2 da SOAB) para comandos enviados aos sensores. Na prática, isso implica gerar os programas que serão compilados e, posteriormente, instalados e executados pelos nós sensores. Também, executar o ajuste das configurações por meio comandos (mensagens) enviados aos sensores.

A proposta de geração automática de programas que devem ser executados pelos sensores (programação) e do ajuste automático e/ou manual das configurações do sistema (configuração) foram motivadas pelo trabalho desenvolvido por Carvalho (CARVALHO, 2005). Neste trabalho, o autor argumenta ser possível economizar energia dos sensores alternando o monitoramento do paciente entre diferentes níveis. Embora o ideal seja monitorar um paciente utilizando o máximo dos recursos, nem sempre isso é estritamente necessário. De acordo com o estado de saúde do paciente a cada instante de tempo, é possível estabelecer um nível para o monitoramento do mesmo. Dessa forma, é possível economizar energia e, assim, aumentar o tempo de funcionamento do hardware de monitoramento. Neste trabalho, essa hipótese é discutida no âmbito da programação e/ou reajuste de parâmetros que comandam o funcionamento do hardware como, por exemplo,

as interfaces de comunicação, a freqüência de operação e o desligamento de partes dos nós sensores quando não estiverem em uso. Essas funcionalidades são comandadas e gerenciadas por módulos do *BWSNET Proxy.*

Um desses módulos é um compilador inteligente para RSCH (BARBOSA *et. al.*, 2007). Esse software (chamado gerador de agente) é capaz de definir mecanismos e políticas que farão parte do sistema (software) que será instalado nos nós sensores. Na prática, os requisitos de uma aplicação são definidos pelos usuários. Em seguida, são repassados ao gerador de agente que utiliza essas informações para decidir quais serão os mecanismos necessários e as melhores políticas para, por exemplo, economizar energia. Como resultado, é gerado o programa que será compilado e instalado nos nós sensores. O gerador de agente possibilita manter a transparência oferecida aos usuários e ao mesmo tempo aumentar a eficiência do sistema porque possibilita utilizar o conhecimento especialista para combinar os requisitos de cada aplicação (definidos pelo usuário) com as funcionalidades (recursos) disponibilizadas pelo sistema.

A quarta e última camada é representada por um sistema operacional para uso em sensores de uma RSCH. O MedOS é composto por: *(i)* um conjunto de instruções (comandos) para configuração dos nós sensores denominado protocolo MedOS; *(ii)* um interpretador de comandos para o protocolo MedOS e *(iii)* um conjunto de bibliotecas de código para gerenciamento do hardware (*device drivers*). Atualmente, os *drivers* disponibilizados pelo MedOS controlam a conversão A/D (Analógico/Digital) e alguns filtros digitais, permitem a programação da interface de comunicação, a seleção dos modos de operação e o ajuste da freqüência de operação da CPU para economia de energia. Na prática, o MedOS é constituído por um conjunto de programas (escritos em linguagem C) para ser utilizado em nós sensores projetados para o monitoramento da saúde humana e desenvolvidos no projeto BWSNET.

O principal módulo do MedOS é o interpretador de comandos. Ele oferece meios que permitem a interação dos usuários e/ou do próprio sistema (camadas superiores da SOAB) com os sensores durante a execução das tarefas. Essas interações são expressas pelo envio de comandos para modificações do funcionamento dos sensores. Portanto, viabilizam o processo de configuração do sistema, no âmbito dos nós sensores. Em termos de implementação, o interpretador de comandos corresponde a uma tarefa de maior prioridade

que pode ou não ser incluída durante a programação do software que será instalado nos sensores. Caso esse módulo não seja incluído, o sistema fica desabilitado para efetuar configurações no âmbito dos sensores. Na prática, todos os comandos para uma configuração são repassados pelo BWSNET *Proxy* ao interpretador de comandos que, deve reconhecer a validade desses comandos dentro do conjunto de comandos do MedOS (protocolo MedOS) e, então, desencadear uma ação. As ações estão relacionadas ao ajuste de parâmetros que comandam as tarefas durante a execução das mesmas.

A metodologia utilizada neste trabalho partiu da modelagem do domínio, seguida pela modelagem da arquitetura e a implementação dos sistemas. Acredita-se que esse ciclo de atividades contribua para o entendimento do problema e da solução proposta. Em síntese, a Figura 1.4 mostra a metodologia utilizada para o desenvolvimento e apresentação deste trabalho.

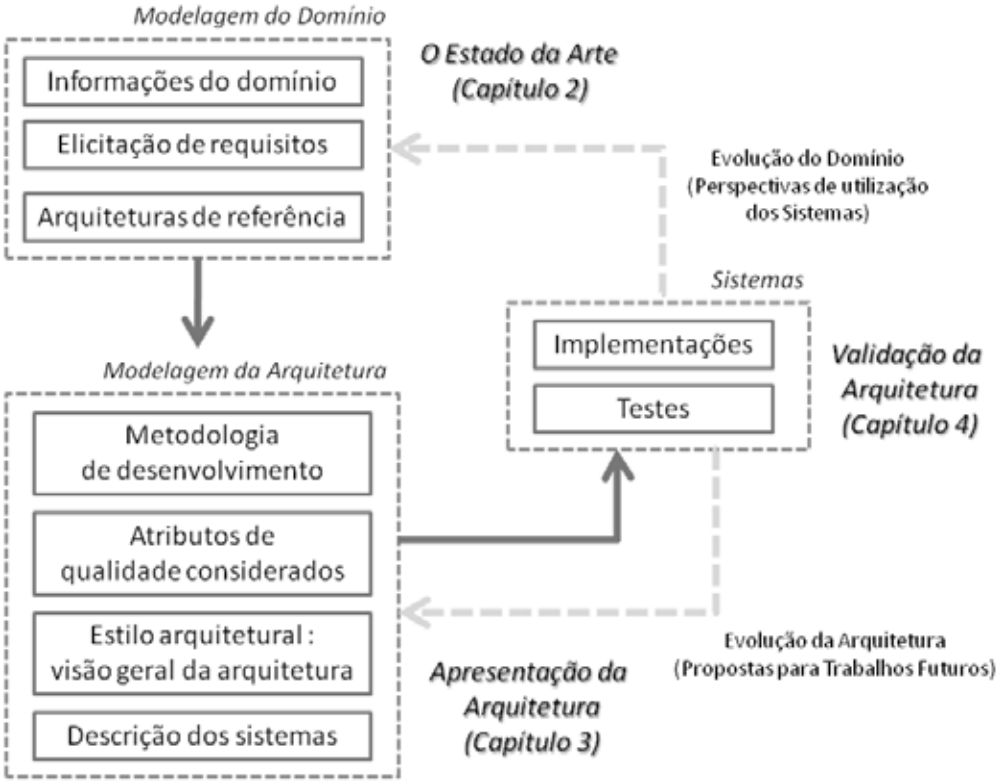


Figura 1.4 – Metodologia utilizada para o desenvolvimento do trabalho.

1.4 - CONTRIBUIÇÕES

Até o momento, a programação e a configuração das redes de sensores é uma atribuição exclusiva de profissionais da Ciência da Computação altamente especializados. A linha de pensamento predominante pressupõe que no futuro esses dispositivos devam ser suficientemente autônomos e inteligentes, e, portanto, não necessitem de intervenções humanas. Neste trabalho, este pressuposto é questionado porque existem pontos que ultrapassam o atual estado de desenvolvimento da tecnologia. Um exemplo refere-se à responsabilidade legal imputada ao médico durante o acompanhamento de um paciente. Em última instância, os médicos são os responsáveis pela decisão final, mesmo que esta tenha sido tomada pelo próprio sistema.

Neste trabalho argumenta-se que o fato de um sistema operar com diferentes níveis de autonomia não exclui a necessidade de mecanismos para intervenções humanas. Argumenta-se, ainda, que os usuários envolvidos diretamente com as aplicações podem efetuar essas intervenções de maneira a melhorar a eficiência do sistema sem comprometer a sua eficácia. Dessa forma, entende-se que o conceito de operação autônoma e a existência de mecanismos para intervenções humanas não são excludentes e sim complementares. Portanto, podem e precisam coexistir. Diante disso, um grande desafio encontrado foi responder à questão de como atender a essas necessidades.

O aspecto original deste trabalho envolve uma proposta de mudança do paradigma utilizado atualmente para programação e configuração das redes de sensores. Propõe que os próprios usuários dessa tecnologia possam definir as funcionalidades necessárias a cada momento e, ainda, intervir no sistema quando julgar pertinente. Isto, sem excluir a possibilidade desses sistemas se tornarem cada vez mais autônomos e eficientes. Assim, um modelo de programação e configuração de RSCH foi proposto. Esse modelo é suportado por um conjunto de abstrações agrupado em quatro níveis hierárquicos e organizados sob forma de uma arquitetura de software de sistema.

Quanto às inovações tecnológicas destacam-se as apresentadas a seguir:

- Uma ferramenta gráfica (visual) para programação e configuração de redes de sensores do corpo humano;
- Um simulador para avaliar o resultado da programação dos sensores de uma RSCH;

- Um compilador inteligente e de nível intermediário. Esse sistema é o núcleo do modelo de programação apresentado neste trabalho. Por meio dele é possível aumentar a eficiência de uma RSCH mantendo a transparência necessária aos usuários. Avalia-se, portanto, que esse conceito tenha ainda grande potencial para novas aplicações e aprimoramento durante os próximos anos.
- A aplicação da multitarefa em nós sensores de uma RSCH. A multitarefa permite que os nós sensores sejam dotados de funcionalidades mais complexas e ao mesmo tempo mais interativas. Também, possibilita a criação de novas abstrações para uso no software embutido nos sensores. Isso pode resultar no aumento na eficiência do sistema. Um exemplo são os algoritmos para escalonamento de tarefas. Concebidos com base no conhecimento especialista, esses algoritmos devem considerar os recursos disponibilizados pelos sensores combinados com os requisitos de cada aplicação. Neste trabalho esses algoritmos foram implementados por artefatos chamados de agentes e, ainda, um conjunto deles, de família de agentes.

1.5 – VISÃO GERAL DO TEXTO

Neste capítulo já foram apresentadas as motivações, os objetivos e as principais contribuições referentes a este trabalho. Também, foram mostradas as evidências e as justificativas que nortearam a escolha do tema tratado, e apresentadas as principais soluções adotadas que, em parte, resultaram em inovações. O capítulo serve também como resumo executivo do trabalho.

No Capítulo 2 são abordados conceitos, definições, desafios e tendências relativas às redes de sensores sem fios, de maneira geral, e às em redes de sensores do corpo humano, de maneira específica. Essas constatações são frutos da investigação do autor ao longo dos últimos quatro anos e têm como objetivo reportar ao leitor o estado atual da arte e a complexidade do tema tratado nesta tese de doutorado.

O Capítulo 3 apresenta os requisitos e as principais especificações de projeto da arquitetura proposta. Essas especificações são relativas ao software e ao hardware que compõem os sistemas que implementam uma arquitetura de redes de sensores do corpo humano. Também, e como parte da modelagem da arquitetura, são destacados a metodologia de desenvolvimento e os atributos de qualidade considerados.

O Capítulo 4 tem como objetivo apresentar a validação da arquitetura proposta. Nesse capítulo são apresentados os protótipos, os testes executados com os protótipos e os resultados obtidos com esses testes. Para isso, são descritas as tecnologias utilizadas e os detalhes de implementação dos projetos apresentados no Capítulo 3.

O Capítulo 5 é a Conclusão. Nesse capítulo são apresentadas considerações sobre os resultados alcançados em relação ao projeto, implementação e validação do modelo proposto. Também, são apresentas propostas de aprimoramento para execução em trabalhos futuros. Uma delas descreve a metodologia e apresenta os protótipos desenvolvidos para a aplicação de testes de usabilidade. Esses testes podem gerar resultados qualitativos, do ponto de vista dos usuários, acerca do modelo de programação proposto neste trabalho.

2 – REQUISITOS, DESAFIOS E PERSPECTIVAS DO USO DE REDES DE SENSORES NO MONITORAMENTO HUMANO

O objetivo deste capítulo é apresentar o estado da arte acerca das redes de sensores utilizadas no monitoramento da saúde humana. Para isso, são apresentados conceitos, requisitos e tendências que estão em voga no momento. Essas questões são importantes para o entendimento do restante do trabalho porque foram absorvidas pelo projeto e implementação da arquitetura proposta nesta tese de doutorado.

2.1 – CENÁRIO ATUAL

Em relação ao monitoramento da saúde do ser humano, três paradigmas têm influenciado a estruturação da prestação de serviços em saúde: *(i)* os sistemas de informação centrados no paciente, *(ii)* a descentralização da prestação dos serviços em saúde em direção ao domicílio e *(iii)* a prioridade dada à melhoria da qualidade de vida dos pacientes. Antevê-se que o próximo paradigma a ser efetivado é a disponibilização de soluções direcionadas aos indivíduos, na forma de monitoração de hábitos de vida e detecção precoce de anormalidades com a finalidade de realização da prevenção primária, isto é, evitar o aparecimento de doenças (BARBOSA *et al*., 2004).

As Redes de Sensores do Corpo Humano (RSCH) se apresentam como ferramentas indispensáveis à implantação desse novo paradigma. Os sensores (nós) podem ser fixados em determinados pontos do corpo humano ou podem ser móveis. Quando móveis, eles podem dispor de um mecanismo próprio (mobilidade ativa), ou aproveitar um veículo disponível (mobilidade passiva). Também, têm capacidade de medir e/ou inferir sobre algum fenômeno externo como, por exemplo, o reconhecimento do contexto de onde se encontram. Ainda, processar e transmitir os dados para outros sensores (nós).

Os sensores para monitoramento da saúde humana são também classificados como: *(i)* não-invasivos, por exemplo, os sensores que monitoram sinais eletrofisiológicos capturados na superfície da pele, como, o eletrocardiograma (ECG); *(ii)* semi-invasivos, que utilizam das cavidades externas para o monitoramento interno do corpo-humano. Por exemplo, uma cápsula endoscópica utiliza o trato gastrintestinal para o monitoramento do sistema

digestivo por meio de imagens capturadas. A Figura 2.1 ilustra uma cápsula endoscópica. E, *(iii)* invasivos - sensores implantados dentro do corpo humano, por exemplo, para monitoramento da pressão intra-ocular. Ou ainda, uma rede minúscula de sensores podem ser utilizados para construir uma retina artificial (SCHWIEBERT *et al.*, 2001). Os sensores invasivos são implantados em pacientes por meio de um procedimento cirúrgico.

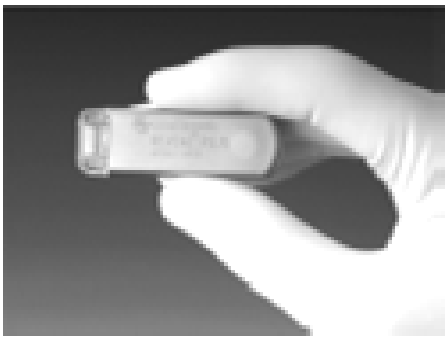


Figura 2.1 – Cápsula endoscópica (YANG, 2006, p. 13).

Para Yang (2006, p. 5) as RSCH têm requisitos específicos em relação às redes de sensores de propósito geral. Em síntese, podem ser destacados:

- Altos níveis de segurança para garantir a confidencialidade das informações dos pacientes que estão sendo transmitidas pela rede sem fios;
- Biocompatibilidade e biodegradabilidade que, em geral, aumentam os custos;
- As aplicações de RSCH são sensíveis à perda de dados. Portanto, precisam de mecanismos para checagem de falhas e que assegurem a Qualidade de Serviço (QoS) mínima aceitável. O conceito de QoS tem várias nuanças. Uma delas é apresentada na Seção 2.2.4.2;
- Reconhecimento do contexto. Em geral, as variações fisiológicas estão muito relacionadas às variações do contexto (ambiente e/ou situação) onde estão inseridos os pacientes. Portanto, a capacidade de reconhecimento do contexto (em inglês, *context awareness*) é muito importante;
- Em geral, utiliza-se pequeno número de nós sensores, entretanto, mais precisos. Cada nó sensor pode executar múltiplas tarefas;
- As aplicações são mais sensíveis aos ruídos causados pelo movimento do corpo humano do que por variações climáticas, ruído do ambiente e perda de sincronia.

Há também aplicações que requerem o funcionamento da RSCH com ubiqüidade e a qualquer instante de tempo (em inglês, *pervasive monitoring*). Seja para obtenção do diagnóstico de algumas enfermidades, como, por exemplo, o diagnóstico de fibrilação atrial[6] ou para o monitoramento de um paciente de alto risco numa unidade de terapia intensiva. Nesse último caso, uma falha na detecção de eventos (anormalidades) pode causar um prejuízo irreversível, isto é, a morte do paciente.

Atualmente, duas linhas de pesquisa têm emergido como fortes tendências acerca da utilização de RSCH: as redes de sensores vestíveis e as redes de nanosensores.

A utilização do conceito de Sistemas Vestíveis (em inglês, *Wearable Systems*) tem se apresentado como solução para monitoramento humano direcionado ao indivíduo. Possibilita o acompanhamento contínuo, não-invasivo, minimamente obstrutivo e, até mesmo, em tempo real de pacientes à distância. De acordo com Bonato (2003), os sistemas vestíveis são dispositivos desobstrutivos que possibilitam aos avaliadores monitorarem as pessoas em atividades cotidianas, durante longos períodos de tempo.

Na prática, um sistema computacional vestível se apresenta como um nó sensor ou como uma rede de sensores embutidas numa roupa. Esses sensores podem capturar dados e/ou, em alguns, casos gerar informações com base nos dados capturados. A Figura 2.2 apresenta um exemplo de sistema vestível. O MIThril (SUNG & PENTLAND, 2004) é um computador vestível capaz de capturar o eletrocardiograma (ECG), o eletromiograma (EMG) não-invasivo, a resistência galvânica da pele e a temperatura cutânea. Ainda, é capaz de determinar o movimento (associado às atividades físicas) de seus usuários.

De maneira geral, as redes de sensores sem fios viabilizam a confecção de sistemas vestíveis empregados para o monitoramento da saúde humana, pela agregação das seguintes características:

- Eliminação de fatores obstrutivos, tais como fios, alimentação externa, tamanho e o próprio peso dos componentes. Isso facilita a integração das tecnologias sob o ponto de vista da ergonomia;

[6] A fibrilação atrial é a arritmia cardíaca que pode levar à alteração da qualidade de vida e, também, à morte. Está presente, principalmente, em pessoas com mais de 60 anos. Normalmente, para o seu diagnóstico é exigido um exame conhecido como holter. O holter é a captura e armazenamento (por meio de um dispositivo móvel) do ECG durante o período de 24h.

- Aptidão dos sensores e sistemas para a produção de resultados relevantes e precisos. A capacidade de processamento inerente a uma rede de sensores possibilita inclusive a obtenção de informações não captadas durante o sensoriamento, mas, por meio de síntese e extração de características. Tanto nos arredores do sensoriamento, quanto na estruturação e no armazenamento dos dados. A essa capacidade dá-se o nome de Fusão de Dados;
- Baixo custo. Geralmente, associado ao uso de componentes de prateleira (em inglês, *off-the-shelf*). Isso pode levar no futuro a um impacto social de proporções consideráveis.

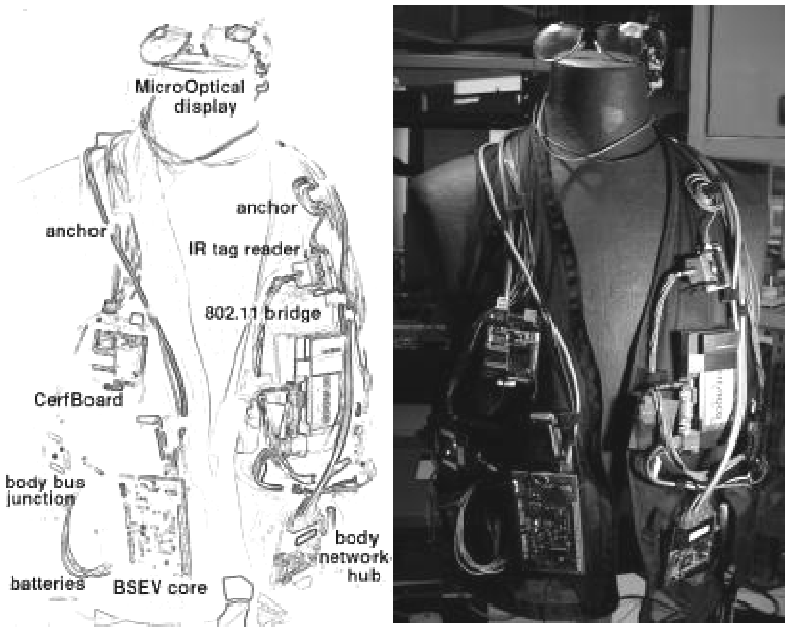


Figura 2.2 – MIThril (SUNG & PENTLAND, 2004).

Embora muitos avanços científicos tenham sido obtidos, restam ainda grandes desafios para a operacionalização de sistemas vestíveis no monitoramento humano. A necessidade de subsistemas que garantam segurança, manutenção mínima e, principalmente, facilitada (por meio de ferramentas específicas) e a capacidade de integração a outros sistemas de informação são indispensáveis para o sucesso desse paradigma.

Em 13 de abril do ano 2000, a NASA (*National Aeronautics and Space Administration*) e o *National Cancer Institut*e (NCI) assinaram um memorando cujo objetivo é desenvolver novas tecnologias biomédicas capazes de possibilitar que em 2020 o homem possa se

lançar numa viagem de ida-e-volta com sucesso a Marte (veja http://nasa-nci.arc.nasa.gov/index.cfm). Para isso, a NASA procura por uma nova forma de medicina, onde "exploradores microscópios" irão percorrer o corpo humano a procura de doenças, como o câncer. Esses nanorobôs de dimensões muito pequenas, poucos nanômetros, equipados com nanosensores devem compor uma rede inteligente e autônoma. Um nanômetro equivale a $1{,}0\times10^{-9}$ metros, ou seja, um milionésimo de milímetro. A Figura 2.3 ilustra (por uma imagem gerada por computador) uma potencial aplicação para um nanorobô. Nesta imagem o nanorobô é utilizado para injetar uma droga numa célula do corpo humano infectada por uma doença.

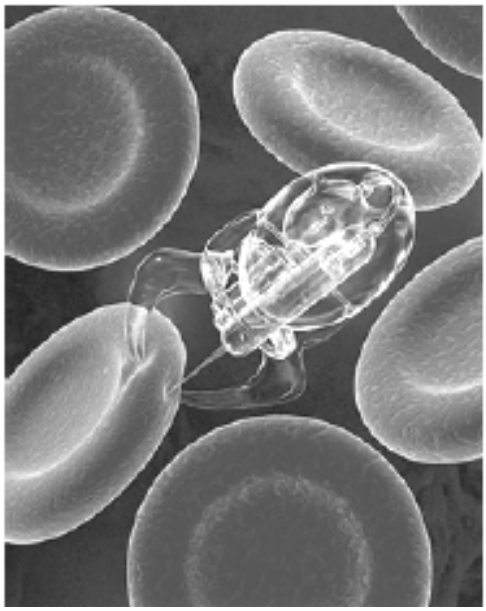


Figura 2.3 – Um nanorobô em operação (YANG, 2006, p. 28).

Tanto para os sistemas vestíveis quanto para os nanosensores, um dos maiores limitantes continua sendo o tempo de funcionamento desses dispositivos, que é determinado principalmente pelo consumo da energia armazenada pelas baterias. Isso se deve principalmente ao fato da evolução da capacidade de armazenamento de energia das baterias (em inglês, *battery energy density*) não ter acompanhado a evolução dos demais dispositivos computacionais. Como exemplo, a Figura 2.4 apresenta um gráfico comparativo (que representa a taxa de crescimento) acerca da evolução das baterias em relação aos demais dispositivos computacionais de 1990 até 2003.

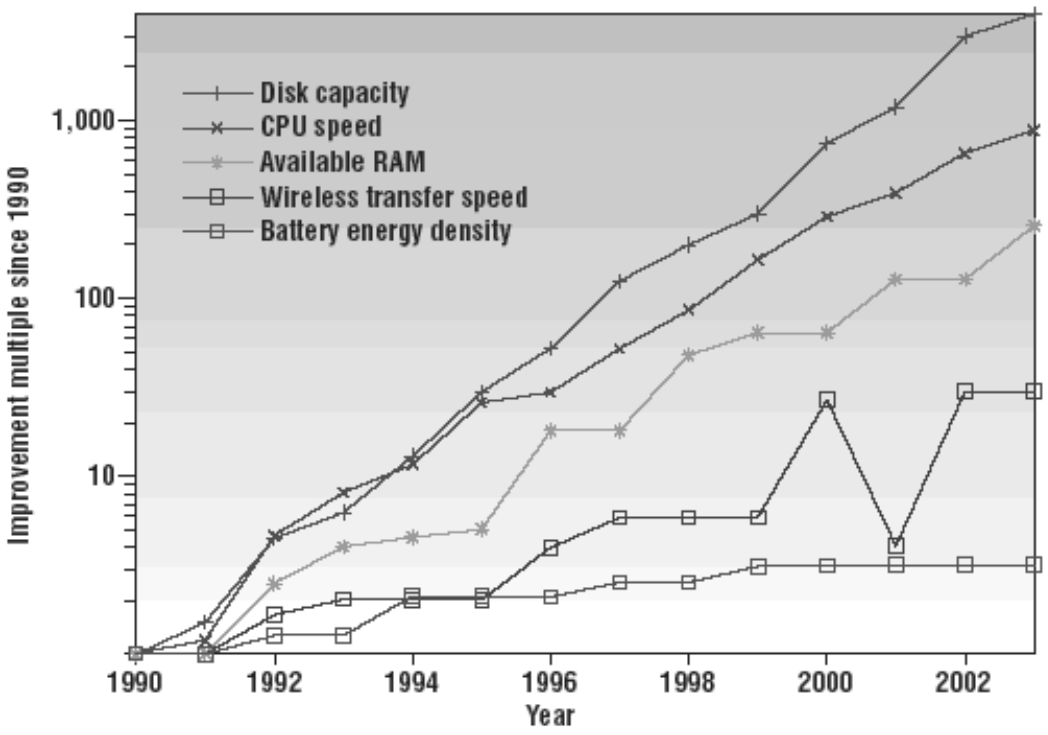


Figura 2.4 – Capacidade de armazenamento das baterias comparada com a evolução dos demais dispositivos computacionais (modificado - PARADISO & STARNER, 2005).

Para lidar com a limitação imposta pelas baterias, atualmente, são duas as principais abordagens: a adequação do software e do hardware para uso racional da energia elétrica e a extração da energia necessária do meio no qual o nó sensor é utilizado.

Atualmente, a primeira abordagem é a mais empregada e será apresentada em detalhes nas Seções 2.2 e 2.3. Já, a segunda abordagem caminha a passos mais lentos, porém, não tão pequenos. Para alguns tipos de sistemas vestíveis a energia pode ser obtida pelo uso de transdutores que convertem energia mecânica, por exemplo, obtida dos movimentos do corpo humano em energia elétrica, necessária para alimentação dos circuitos eletrônicos. Um exemplo disso é apresentado na Figura 2.5. O *shoe generators* é um protótipo de nó sensor vestível (em forma de calçado) desenvolvido pelo MIT Media Laboratory. É capaz de gerar até 60 mW (miliwatts) de potência para alimentação dos circuitos eletrônicos.

Caminhando na direção da nanotecnologia alguns avanços significativos já foram conseguidos. Por exemplo, dos trabalhos desenvolvidos por Montemagno (MONTEMAGNO, 2001) e Soong (SOONG *et al.*, 2001) obteve-se um nanomotor movido por Trifosfato de Adenosina (ATP) – substância encontrada no citoplasma e no

nucleoplasma de todas as células do corpo – capaz de desempenhar 115 rotações por segundo (rps) e 100.000 (cem mil) vezes menor que um grão de areia. Todavia, a aplicabilidade dessa abordagem esbarra ainda em questões tecnológicas e éticas.

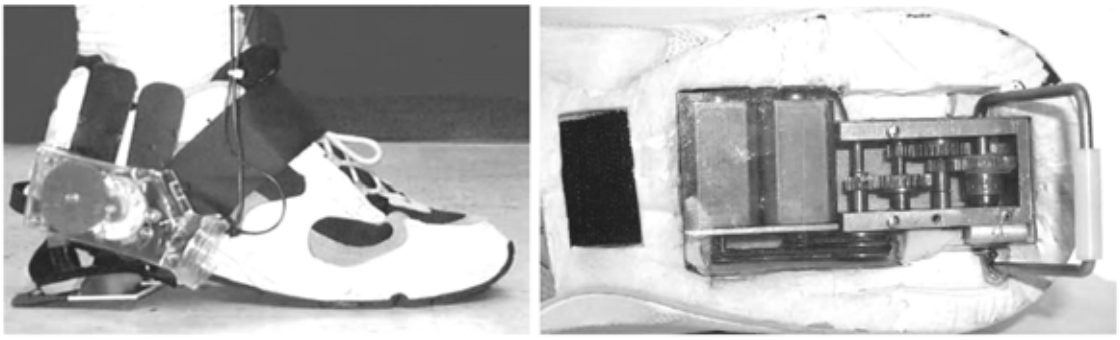


Figura 2.5 – *Shoe generators* (PARADISO, 2006).

Recentemente, duas novas alternativas estão se tornando viáveis para resolver o problema da energia. A primeira prevê a utilização do próprio meio de comunicação (o canal de rádio freqüência) para transmitir a energia elétrica necessária para a recarga das baterias. Nessa abordagem, a energia elétrica é modulada juntamente com os dados transmitidos aos sensores. Um exemplo disso, é o módulo *Wireless Power Platform Powerharvester* (POWERCAST, 2007) que a empresa americana PowerCast pretende comercializar até meados de 2008.

A segunda alternativa refere-se à utilização do campo magnético como força eletromotriz (SHOHAM *et al.*, 2007). Isso funciona como um imã arrastando um alfinete sobre uma superfície de papel. Pensando assim, cientistas israelenses inventaram um microrobô (chamado ViRob), já testado em tubos plásticos e em artérias de animais. Tem um milímetro de diâmetro e quatro de comprimento. O robô rasteja, e pode se agarrar às paredes dos vasos sanguíneos. A direção e a velocidade desse robô são controladas por um campo magnético, sem fios e, também, sem baterias. A Figura 2.6 exibe imagens do ViRob.

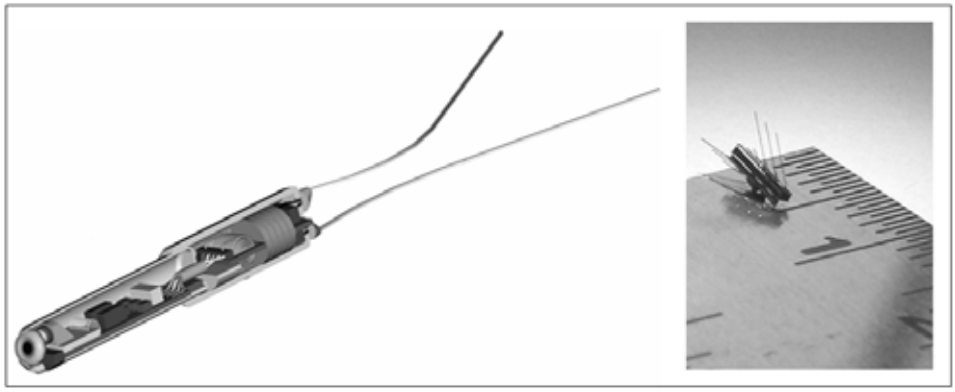


Figura 2.6 – ViRob (modificado - SHOHAM *et al.*, 2007).

2.1.1 – Projetos de vanguarda

A miniaturização e a redução dos custos da tecnologia abrem caminho para uma diversidade de objetos de consumo que da ficção vêm se tornando realidade. Em (ASADA *et al.*, 2003) é apresentado um protótipo de nó sensor sob forma de anel capaz de capturar a freqüência cardíaca e monitorar o nível de oxigênio no sangue. Além disso, o sistema pode calcular a variabilidade da freqüência cardíaca e transmitir todas essas informações sem fios, a uma taxa de 100 kbps (kilobits por segundo). A Figura 2.7 ilustra a implementação do sistema em questão.

Figura 2.7 – Um sensor vestível em forma de anel (ASADA *et al.*, 2003).

AMON (em inglês, *Advanced Telemedical Monitor*) (ANLIKER *et al.*, 2004) é um outro exemplo de sistema vestível. Implementa um dispositivo de alerta por meio de um monitor multiparamétrico (que captura vários tipos de dados) vestível. Integra hardware e software

pelo conceito *All-in-One Wrist-Worn* (em português, tudo num único bracelete vestível). A Figura 2.8 ilustra o AMON.

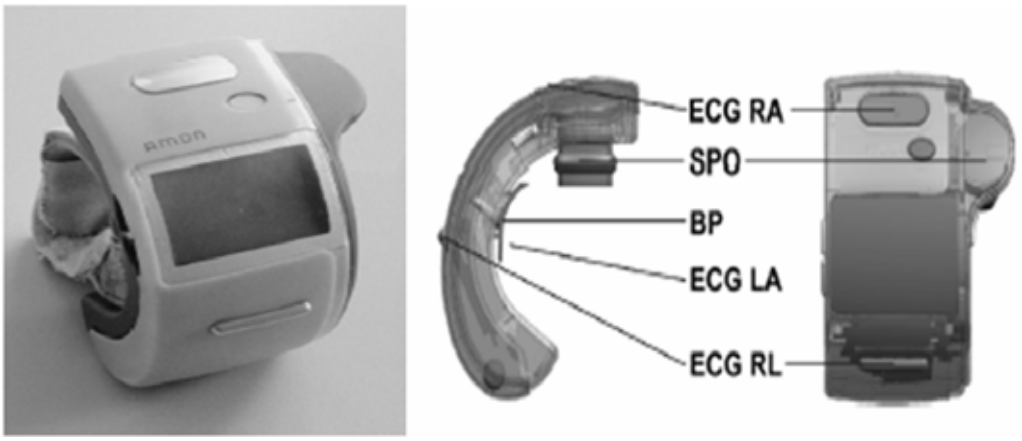


Figura 2.8 – AMON (ANLIKER *et al*, 2004).

As informações capturadas por AMON são: a pressão arterial não-invasiva (sensor BP), o nível de oxigênio no sangue (eletrodo SPO), o ECG (eletrodos ECG LA, ECG RL e ECG RA), o movimento e a temperatura cutânea. Depois de capturadas, as informações são pré-processadas, parcialmente analisadas e, então, transmitidas para uma central por meio de uma interface de comunicação, que acessa a rede de telefonia celular e que está embutida no dispositivo.

Segundo os autores, o aspecto positivo do projeto AMON refere-se à implementação de um sistema que consome pouca energia, tem capacidade para reconhecimento de padrões de informações médicas para diagnóstico clínico. O aspecto negativo refere-se ao resultado de baixa qualidade produzido por alguns dos sensores. Existem ainda limitações técnicas para aplicação do conceito *All-in-One Wrist-Worn*. Por exemplo, o tamanho e a disposição dos eletrodos têm influência direta na qualidade do sinal fisiológico capturado. Desse projeto constatou-se, por exemplo, que o resultado do ECG obtido por meio de um bracelete não é satisfatório para a maioria das aplicações clínicas.

O projeto *Wearable Health Monitoring System* (WHMS) (JOVANOV, 2006) é um dos pioneiros no uso de redes de sensores para o monitoramento humano. Nos últimos sete anos, vários conceitos foram desenvolvidos na Universidade do Alabama, em Huntsville. Por exemplo, o conceito de WBAN (em inglês, *Wireless Sensor Body Area Netwok*), ou

seja, uma rede de sensores sem fios cujo escopo é a área do corpo humano. A topologia WBAN tem influenciado a concepção de alguns novos projetos revelados recentemente em trabalhos científicos.

Numa WBAN, a rede de sensores é hierarquizada pelo modelo cliente-servidor. Cabe aos clientes (nós sensores) o sensoriamento e transmissão dos dados a um servidor. Esse servidor é chamado *Personal Server* (PS). É o responsável pelo processamento das informações capturadas. Como elemento para interconexão da rede de sensores com outras redes tem-se um *gateway* móvel. Em forma de um computador de mão, o *gateway* móvel é o responsável pelo envio dos dados coletados pelos sensores para outros sistemas. A Figura 2.9 detalha os elementos de uma WBAN.

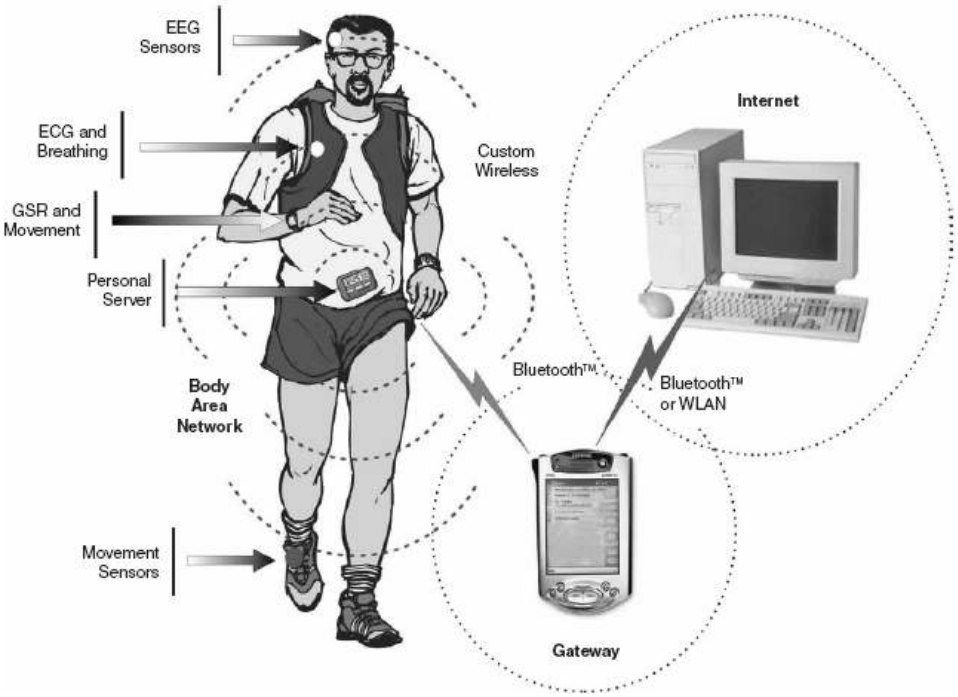


Figura 2.9 - *Wireless sensor Body Area Netwok* (WBAN) (JOVANOV, 2006).

Em (JOVANOV *et al*, 2003) é apresentada uma aplicação desenvolvida no projeto WHMS. Uma rede de sensores projetada para avaliação psiquiátrica de militares sob intenso treinamento. Essa aplicação utiliza de medidas obtidas da freqüência cardíaca para quantificar a variabilidade da freqüência cardíaca e, assim, inferir sobre o nível de estresse antes e durante o treinamento.

O *CodeBlue: Wireless Sensor Networks for Medical Care* (WELSH, 2006) é um projeto desenvolvido pela Universidade de Harvard em colaboração com outras instituições. Tem como objetivo explorar a utilização das redes de sensores para um grande número de aplicações médicas, incluindo emergência, resposta a desastres e reabilitação de pacientes. Juntamente com o hardware, vem sendo desenvolvida uma arquitetura de software de sistema cujo objetivo é viabilizar a integração da rede de sensores com outros sistemas. De acordo com Welsh *et al*. (2004), o CodeBlue é sobretudo uma infra-estrutura de software empregada para construção de um sistema de emergência médica, integrando sensores sem fios para obtenção de sinais vitais, PDAs e PCs.

Quanto às informações monitoradas, elas são: o nível de oxigênio no sangue, o ECG, a pressão arterial, o EMG não-invasivo e o movimento do corpo, utilizado para avaliação das atividades físicas dos pacientes. Como objetivo secundário desse projeto, está a avaliação da arquitetura Motes e do sistema operacional TinyOS, como plataformas de suporte para aplicações médicas. A Figura 2.10 ilustra um dos tipos de nós sensores desenvolvido no projeto CodeBlue.

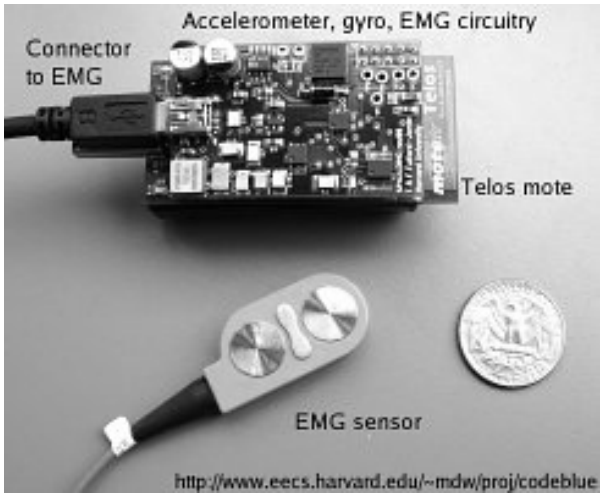


Figura 2.10 – Nó sensor desenvolvido no projeto CodeBlue (WELSH, 2004).

Como exemplo de aplicação desenvolvida no CodeBlue, em (WELSH *et al,* 2005) é apresentado um sistema cujo objetivo é facilitar o atendimento de vítimas de desastres e acidentes. Esse sistema integra informações coletadas no momento do resgate das vítimas com informações de outros sistemas de saúde. Assim, é possível que os profissionais da saúde, à distância, possam dar parecer ou indicar o melhor procedimento a ser executado.

A disponibilização de sistemas computacionais em forma de objetos do cotidiano por si só não garante a utilização desses sistemas. É preciso torná-los confortáveis e práticos. Uma tendência atual é a utilização de novas tecnologias que possibilitem melhor integração dos circuitos eletrônicos ao vestuário das pessoas. A utilização de novos tecidos, compostos por novos materiais, como, por exemplo, a fibra ótica e o algodão possibilitam melhorar a ergonomia e a praticidade do uso de sistemas vestíveis. Um exemplo disso são os chamados *smart textiles* ou *smart fabrics*. A Figura 2.11 ilustra essa nova tendência. Linz *et al.* (2006) desenvolveram um nó sensor de ECG em forma de camiseta. Esse sistema é capaz de operar continuamente por 24 horas com um rádio transmissor Bluetooth ativado. Ainda, pode ser dobrado, esticado e lavado.

Figura 2.11 – *The EKG Shirt* (LINZ *et al.,*2006).

2.2 – A RESPEITO DO SOFTWARE

O software tem papel fundamental em qualquer dispositivo computacional. Nas RSCH é também muito importante. Em relação às funcionalidades atribuídas ao software, destacam-se:

- Interconexão lógica dos nós sensores e da rede de sensores com outras redes, por exemplo, a Internet;
- Abstração do hardware como forma de aumentar a portabilidade[7] e facilitar o desenvolvimento de novas aplicações (aumento da transparência do ponto de vista do programador);
- Capacidade de operação autônoma. É determinada pela eficácia e eficiência dos programas que executam as tomadas de decisão.

A modularização em camadas prevê a existência de diferentes níveis de abstração, desde o hardware até os usuários, agrupando funcionalidades correlacionadas. Esses diferentes níveis compõem uma arquitetura de software. Um conjunto de abstrações implementadas por programas que, em parte, podem estar embutidos nos nós sensores.

Por exemplo, a separação da gerência do hardware (de cada nó sensor) das funcionalidades da rede de sensores pode resultar no aumento da eficiência do código e em facilidade de manutenção do sistema. Isso possibilita acomodar de maneira organizada todos os requisitos do hardware, além de novos protocolos e serviços de rede. Em muitos projetos a camada responsável pela gerência do hardware é chamada de Sistema Operacional (SO), enquanto que, o nível de abstração que gerencia as funcionalidades da rede é chamado de *middleware*.

De acordo com David Culler (CULLER *et al*, 2005), um grande desafio para o projeto de redes de sensores está em selecionar agrupamentos apropriados de abstrações em forma de componentes de software. Esse processo deve visar o aumento da portabilidade sem aumentar o consumo de energia.

[7] Portabilidade é uma propriedade de sistema. Nesse contexto refere-se à possibilidade de utilização dos mesmos programas em diferentes plataformas de hardware.

2.2.1 – Arquitetura em camadas

De acordo com Blumenthal *et al.* (2003) e Hurler *et al.* (2004), a separação do software em blocos funcionais aumenta a flexibilidade do sistema, isto é, a capacidade de crescimento incremental e de adaptação (facilidade para modificação). Assim, uma evolução da plataforma de suporte, especificamente do hardware, não implicará necessariamente na substituição de toda arquitetura de software.

A modularização em camadas tem sido aplicada com sucesso na organização hierárquica dos serviços implementados pelos protocolos das redes de computadores, tal como o modelo TCP/IP (arquitetura Internet). Também, tem servido como inspiração para novos projetos de arquiteturas de software para as redes de sensores. A Figura 2.12 apresenta um modelo hipotético em que os programas e protocolos podem ser agrupados de forma análoga ao modelo TCP/IP formando um conjunto hierárquico de camadas e serviços.

O modelo hierárquico em camadas delimita com clareza as funcionalidades de cada subsistema e possibilita que cada um possa ser substituído quando necessário. Contudo, um requisito inexplorado em relação às redes de sensores refere-se à padronização dos serviços, isto é, como uma camada deve disponibilizar suas funcionalidades a outrem, feito por meio de interfaces. Também, quais devem ser essas funcionalidades e, principalmente, como devem ser desempenhadas.

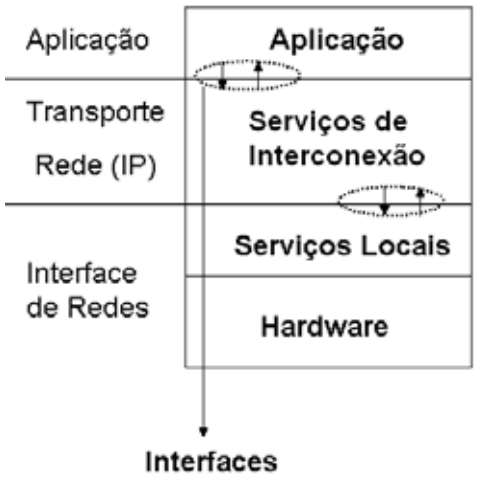


Figura 2.12 - Arquitetura de sistema em camadas inspirada no modelo TCP/IP.

Kiczales (KICZALES *et al.*, 2003) apresenta alternativas que complementam o modelo de modularização funcional em camadas. Desse trabalho, avalia-se que a abordagem mais promissora para as redes de sensores é o uso de meta-interfaces (BARBOSA *et al.*, 2005). Enquanto as interfaces funcionais oferecem transparência aos programadores acostumados ao modelo tradicional, as meta-interfaces devem oferecer translucidez àqueles que necessitam adaptar a plataforma às aplicações (KICZALES *et al.*, 1996). Isto é, enquanto as interfaces funcionais implementam o acesso aos componentes que implementam os serviços da maneira convencional, por exemplo, por chamadas a procedimentos e listas de parâmetros fixos, as meta-interfaces disponibilizam acesso às estruturas internas dos programas que implementam os serviços, para configurações e reajustes.

2.2.2 – Requisitos não-funcionais

Requisitos são funções, condições, atributos, propriedades ou características a serem satisfeitas pelos sistemas. Especificam as funções que o sistema deve ser capaz de executar, ou, dito de outra forma, especificam o serviço a ser prestado pelo sistema (requisitos funcionais). Também, podem estar relacionados às propriedades que o sistema deve possuir (requisitos não-funcionais) (modificado - Staa, 2000, p. 338).

Ravula *et al.* (RAVULA *et al*, 2005) apresenta o conceito de qualidade de software para as redes de sensores. Nesse trabalho, os parâmetros analisados são referentes aos requisitos não-funcionais do *middleware*. Entretanto, esse ponto de vista é pouco abrangente. Privilegia serviços de interconexão em detrimento dos demais.

Neste trabalho, argumenta-se que os seguintes requisitos não-funcionais devem ser considerados para o desenvolvimento do software utilizado em redes de sensores:

- Facilidade de crescimento incremental;
- Generalidade;
- Baseado em componentes;
- Metaprogramado.

O crescimento incremental pode ser alcançado fazendo uso de compiladores customizados em relação às restrições de memória e de processamento do hardware. Por exemplo, com a redução de tipos de dados, variáveis e componentes que não são necessários.

A generalidade possibilita adaptar as interfaces ao invés dos programas. Assim, obtém-se maior clareza do código, desempenho da plataforma, simplicidade do uso e facilidade de manutenção. Por exemplo, uma interface pouco genérica para um componente responsável por calcular a raiz quadrada de um número é "sqrt(int número)". Uma abordagem mais genérica é a interface "raiz(int número, int expoente)" porque por meio desta não é necessário um outro componente para, por exemplo, calcular a raiz cúbica quando essa operação for necessária.

Um sistema baseado em componentes de software interconectáveis deve ser construído pela união de seus módulos independentes. Esses componentes são reutilizáveis e cada um implementa uma funcionalidade bem definida. Assim, é possível tornar viável a construção de plataformas que podem ser criadas ou adaptadas em função das necessidades de cada aplicação ou de uma categoria de aplicações. A adaptabilidade provida pelo uso de componentes facilita a inclusão, a substituição e a remoção de mecanismos e das políticas utilizadas. Entretanto, quando se caminha na direção das políticas, exige modularização de menor granularidade. Isso pode aumentar significativamente o custo de manutenção da arquitetura de software e gerar ineficiência no desempenho.

Em grande parte das vezes o que se busca é o "reajuste de comportamento" de um componente frente a uma nova aplicação ou situação encontrada. Este objetivo pode ser alcançado por meio da metaprogramação.

Metaprogramas são programas que manipulam ou representam outros programas ou a eles mesmos (FRÖHLICH, 2001). Podem ser implementados de duas formas:

- Em tempo de execução. Por meio do conceito de reflexão computacional;
- Em tempo de compilação. Por meio de linguagens de programação multiníveis.

Reflexão Computacional refere-se à capacidade de um sistema "ser racional" sobre seus próprios atos. Em outras palavras, significa capacidade de inspeção e/ou adaptação do próprio software (COULSON, 2003). Um sistema reflexivo mantém sua auto-

representação interconectada de maneira causal (MAES, 1987). Isso implica que mudanças executadas na auto-representação devem ser propagadas para camadas subjacentes. Ainda, pressupõe que sejam considerados dois níveis de abstração: o metanível (auto-representação) e o nível base, responsável pelo interfaceamento com as demais camadas de uma arquitetura de sistema. As meta-interfaces atuam sobre o meta-nível para inspeção e/ou adaptação (configuração), enquanto que, as interfaces funcionais atuam sobre o nível base para acesso aos requisitos funcionais do sistema. O trabalho desenvolvido por Flávia Delicato em (DELICATO, 2005) apresenta um *middleware* reflexivo para redes de sensores. Esse sistema possibilita que as aplicações possam a cada instante de tempo inspecionar e modificar, de acordo com os próprios interesses, as configurações acerca da acurácia dos dados, do consumo de energia e da largura de banda disponibilizada pela rede de sensores.

Por outro lado, a metaprogramação em tempo de compilação ou metaprogramação estática vem sendo adotada por alguns projetos de redes de sensores como forma de aumentar a portabilidade das aplicações (software) sem aumento de demanda por processamento e memória. Para isso, tem-se aplicado o conceito de programação genérica que, na prática, em muitas linguagens de programação refere-se ao uso de tipos genéricos ou, simplesmente, *templates*. O custo dessas abstrações (principalmente em termos de memória) é diluído nos sistemas após a compilação. Um exemplo disso é a implementação de uma camada de abstração de hardware, em inglês *Hardware Abstraction Layer* (HAL). Pelo uso da metaprogramação é possível manter a uniformidade das interfaces sem que haja necessidade de implementar uma nova HAL, para cada novo dispositivo incorporado ao hardware do nó sensor. As HALs passam a ser implementadas em razão de um grupo de funcionalidades comuns e não das diferenças nas tecnologias. Com isso, uma aplicação desenvolvida para ser executada num tipo de hardware pode ser executada também em um outro nó sensor sem muitas modificações.

Em (FRÖHLICH *et al,* 2005) é apresentada uma abstração chamada de mediador de hardware. Essa abstração possibilita adaptar o sistema operacional EPOS (*Embedded Parallel Operating System*) a diferentes tipos de nós sensores. A Figura 2.13 apresenta um exemplo de mediador de hardware implementado para acesso ao barramento GPIO (*General Purpose Input/Output*) de um microcontrolador. Esse componente aplica a metaprogramação estática como forma de abstrair as operações de baixo nível. Ao invés de

usar programas escritos em linguagem *assembly* são utilizados operadores C++ (alto nível) sem nenhuma demanda adicional por memória e processador.

```
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 2.13 - Exemplo de mediador de *hardware* (FRÖHLICH *et al,* 2005).

Outro requisito não mencionado pela literatura científica atual refere-se à padronização de elementos da arquitetura de software como forma de induzir a uma abertura dos sistemas utilizados em redes de sensores. Um sistema aberto está preparado para comunicar-se com qualquer outro sistema aberto. Para isso, deverá dispor de regras padronizadas que governam o formato, o conteúdo e o significado dos interlocutores reconhecidos. É fator preponderante para que o software utilizado em redes de sensores se torne portável e extensível (BARBOSA *et al.*, 2005).

2.2.3 - Requisitos funcionais

Em geral, os requisitos funcionais estão vinculados a uma aplicação. Entretanto, nesta seção serão propostos requisitos funcionais genéricos para as redes de sensores. Esses requisitos foram incorporados e implementados parcialmente pela arquitetura de software de sistema proposta neste trabalho. Também, realçam a necessidade de configuração das redes de sensores de acordo com as necessidades de cada aplicação a cada instante de tempo.

2.2.3.1 - Acesso remoto

O acesso remoto pode ser descrito como o mecanismo que viabiliza a troca de informações. Abstrai a complexidade do acesso à rede para o programador por meio de primitivas, tal como, *"send()" e "receive()"* ou, ainda, por meio de comandos de interface. Essas primitivas (comandos) são encapsuladas em bibliotecas de código e anexadas ao programa da aplicação quando solicitadas.

Diferentes políticas podem ser descritas e implementadas com objetivo de estabelecer uma semântica que organiza o envio e o recebimento das informações, assim como, aquelas que definem os tamanhos de *buffers* onde as informações serão pré-armazenadas.

O aumento da transparência com aumento na demanda por processamento e memória pode resultar no aumento do consumo de energia. Então, o projeto e implementação do acesso remoto deve considerar a utilização de mecanismos e políticas que promovam a adaptação do sistema.

2.2.3.2 - Nomeação, descoberta e roteamento

Nomeação é o mecanismo que possibilita cada sensor ser identificado de forma unívoca. A solução convencional com base no endereço IP (*Internet Protocol*) requer capacidade para manipulação de *bits* proporcional ao número de nós. Essa demanda excessiva por memória inviabiliza o uso do protocolo IP em redes de sensores de média e alta densidade. Entretanto, alternativas que buscam aproveitar a redundância existente entre os possíveis grupos de endereços estão sendo desenvolvidas. Como exemplo, pode-se destacar o uIP (DUNKELS *et al*, 2004), uma implementação do protocolo IP para redes de sensores desenvolvida pelo Instituto Sueco de Ciência da Computação. Esse mecanismo associa informações de localização física dos sensores aos três primeiros octetos da estrutura de endereço IP. Então, se os sensores estiverem numa mesma área (localidade física) os 24 (vinte e quatro) primeiros *bits* do endereço IP podem ser compactados. Isso reduz a demanda por memória e, de acordo com os autores, viabiliza a utilização do IP em redes de sensores de grande escala.

O mecanismo de descoberta oferece a transparência necessária para a localização dos sensores, bem como dos serviços ofertados pelos mesmos. Por exemplo, as consultas aos sensores podem ser baseadas em funcionalidades que os mesmos oferecem, ao invés de nomes ou de outro tipo de identificador de sensor. Esse mascaramento pode ser estático – por meio de uma tabela associativa previamente criada - ou dinâmico - implementado e mantido por meio de um repositório. Esse repositório é responsável pelo gerenciamento de informações referentes a todos os recursos e serviços disponibilizados pela rede de sensores a cada momento.

O roteamento é implementado por um mecanismo que efetua a escolha do melhor caminho para o fluxo de dados entre dois ou mais nós. Em geral, as melhores políticas para efetivar o roteamento dependerão da topologia escolhida para a montagem da rede de sensores. Entretanto, o algoritmo de roteamento pode considerar outros fatores, além da localização física e das funcionalidades, para aumento da eficiência. Um exemplo disso é considerar a energia residual de cada nó sensor para a escolha do melhor caminho. Isso pode evitar que um mesmo grupo de sensores seja sempre escolhido e, como conseqüência, que esses sensores fiquem inoperantes (por falta de energia) mais rapidamente.

As funcionalidades ofertadas pelos mecanismos de nomeação, descoberta e roteamento requerem rapidez, transparência e flexibilidade para troca ou ajuste das políticas que promovam, principalmente, economia no consumo de energia.

2.2.3.3 - Gerência, manutenção e monitoramento dos recursos

Ahamed *et al*. (2004) verifica a possibilidade de dois tipos de políticas empregadas em mecanismos para gerência de redes de sensores:

- Gerência proativa. A ação executada pelo mecanismo antecipa a ocorrência de um evento relativo ao funcionamento da rede;
- Gerência reativa. A ação é tomada após ocorrência do evento.

Nesse contexto, as principais funcionalidades são: *(i)* monitoramento e ajuste do consumo de energia; *(ii)* verificação de falhas; *(iii)* segurança e proteção de recursos (hardware e os dados manipulados); *(iv)* inicialização e configuração remota e *(v)* manutenção da consistência: *(a)* adição e remoção de nós sensores e *(b)* sincronização do tempo.

Entretanto, uma solução que contemple todas essas funcionalidades formando um pacote único de software pode ser impraticável porque tem impacto significativo no desempenho do sistema, especialmente, se esses mecanismos estiverem implementados nos nós sensores. Assim, há necessidade de uma estrutura flexível, por meio da qual o programador possa definir as funcionalidades e as políticas necessárias para a aplicação a cada momento.

2.2.3.4 - Fusão de dados

L. Wald (1999) define o termo fusão de dados da seguinte forma: "fusão de dados é uma estrutura formal na qual estão expressos os meios e as ferramentas para junção dos dados originários de diferentes fontes. Seu objetivo é a obtenção de informações de maior qualidade; a definição exata de 'maior qualidade' irá depender da aplicação".

Em muitas situações é interessante que a rede de sensores disponha de um mecanismo capaz de sintetizar informações a partir da coleta de dados. Esse mecanismo poderá ser utilizado para tomada de decisões e/ou apenas para economizar recursos, por exemplo, evitar a transmissão desnecessária de um dado redundante e, assim, economizar energia.

A fusão de dados pode ser utilizada para combinar tanto dados de sensores do mesmo tipo (que observam a mesma entidade) quanto dados de sensores de tipos diferentes. No primeiro caso, tipicamente as leituras dos sensores são combinadas com o objetivo de eliminar redundâncias e minimizar os ruídos, aumentando a precisão e reduzindo o volume de dados transmitidos. No segundo caso, o objetivo é aumentar a resolução do dado gerando um novo dado mais representativo (SENE Jr. *et al,* 2006).

Por exemplo, para uma aplicação de RSCH que busca diagnosticar trombose [8], detectar apenas a variação de temperatura por meio de sensores localizados num dos membros não garante necessariamente o diagnóstico. É preciso "fundir" a informação de temperatura com outras informações, como, por exemplo, com a pressão arterial capturada por outro nó sensor no mesmo membro.

[8] Trombose é uma patologia caracterizada pela formação de um coágulo (trombo) num vaso sanguíneo. Esse coágulo interrompe a passagem do fluxo sanguíneo. Um dos sintomas da trombose é a diminuição da temperatura cutânea na região onde o fluxo sanguíneo foi interrompido.

A adoção de um mecanismo de cooperação deve ser avaliada de acordo com os requisitos de cada aplicação. Em (CULLER *et al*, 2002) é dito que o consumo de energia para executar mil instruções de código é equivalente ao consumo para transmitir um *bit* de informação a cerca de cem metros de distância por rádio freqüência. Nesse caso, a fusão (agregação) dos dados antes da transmissão pode ser benéfica.

Noutra análise, Vieira *et al*. (2003) verificou que para aplicações com grandes restrições em largura de banda e atraso, a agregação de informações apenas como alternativa para economia de energia, por exemplo, durante o envio de dados, pode ser dispensável. Um exemplo disso é uma RSCH monitorando o sinal de ECG. Nesse caso, provavelmente a densidade da rede é pequena, tem pouca redundância e a quantidade de dados transmitidos é grande. Ainda, em muitas aplicações do ECG, os dados coletados devem ser imediatamente transmitidos.

Assim, para muitas aplicações o mecanismo de cooperação é sinônimo do aumento da produtividade, do desempenho e, até mesmo, do aumento do tempo de funcionamento do sistema. Entretanto, para outras aplicações esse mecanismo é indiferente e, até mesmo, prejudicial pelo consumo desnecessário de memória e processamento.

Dessa forma, a existência de uma plataforma de software flexível – que possibilite a substituição de mecanismos e a troca de políticas – se faz necessária. Porém, a flexibilidade não pode gerar sobrecarga para o sistema, afetando a rapidez com que o mesmo deverá atender às requisições. Ainda, em pontos adicionais de falhas, que podem diminuir a disponibilidade do sistema.

2.2.3.5 – Multiprogramação em redes de sensores

Em um sistema multiprogramado vários programas são mantidos ao mesmo tempo na memória. Esses programas são organizados em tarefas (em inglês, *tasks*). As tarefas compartilham o processador e demais recursos de um mesmo nó sensor durante a execução. A cada tarefa é associada uma funcionalidade. Por exemplo, enviar/receber dados, executar a conversão A/D (Analógico/Digital) de uma amostra de sinal, dentre outras. Atualmente, o principal objetivo de um SO instalado dentro dos nós sensores é fornecer mecanismos (software) que suportem a multiprogramação.

Os principais benefícios da multiprogramação em redes de sensores são:

- Diminuição da necessidade de intervenções humanas. Os nós sensores não precisam ficar em espera do programador (ou responsável pela manutenção do sistema) para executar o *deployment* (carregar em memória e inicializar) de uma nova tarefa (programa) sempre que a execução da tarefa antiga terminar ou precisar ser finalizada. Para isso, o responsável passa a ser um software residente na memória do nó sensor;
- Maior facilidade de programação. Obtida dos mecanismos (bibliotecas de código e interfaces de programação) disponibilizados para desenvolvimento e controle da execução das tarefas. Como conseqüência, a possibilidade de inserção de novos mecanismos para configuração à distância e em tempo de execução.

De acordo com Richard Han (HAN *et al*., 2005), a multiprogramação em redes de sensores pode ser implementada por meio de dois paradigmas distintos: o orientado a eventos e o baseado em *threads.* A principal diferença entre ambos é que o modelo orientado a eventos, usualmente, não oferece suporte à preempção de tarefas. Isto é, as tarefas em execução só poderão liberar o processador diante da ocorrência de uma interrupção (relacionada a um evento) ou quando a execução da tarefa for finalizada.

O modelo orientado a eventos é mais simples. A concorrência entre tarefas é tratada em tempo de compilação. Portanto, necessita de menos memória. Uma única pilha de execução (espaço em memória) e uma estrutura de dados mais simplificada para o gerenciamento das tarefas. Não há troca de contexto (HAN *et al*., 2005).

O contexto de execução de uma tarefa é representado por um registro. O TCB (sigla em inglês, *Task Control Block*) contém todas as informações sobre a execução de cada tarefa. Por exemplo, o contador de programa, os endereços de memória, os registradores e demais recursos alocados. A troca de contexto prevê o salvamento do estado de execução de uma tarefa (uma instância do TCB) para que quando a mesma for recarregada na pilha de execução possa ser reiniciada a partir da última instrução executada anteriormente. O mecanismo que executa a troca de contexto é chamado de *dispatcher*, enquanto que, o mecanismo responsável pela seleção de qual tarefa será a próxima a ser executada é chamado escalonador de tarefas.

O modelo baseado em *threads* comporta melhor tarefas de longa duração, de grande complexidade ou que podem bloquear o processador porque possibilita a troca de contexto. Também, diante de um impasse ou por necessidade da própria aplicação, uma tarefa pode ser suspensa em detrimento de outra tarefa de maior prioridade ou, simplesmente, para evitar que o processador fique parado enquanto há outros programas a serem executados. Esse processo é chamado de preempção. Um sistema com suporte a preempção é chamado preemptivo. No modelo orientado a eventos uma tarefa pode, por exemplo, paralisar todo sistema (nó sensor) enquanto aguarda uma operação de entrada/saída.

2.2.4 – Arquiteturas de Referência

2.2.4.1 - O TinyOS: multiprogramação orientada a eventos

TinyOS é um sistema operacional projetado para suporte a aplicações de propósito geral em redes de sensores. Utiliza o modelo de programação orientado a eventos e tem como principal objetivo a eficiência no uso dos recursos de hardware (HILL, 2003). Foi desenvolvido na Universidade da Califórnia, em Berkeley. É de código aberto e pode ser obtido pelo site http://www.tinyos.net.

No TinyOS as tarefas são organizadas numa fila gerenciada por uma política FIFO (em inglês, *First In First Out*). O sistema é não-preemptivo e não dispõe de mecanismos que associem prioridades à execução das tarefas. Portanto, a ordem de execução das tarefas não pode ser alterada em tempo de execução. Uma tarefa é suspensa apenas quando há necessidade de executar um evento. No TinyOS eventos são mapeados em interrupções. Quando a fila de tarefas está vazia, o processador entra no modo de baixo consumo até a ocorrência de uma interrupção. Essa nova interrupção acionará um conjunto de comandos relativo ao tratamento da mesma e/ou a execução de uma tarefa pré-programada.

O TinyOS, suas bibliotecas e aplicações são programados utilizando uma linguagem de programação chamada de NesC (GAY *et al*, 2003). A NesC[9] oferece uma abstração para a programação que utiliza dois conceitos: composição e construção. Por exemplo, os serviços que gerenciam o hardware são construídos sob forma de componentes que podem

[9] Maiores detalhes a respeito da sintaxe nesC podem ser obtidos em http://nescc.sourceforge.net/.

ser combinados (composição) para construir outros componentes ou combinados para formar uma aplicação completa. O código-fonte dessa aplicação somado aos demais componentes do TinyOS podem então ser compilados pelo compilador NesC. O resultado dessa compilação são programas (em linguagem C) que representam a aplicação. Posteriormente, essa aplicação é compilada para o hardware em uso.

A Figura 2.14 mostra um trecho do código-fonte de uma aplicação programada em NesC. O objetivo dessa aplicação é piscar um *led* de um nó sensor, uma vez por segundo. A aplicação *Blink* é composta por dois componentes: um módulo "BlinkM.nc" e uma configuração "Blink.nc". O arquivo "BlinkM.nc" provê a implementação da aplicação, enquanto que, o arquivo "Blink.nc" é utilizado para descrever quais componentes são necessários e como devem ser interconectados. O grande benefício desse modelo é a agilidade para programação de novos componentes com base no reuso de componentes já existentes.

A interconexão dos componentes acontece por meio das interfaces. A sintaxe da linguagem define o operador "." para separar o componente (elemento da esquerda) da interface (elemento da direta), por exemplo, "Main.StdControl" (veja Figura 2.14). O operador "->" descreve a interconexão dos componentes. Por exemplo, "Main.StdControl-> Blink.StdControl" significa que o componente "Blink" implementa a interface "StdControl" redefinindo a implementação provida pelo componente "Main". Isso pode ser verificado pela observância da palavra chave "implementation {" no arquivo "BlinkM.nc", veja Figura 2.14.

Sobre a plataforma TinyOS vários outros projetos têm sido desenvolvidos nos últimos anos em forma de *middleware*, dentre os quais destacam-se:

- Uma máquina virtual chamada Maté (CULLER *et al*., 2002);
- O TAG, um sistema para extração de informações em redes de sensores. Oferece uma interface para consultas baseada numa linguagem *SQL-like* (MADDEN *et al*., 2002);
- Um sistema de localização para redes de sensores (WHITEHOUSE, 2004);
- TinySEC, uma camada de enlace com criptografia (KARLOF *et al*., 2004);

- Deluge, um sistema que permite reconfiguração pela reprogramação da memória dos nós sensores (HUI & CULLER, 2004);
- Agila, uma plataforma para agentes móveis com suporte à migração de código e, também, à migração do contexto de execução de uma aplicação (migração de tarefas) (FOK *et al*., 2005).

Figura 2.14 – Exemplo de aplicação desenvolvida utilizando NesC e componentes do TinyOS.

Entretanto, a ausência da troca de contexto no TinyOS dificulta:

- A implementação de tarefas mais complexas. Isso requer do programador a utilização de componentes muito pequenos e conhecimento vasto de toda a semântica das operações que serão executadas com objetivo de evitar bloqueios;
- A implementação do conceito de tempo compartilhado (em inglês, *Time Sharing*). Definido como uma extensão lógica da multiprogramação, também chamado de multitarefa (SILBERSCHATZ *et al*., 2001, p.7). Esse conceito prevê a possibilidade de alternar a execução das tarefas no transcorrer do tempo. Ainda, que os usuários possam interagir com cada programa (tarefa) durante a execução do mesmo. A multitarefa possibilita ainda que os nós sensores possam implementar estruturas de decisão mais sofisticadas, podendo aumentar a capacidade de operação autônoma.

Além disso, o modelo de programação proposto TinyOS requer que seus programadores se adaptem à sintaxe e aos conceitos de programação proposto pela linguagem NesC.

2.2.4.2 - O MiLAN

Para uma RSCH, o ideal é ter sempre todos os sensores ativos com o máximo desempenho. Todavia, o atual estágio de desenvolvimento tecnológico das baterias impede que um sistema com tal característica possa ser utilizado por muito tempo. Em questão de horas, esse sistema pode estar inutilizado por falta de energia.

Pensando nisso, foi desenvolvido na Universidade de Rochester o *Middleware Linking Applications and Networks* (MiLAN) (CARVALHO *et al.,* 2003) (CARVALHO *et al.*, 2004). O MiLAN é um *middleware* que tem como principal objetivo a gerência proativa dos parâmetros que determinam a Qualidade de Serviço (*Quality of Service* - QoS) oferecida pela rede de sensores.

O conceito de QoS utilizado pelo MiLAN está relacionado à capacidade da rede de sensores em se ajustar diante dos requisitos de cada aplicação, a cada instante de tempo. Tem objetivo de aumentar o tempo de funcionamento da rede de sensores pelo uso eficiente da energia armazenada nas baterias, resguardando a precisão necessária para cada tipo de informação solicitada a cada momento. Para isso, o MiLAN monitora e ajusta a largura de banda e o consumo de energia de acordo com a importância de cada sensor, no contexto definido para cada aplicação.

No MiLAN, uma aplicação é descrita por meio de variáveis. Uma variável é definida, usualmente, por um ou mais sensores (fontes de dados). Uma variável também pode ser definida com base em outras variáveis. Isso dá origem a uma estrutura hierárquica correspondente a um grafo. A cada aresta desse grafo é associado um valor que expressa como uma variável se relaciona com os sensores e/ou com outras variáveis. Esse grafo é chamado Grafo de QoS dos Sensores. A Figura 2.15 ilustra os grafos que definem uma aplicação usando o modelo proposto pelo MiLAN. Neste caso, a aplicação está relacionada ao monitoramento da freqüência cardíaca (em inglês, *heart rate*). Essa informação pode ser obtida com base em quatro outras variáveis (retângulos) e/ou com base em quatro fontes de dados diferentes (círculos), caso a variável “ECG diag.” seja utilizada. Definida a aplicação, o objetivo do MiLAN é determinar quais os conjuntos de sensores satisfazem os requisitos de QoS para cada variável, a cada instante de tempo. Para descrever esse processo, os autores utilizaram a notação de conjuntos.

Os sensores de interesse para uma dada aplicação definem F_A (em inglês, *application feasible sets*). Cada elemento de F_A é um subconjunto de sensores responsável por prover determinado serviço com qualidade maior ou igual ao mínimo aceitável para cada variável definida pela aplicação.

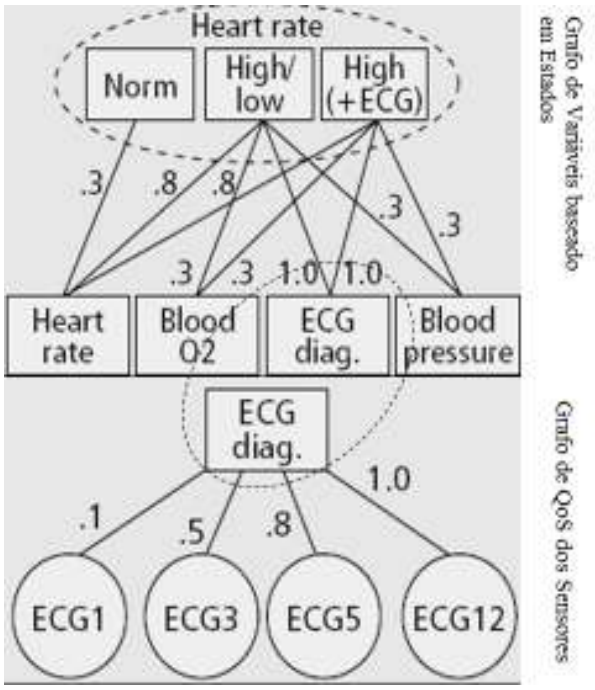


Figura 2.15 – Uma aplicação no MiLAN (modificado - CARVALHO *et al.*, 2004).

O conjunto de nós sensores que podem ser suportados pela rede define $F_{N.}$ (em inglês, *network feasible set*). Como apenas os subconjuntos F_A podem prover o serviço requerido pela aplicação, então, pode-se combinar essas restrições para obter o conjunto de todos os subconjuntos factíveis (F). Isto é, todos os sensores que podem ser utilizados de fato pela aplicação. Para isso, utiliza-se da expressão representada pela Equação 2.1.

$$F = F_A \cap F_N \tag{2.1}$$

O MiLAN deve então escolher o subconjunto de sensores dentre um ou mais subconjuntos de F baseado nos compromissos (*tradeoff*) que podem modificar o desempenho da rede a cada momento. Se F for um conjunto vazio, o MiLAN deve enviar uma exceção para a aplicação. Por outro lado, a decisão de quais elementos devem ser selecionados deve representar a melhor relação entre as necessidades da aplicação e o desempenho da rede a cada instante de tempo. Essa decisão é dependente dos requisitos de cada aplicação e das

condições de operação da rede, por exemplo, o nível de energia. Para expressar as restrições de cada aplicação é utilizado o Grafo de Variáveis Baseado em Estados (veja Figura 2.15). Esse grafo delimita as possibilidades utilizando os possíveis estados que uma variável, que compõe uma aplicação, pode estar. Na Figura 2.15 os estados da variável *heart rate* são representados por retângulos envolvidos por um círculo pontilhado. Na prática, cada estado determina a qualidade do serviço requisitada para cada situação que a aplicação possa estar envolvida.

O MiLAN apresenta uma arquitetura modular e em camadas, composta por um núcleo (*core*) e *plugins.* Os *plugins* são responsáveis pela tradução (em inglês, *mashalling/unmashalling*) dos parâmetros entre o MiLAN e a infra-estrutura subjacente: protocolos de rede, SO e hardware. É papel dos *plugins* determinar quais conjuntos de nós sensores podem ser suportados pela rede, bem como informações específicas sobre os protocolos. Por exemplo, qual a funcionalidade que cada nó sensor é capaz de desempenhar. Para isso, o MiLAN é compatível com dois serviços de descoberta: o Bluetooth SDP (*Session Description Protocol*) (AVANCHA, 2002) e o SLP (*Service Location Protocol*) (IETF, 2001). A Figura 2.16 mostra como o MiLAN se adapta facilmente aos diferentes tipos de rede pelo uso de uma arquitetura em camadas.

Por outro lado, o núcleo (*core*) é o responsável por receber os grafos com as especificações de QoS e dos estados da aplicação, calcular os parâmetros acerca dos ajustes necessários e, então, executar os ajustes.

2.2.5 – Tendências

De acordo com Lu *et al.* (2004), o *Wireless World Research Forum* (WWRF) considera como necessário para a próxima geração de sistemas *wireless* – conhecida como *beyond the third generation* (B3B) ou *fourth generation* (4G) – as seguintes características:

- Interfaces intuitivas com aplicações, serviços e dispositivos remetendo o controle ao usuário;
- Serviços e aplicações serão personalizados e adaptados de acordo com o ambiente. Sendo ubíquos do ponto de vista do usuário;

- Serviços adequadamente integrados serão providos aos usuários, grupos de usuários, comunidades e máquinas, independentemente, do lugar e da tecnologia de interconexão utilizada. Ainda, com QoS previamente estabelecida;
- Os desenvolvedores poderão criar ainda com eficiência e facilidade novos serviços e modelos de negócio baseados nos componentes da arquitetura do mundo *wireless*.

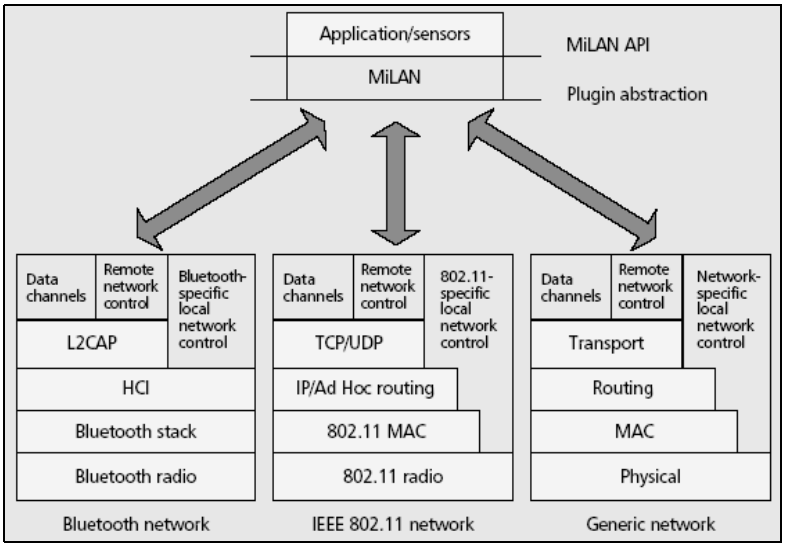


Figura 2.16 – Arquitetura do MiLAN e a interação com outras arquiteturas de software (CARVALHO *et al*, 2004).

Contudo, os sistemas 4G devem integrar, complementar e unificar antes de competir com a infra-estrutura de acesso existente. Nesse contexto, a importância do modelo de software é cada vez maior. Sistemas *wireless* pouco adaptáveis tendem a ser inutilizados mesmo antes do término da prototipação ou, ainda, serem suplantados a cada nova incursão de tecnologia.

2.3 – A RESPEITO DO HARDWARE

Em relação ao hardware, de maneira geral, as aplicações que utilizam as redes de sensores sem fios apresentam os seguintes requisitos:

- Baixo custo, alcançado pelo uso de componentes ou blocos funcionais disponíveis comercialmente e popularizados pela indústria;
- Flexibilidade, isto é, capacidade de crescimento incremental e/ou facilidade para substituição de módulos;

- Capacidade de processamento, memória e transmissão sem fios (*wireless*);
- Baixo consumo de energia elétrica.

Atualmente, o baixo consumo de energia é verificado por meio de duas características distintas, que podem estar presentes nos dispositivos eletrônicos disponíveis comercialmente. A primeira delas é chamada *low power.* Relaciona o consumo de energia elétrica por ciclo de trabalho. A segunda é chamada *energy-efficiency*. Essa representa o consumo de energia por instrução executada. Por exemplo, o processador da família ATMega128L quando operando a 4 MHz (megahertz), consome cerca de 16,5 mW (miliwatts) com eficiência de 242 MIPS (*Million Instructions per Second*) por Watt (W). Isso é equivalente a 4 nJ (nanojoules) por instrução. Já, um processador da família ARM operando a 40 MHz consome 75 mW com eficiência de 480 MIPS/W (2,1 nJ/instrução) (VIEIRA *et al.,* 2003).

Tratando-se da implemntação de uma RSCH não-invasiva, uma arquitetura de nó sensor vestível vem se tornando popular. O modelo apresentado pela Figura 2.17 é de baixo custo, de fácil implementação e eficaz, para a maioria das aplicações. Foi utilizado pela primeira plataforma de redes de sensores sem fios disponível comercialmente, a plataforma Motes (CROSSBOW, 2006), concebida na Universidade da Califórnia, em Berkeley. Atualmente, muitos projetos acadêmicos de redes de sensores vestíveis utilizam este modelo de arquitetura. Dentre os quais se destacam o Codeblue (LORINCZ *et al*., 2004), o WHMS (JOVANOV *et al*., 2006) e o UbiMon (ICL, 2006).

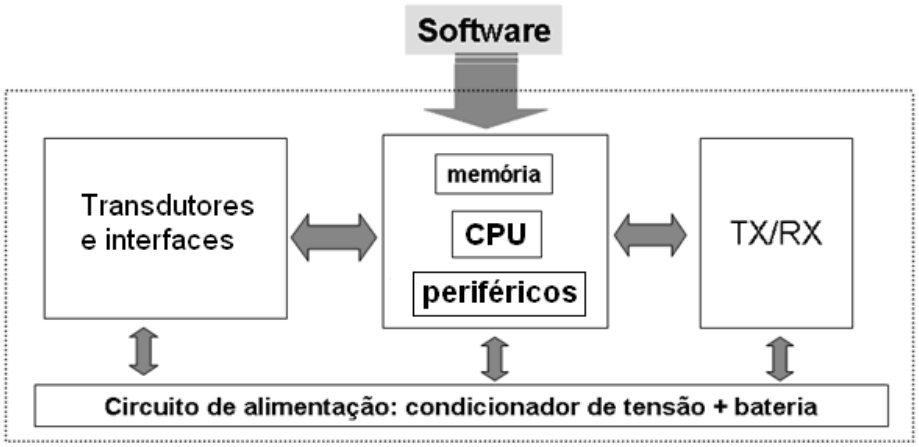


Figura 2.17– Arquitetura de um nó sensor genérico.

O projeto e a implementação de um nó sensor para uma RSCH oferece ainda desafios relativos aos transdutores e interfaces, à adequação do circuito de alimentação, à escolha da unidade de processamento e à escolha da tecnologia para interconexão da rede.

Os circuitos que interagem com o corpo humano por meio de eletrodos necessitam ser isolados para evitar acidentes. Também, devem ser pequenos, leves e minimamente obstrutivos, promovendo a ergonomia. Ainda, é importante considerar questões químicas e biológicas que podem afetar a biocompatibilidade. Por exemplo, alguns metais podem causar reações imunológicas provocando rubor na superfície da pele. Por outro lado, a escolha dos componentes eletrônicos precisa ser pautada na precisão, em razão da natureza das grandezas monitoradas, na adequação aos demais componentes utilizados e no custo. Por exemplo, em alguns circuitos para captura de sinais eletrofisiológicos é necessária a utilização de um módulo adicional para compatibilização das tensões elétricas entre os amplificadores e demais módulos, um conversor de tensão. Esse dispositivo pode aumentar o consumo de energia, aumentar o tamanho do circuito e, ainda, aumentar significativamente o custo do nó sensor.

A escolha da unidade de processamento deve-se pautar na demanda da aplicação por processamento, memória e periféricos. Ainda, resguardar o custo e, principalmente, o consumo de energia. Os microcontroladores fornecem solução para muitos tipos de aplicações por encapsularem a maioria das funcionalidades dentro de um único circuito integrado, facilitando a prototipação e reduzindo o custo. Em inglês, esse conceito é conhecido como *System-on-a-chip* (SoC). O elemento central da arquitetura mostrada na Figura 2.17 é um microcontrolador. A Tabela 2.1 apresenta um comparativo dos principais sistemas microcontroladores utilizados atualmente em projetos de RSCH.

Além de proporcionarem baixo consumo de energia (*low power*), circuitos integrados que usam tecnologia CMOS (*Complementary Metal Oxide Semiconductor*), em geral, disponibilizam mecanismos que podem ser manipulados por políticas para utilização mais eficiente da energia elétrica. Atualmente, são duas as principais abordagens que fazem uso dessa funcionalidade. A primeira fundamenta-se na possibilidade do desligamento de partes do circuito integrado (em inglês, *Dynamic Power Manager* - DPM). A segunda explora a possibilidade de ajuste da freqüência de operação do circuito em relação à tensão de alimentação (em inglês, *Dynamic Scaling Voltage* – DSV).

A idéia básica por trás da DPM envolve tanto hardware quanto software. Com base numa análise estocástica de eventos passados, o sistema tenta prever quando e quais podem ser os eventos futuros e, assim, determinar quais elementos do circuito devem estar ativos. Para isso, muitos circuitos integrados dispõem de um mecanismo (uma máquina de estados) com pelo menos três modos de operação: ativo, em espera (em inglês, *idle*) e desligado.

A DSV reduz o consumo de energia em um circuito CMOS reduzindo a tensão de alimentação fornecida a esse circuito, concomitantemente, com a redução da freqüência de operação. Variando a tensão de alimentação de acordo com a freqüência de operação consegue-se obter uma redução quadrática do consumo de energia.

O microcontrolador MSP430F149 vem se firmando como o mais popular para aplicações biomédicas de redes de sensores. Possui uma unidade central de processamento do tipo RISC (*Reduced Instruction Set Computer*) de 16 bits, integrada com memória RAM (*Random Access Memory*) e memória *Flash.* Possui também vários periféricos. Sua principal característica está relacionada ao consumo extremamente baixo de energia elétrica por ciclo de trabalho, embora, tenha eficiência aproximada de 0,361 nJ/instrução. Outra característica importante do MSP430 é a disponibilização de um conversor Analógico/Digital (A/D) com resolução de 12 bits. Isso confere melhor qualidade aos sinais eletrofisiológicos capturados. Ainda, em muitos casos evita a necessidade de incluir outro conversor A/D no projeto do nó sensor.

O MSP430 possui seis modos de operação comandados pelo oscilador em uso. Cada um desses modos habilita um conjunto de periféricos. Isso viabiliza a implementação da DPM. Além disso, o MSP430 pode aplicar parcialmente a DSV porque possibilita mudanças na freqüência de operação do microcontrolador, permitindo reduzir o consumo de corrente. Para isso, o MSP430 disponibiliza um oscilador interno controlado por software, o DCO (*Digitally Controlled Oscillator*). Entretanto, o MSP430 precisa de hardware e de software adicional para regular a tensão fornecida pela fonte de alimentação.

A tecnologia utilizada para interconexão dos nós sensores é responsável por assegurar que as informações capturadas sejam transmitidas. A utilização de um módulo transmissor/receptor (TX/RX) por meio de Rádio Freqüência (RF) é a solução mais

empregada atualmente. Em RSCH, essa tecnologia possibilita mobilidade, oferece alcance satisfatório e custo razoável. Entretanto, os experimentos indicam que esse módulo determina o tempo de funcionamento do sensor, pois, usualmente, é maior consumidor de energia.

Tabela 2.1 – Comparativo de alguns microcontroladores utilizados atualmente.

Características	**ATmega128** (ATMEL, 2006)	**PIC18F2550** (MICROCHIP, 2007)	**MSP430F149** (TI, 2006)	**ARM AS-1100** (SINHA & CHADRAKASAN, 2005)
bits	8	8	16	32
Flash	128 KBytes	32 KBytes	60 KBytes	-
RAM	4 Kbytes	2048 bytes	2048 bytes	-
ADC	10 bits	10 bits	12 bits	-
Timers	3	2	3	-
Porta USB	-	USB 2.0	-	-
Tensão de Operação	2,7-5,5v	2,0-5,5v	1,8-3,6v	3-3,6v
Consumo: modo ativo	1,5mA	2,5 mA typical @ 5V	340µA @ 3v	230mW@133MHz
Consumo: modo espera	1,6mA	1,1 mA	1,3 µA	50mW@133MHz
Consumo: desligado	<1µA retenção RAM	<1µA retenção RAM	<0,1µA retenção RAM	Tipicamente 25µA

Em geral, esses módulos disponibilizam mecanismos que possibilitam regular (reduzir) o alcance de transmissão e, como conseqüência, a demanda por energia. Essa estratégia é interessante para uma RSCH porque a área de cobertura é pequena (no máximo o tamanho do corpo). Outra estratégia, proposta por Carvalho *et al*. (2003b), argumenta que em muitas situações é interessante transmitir a informação capturada do paciente em intervalos de tempos definidos de acordo como o estado de saúde do mesmo. Nos momentos de inatividade (o sensor está apenas armazenando), o rádio transmissor é desligado ou fica em espera (*idle*). Essa estratégia possibilita economizar energia elétrica sem comprometer o monitoramento da saúde do paciente.

Hoje em dia, as tecnologias mais empregadas para transceptores RF utilizados em rede de sensores são: o Bluetooth (TOBS, 2006) e o Zigbee (ZA, 2006). Os dispositivos Bluetooth fazem parte do cotidiano das pessoas. São facilmente encontrados em telefones celulares, em computadores pessoais e, até mesmo, em carros.

Em abril de 2003, o grupo de trabalho IEEE 802.15 WPAN™ *Task Group* 4 (TG4) (2006) conseguiu a aprovação da norma IEEE 802.15.4. Essa norma especifica requisitos para um

modelo de rede sem fios, capaz de operar em 2,4 GHz (gigahertz), com taxas de até 250 Kbps e, ainda, que assegura pouco consumo de energia.

Pensando nisso, uma associação de empresas - a ZigBee Alliance - vem trabalhando com objetivo de viabilizar uma tecnologia de rede sem fios que seja confiável, de custo acessível e de pouco consumo de energia elétrica, cujo foco serão aplicações de monitoramento e controle. Essa tecnologia é conhecida como Zigbee. O *transceiver* Chipcon CC2420 (CHIPCON, 2006) foi a primeira implementação Zigbee empregada em rede de sensores sem fios.

2.3.1 – Tendências

Como tendências acerca do projeto do hardware para RSCH, verificam-se:

- Aumento da capacidade de reconfiguração do nó sensor;
- Obtenção da energia elétrica com base no ambiente monitorado;
- Utilização de outros meios de comunicação.

O uso de elementos como *Field Programmable Gate Array* (FPGA) e *Complex Programmable Logic Device* (CPLD) vem se tornando cada vez mais freqüente. O objetivo desses dispositivos é permitir que a lógica dos circuitos eletrônicos possa ser concebida e alterada por software. Em geral, a inserção desses dispositivos aumenta o custo e o consumo de energia do sensor, entretanto, facilita a reconfiguração do mesmo. Isso pode se tornar benéfico quando é necessária a substituição dos componentes eletrônicos do hardware. Em (JOVANOV *et al.,* 2004) é apresentado um nó sensor para o monitoramento do nível de oxigênio no sangue. Esse sistema conta com um CPLD para geração de sinais de controle utilizados no processamento digital dos sinais capturados. Com isso, é possível executar ajustes no sistema com objetivo de manter a precisão necessária para a informação monitorada. Mesmo quando há mudanças no ambiente e/ou no estado de saúde do paciente.

Segundo Joe Paradiso (PARADISO & STARNER, 2005), sistemas computacionais podem obter a energia elétrica necessária para o funcionamento com base na conversão da energia mecânica obtida das atividades humanas, por exemplo, uma caminhada. Podem ainda, converter a energia do ambiente onde se encontram, por exemplo, calor, luminosidade,

ondas de rádio e vibrações em energia elétrica. De acordo com Chandrakasan (CHANDRAKASAN *et al*., 2004), ao ultrapassar a barreira de consumo de aproximadamente 100μW (microWatts), teoricamente, os nós sensores podem ser alimentados pela energia extraída do seu ambiente operacional sem a necessidade de transdutores muito sofisticados. Em inglês, isso é chamado de *Energy Scavenging.*

Em (STARK, 2006) é apresentado o *Thermo Life*. De pequeno tamanho (1,4 mm), esse sistema é capaz de produzir energia elétrica pela variação da temperatura do ambiente em que se encontra. Numa RSCH, por exemplo, se a diferença entre a temperatura cutânea de uma pessoa e do ambiente onde ela se encontra for aproximadamente 5^{o} K (Kelvin), esse sistema pode ser utilizado para gerar aproximadamente 30 μW (microwatts) de potência com tensão igual a 3 V (volts).

Em relação à utilização de outros meio de comunicação, um bom exemplo de inovação refere-se à pesquisa desenvolvida por cientistas do Massachusetts Institute of Technology (MIT) e do IBM's Almaden Research Center[10]. Os resultados das pesquisas desenvolvidas nesses institutos indicam que, no futuro, os atuais enlaces de RF poderão ser substituídos por um meio de comunicação nunca antes explorado: a pele do corpo humano. Explorando a salinidade da pela humana como condutor de eletricidade, pesquisadores conseguiram desenvolver uma tecnologia capaz de transmitir informações em taxas maiores que 400 Kbps (kilobits por segundo) utilizando níveis muito baixos de potência.

[10] Veja http://www.almaden.ibm.com/cs/user/pan/pan.html.

3 – ESPECIFICAÇÕES E PROJETO

Neste capítulo são apresentados os elementos relativos ao projeto de uma arquitetura de redes de sensores do corpo humano. São apresentadas também as principais especificações dos sistemas que implementam a arquitetura proposta.

3.1 – METODOLOGIA ADOTADA

O processo utilizado para o desenvolvimento da arquitetura proposta neste trabalho baseia-se fortemente na metodologia de desenvolvimento ágil Programação eXtrema, ou simplesmente, XP[11]. XP é uma metodologia definida para desenvolvimento de sistemas e indicada para equipes pequenas e médias. Essas equipes trabalham em geral com requisitos pouco delimitados e, principalmente, mutáveis.

As quatro características fundamentais da metodologia XP são: interação entre os participantes (comunicação); simplicidade, ou seja, o foco principal está no resultado gerado, por exemplo, o sistema em funcionamento é mais importante que a documentação relacionada a esse sistema. Ao contrário das metodologias mais tradicionais, a documentação é um apêndice e não o principal. A terceira característica é a constante colaboração do cliente com a equipe de desenvolvimeno de software (conhecida como *feedback*). A quarta característica é chamada de coragem, isto é, a equipe de desenvolvimento ser capaz de se adaptar à mudanças durante o processo (Kon, 2007).

XP é particularmente apropriada para o tipo de projeto desenvolvido neste trabalho por compartilhar os valores supracitados. Em especial, destacam-se:

- Privilegiar o desenvolvimento incremental, com liberação freqüente de novas versões em produção, valorizando o *feedback* do especialista da área. Avalia-se que esta característica é importante para garantir resultados efetivamente úteis em menor espaço de tempo;
- Ser fortemente baseada em testes automáticos. Esta característica provê a garantia necessária para o desenvolvimento dos sistemas, sem comprometer a qualidade esperada.

[11] http://www.extremeprogramming.org/.

Para especificação da arquitetura SOAB e dos sistemas implementados foi utilizada a linguagem UML[12] (*Unified Modeling Language*). UML é uma linguagem para modelagem de sistemas não proprietária. Ela possibilita documentar sistemas de maneira mais eficiente, isto é, sintetizada e visual (por meio de diagramas). UML é considerada uma das linguagens mais expressivas para modelagem de sistemas. Por meio de seus diagramas é possível representar sistemas em diversas perspectivas de visualização, facilitando a comunicação de todas as pessoas envolvidas no processo de desenvolvimento.

3.2 – ATRIBUTOS DE QUALIDADE

Muitas vezes a avaliação de sistemas 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 o projeto. Esse tipo de avaliação é impreciso e, em geral, avalia apenas a conformidade dos requisitos funcionais especificados no projeto. Um exemplo disso é apresentado em (ROCHA *et al*., 1995). No trabalho em questão, o autor utiliza parâmetros (atributos) de qualidade para um sistema especialista desenvolvido para diagnóstico de infarto agudo do miocárdio. A definição dos atributos é baseada num modelo próprio denominado *Rocha's quality evaluation model* e não em normas internacionais.

O projeto, a implementação e a avaliação dos sistemas propostos nesta tese foram executados com base nas 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 de qualidade para o software. Esses parâmetros são chamados características e subcaracterísticas em (ISO, 2001) e fatores e subfatores em (IEEE, 1993). A Tabela 3.1 apresenta esses parâmetros. Além desses atributos de qualidade, foram considerados os requisitos não-funcionais apresentados na Seção 2.2.2. Esses requisitos expressam as necessidades de um domínio de aplicações, por exemplo, as redes de sensores do corpo humano. Portanto, devem ser considerados.

[12] http://www.uml.org/

Tabela 3.1 – Parâmetros de qualidade para o software.

Características	Objetivos	Subcaracterísticas	Descrição
Funcionalidade	Verificar se as funções e propriedades satisfazem todas as necessidades implícitas e explícitas dos usuários	Adequação	Relativo à presença das funcionalidades especificadas
		Acurácia (*Accuracy*)	Averiguar se o produto gera resultados precisos ou dentro do esperado
		Interoperabilidade	Capacidade de interagir e interoperar com outros sistemas, de acordo com o especificado
		Conformidade	Observância a padrões, convenções ou regras estabelecidas
		Segurança de acesso	Capacidade para prevenir o acesso não autorizado
Confiabilidade	Determinar quão imune à falhas é o software	Maturidade	Indicação da freqüência de falhas
		Tolerância a falhas	Capacidade do produto para manter determinados níveis de desempenho mesmo na presença de problemas
		Recuperabilidade	Capacidade do produto para re-estabelecer o nível de desempenho desejado e recuperar dados em caso de ocorrência de falha
Usabilidade	Constatar a facilidade de uso	Inteligibilidade (*Understandability*)	Medida do esforço necessário para entendimento do software
		Facilidade de aprendizado	Facilidade encontrada pelo usuário para aprender a utilizar o produto
		Operacionalidade	Facilidade para operar o produto, incluindo: cores, formatos de tela e sons
		Comunicabilidade (*Communicativeness*)	Facilidade que o software oferece para interação com os usuários de acordo com suas características psicológicas
Eficiência	Averiguar economia de tempo e/ou de recursos	Tempo	Medida do tempo de resposta e de processamento
		Recursos	Medida da quantidade de recursos necessários (CPU, disco e memória, dentre outros)
Manutenibilidade	Verificar a facilidade para modificação e testes	Analisabilidade	Medida do esforço necessário para diagnosticar deficiências ou causas de falhas, ou localizar as partes a serem modificadas para corrigir os problemas
		Modificabilidade	Medida do esforço necessário para realizar alterações, remover falhas ou para adequar o produto a eventuais mudanças de ambiente operacional
		Estabilidade	Medida do risco de efeitos inesperados provenientes de modificações
		Testabilidade	Medida do esforço necessário para testar o software alterado
Portabilidade	Averiguar a possibilidade de utilização do sistema em outro ambiente de execução	Independência do hardware	Facilidade de se adaptar o produto para funcionar em outros ambientes operacionais (quanto ao hardware)
		Independência do software	Facilidade de se adaptar o produto para funcionar em outros ambientes operacionais (quanto ao software)
		Reusabilidade (*Reusability*)	Facilidade para reuso do produto em outras aplicações
		Facilidade de Instalação (*Installability*)	Medida do esforço necessário para se instalar o produto

3.3 – VISÃO GERAL DA ARQUITETURA

De acordo com Ruiz *et al.* (2004), a organização de uma rede de sensores pode ser:

- Hierárquica, quando os nós estão organizados em grupos (*clusters*). Cada grupo tem um líder (*cluster-head)* eleito pelos demais membros do grupo e, ainda, os grupos podem organizar hierarquias entre si;
- Plana, quando os nós sensores não estão organizados em grupos.

Ainda, segundo Ruiz *et al.* (2004), o ponto de acesso é o elemento por meio do qual a rede comunica-se com outras redes ou com um ou mais observadores. O ponto de acesso pode ser implementado em um nó sensor chamado nó sorvedouro (em inglês, *sink node*), ou, ainda, em uma estação base.

Em redes de sensores do corpo humano, a topologia utilizada tem organização plana e, usualmente, tem formato de estrela. O elemento central é o LPU (*Local Processing Unit*). Na maioria dos projetos, o LPU é também o ponto de acesso. O ponto de acesso recebe os dados capturados dos sensores e os disponibiliza para outros sistemas, ou para os usuários interessados. É também funcionalidade do ponto de acesso encaminhar aos nós sensores instruções para reajustes e configurações. Neste trabalho, o termo *gateway* é utilizado para representar, de maneira genérica, os possíveis pontos de acesso.

Esse modelo organizacional é utilizado em razão do grande volume de dados a serem transmitidos com requisitos de tempo real, isto é, com mínimo possível de perdas, atrasos e *jitter* (variação do atraso). Os requisitos acerca das taxas de transmissão e largura de banda para monitoramento de alguns sinais eletrofisiológicos torna inviável, em muitas aplicações, a utilização de um modelo hierárquico com muitos níveis (muitos saltos). Por exemplo, para um nó sensor de ECG que monitora doze canais com amostragem de 1000 Hz por canal, dependendo da resolução, pode gerar continuamente 192.000 bits/s de dados. Isto é cerca de três vezes maior que o especificado para um canal de voz no formato digital PCM (*Pulse Code Modulation*). Ainda, esses dados, em muitas aplicações, devem ser transmitidos de maneira a minimizar perdas e atrasos.

A Figura 3.1 oferece uma visão geral do sistema que foi desenvolvido para o monitoramento da saúde humana. Esta figura apresenta também a disposição dos principais

artefatos da arquitetura SOAB (apresentada da Seção 1.3), ou seja, como a SOAB está presente em cada um dos elementos do sistema.

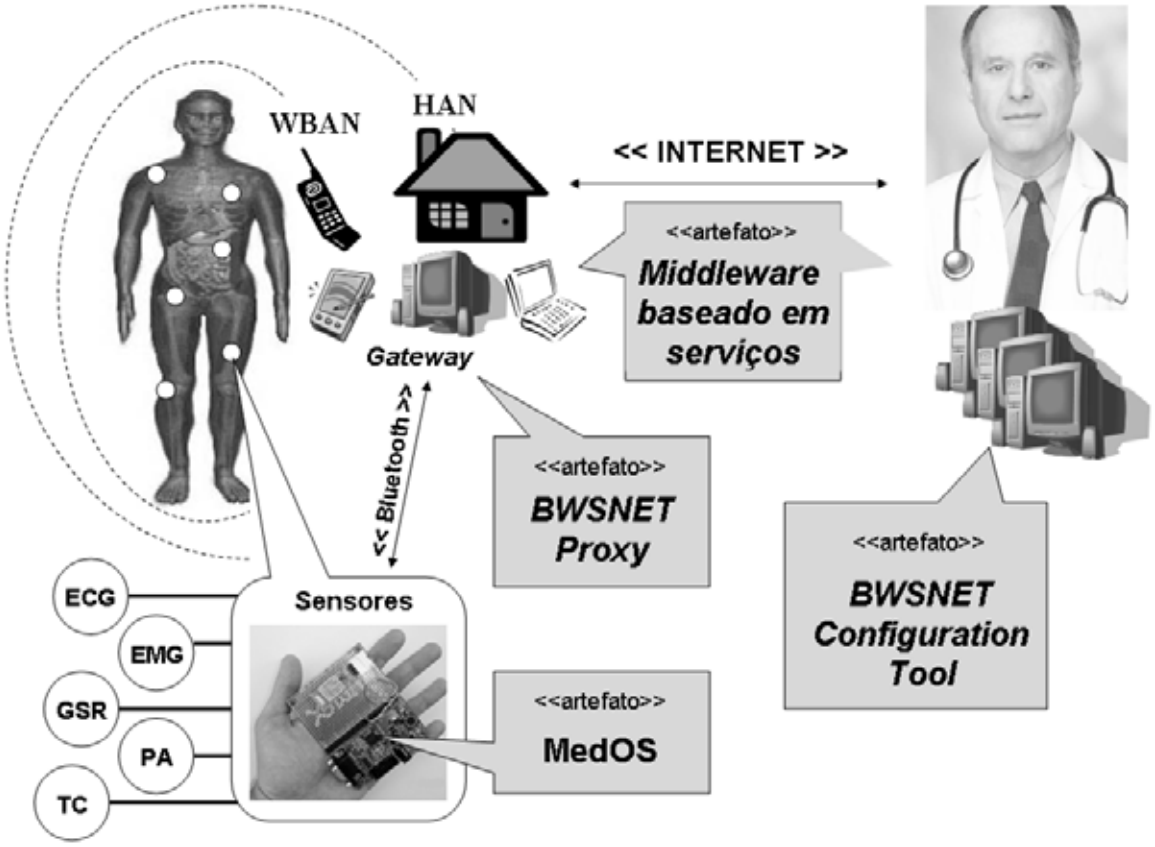


Figura 3.1 – Visão geral do sistema proposto.

Uma WBAN (*Wireless Body Area Network*) refere-se à área de cobertura do corpo humano, enquanto que, uma HAN (*Home Area Network*) refere-se ao ambiente, em curto espaço, no qual o paciente pode estar inserido. Por exemplo, um hospital, um escritório ou a própria residência.

Os dispositivos de *gateway* podem ser PCs (*Personal Computers*), *laptops*, PDAs (*Personal Digital Assistants*) ou, até mesmo, um telefone celular. Esses dispositivos devem possuir pelo menos uma interface de rede compatível com a rede de sensores do corpo humano.

O Bluetooth é particularmente interessante para implementação da comunicação entre os sensores e o *gateway* (veja Figura 3.1). Dada a quantidade de sensores não-invasivos,

avalia-se que uma piconet Bluetooth acomoda bem esses nós sensores (como *slaves*) que intercomunicam com o *gateway* (o *master*). Além disso, é importante que o alcance e as taxas de transmissão dos sensores possam ser ajustados, com objetivo de economizar energia e minimizar perdas e atrasos. Bluetooth possibilita esses ajustes pelas diferentes classes de serviço e modos de operação. Entretanto, o aspecto decisivo é o fato do Bluetooth ser um padrão para redes sem fios de curto alcance e aplicações de propósito geral e, principalmente, estar disponível em muitos objetos de consumo. Isso possibilita o monitoramento ininterrupto e em qualquer lugar que o paciente se encontre. Esse conceito é conhecido como monitoramento ubíquo e pervasivo (anglicismo do termo *pervasive*).

Outro aspecto do sistema apresentado na Figura 3.1 refere-se à existência de sensores multifuncionais. São chamados nós sensores multifuncionais (em inglês, *multimodal*) os nós sensores compostos por interfaces e circuitos analógicos para aquisição e condicionamento de mais de um tipo de dado. Em contrapartida, os nós sensores monofuncionais dispõem de interfaces e circuitos analógicos para lidar com apenas um tipo de dado.

De acordo com Carvalho (2005), dado refere-se a uma medida obtida com base no ambiente monitorado, enquanto que, uma variável é determinada pela análise dos dados (extração de características). Portanto, a informação de interesse pode estar relacionada tanto aos dados quanto às variáveis.

Os nós sensores multifuncionais devem ser vistos como unidades autônomas capazes capturar e/ou sintetizar mais de um tipo de dado relevante à aplicação. Neste trabalho, esta definição é importante porque possibilita que as abstrações providas pelo software, para programação e configuração dos nós sensores, possam incluir, de uma forma diferente, os conceitos apresentados na Seção 2.2.3.5, como, por exemplo, o conceito de multitarefa ou multiprogramação baseada em *threads*.

A utilização de nós sensores multifuncionais amplia a topologia do sistema (apresentado na Figura 3.1), transformando cada nó sensor num grafo de funcionalidades, mostrado na Figura 3.2. Cada elemento desse novo grafo representa uma funcionalidade provida pelo nó sensor. O tamanho desse grafo é delimitado pela diversidade de sensores disponíveis e

pelas tarefas propostas para cada sensor. As tarefas podem estar em função tanto dos dados quanto das variáveis.

Um dos pontos explorados nesta tese propõe a utilização de políticas específicas para o escalonamento das funcionalidades de cada nó sensor. Isso tem objetivo de alcançar maior desempenho para cada aplicação. Essas políticas devem considerar requisitos do sistema, como, por exemplo, energia e memória disponível em cada nó sensor e, ainda, requisitos da própria aplicação como, por exemplo, por quanto tempo uma funcionalidade deve ser ativada e com qual intensidade. Essa nova capacidade atribuída aos nós sensores possibilita maior independência do *gateway* para tomadas de decisão, reajustes e configurações. Portanto, maior autonomia ao sistema (nós sensores).

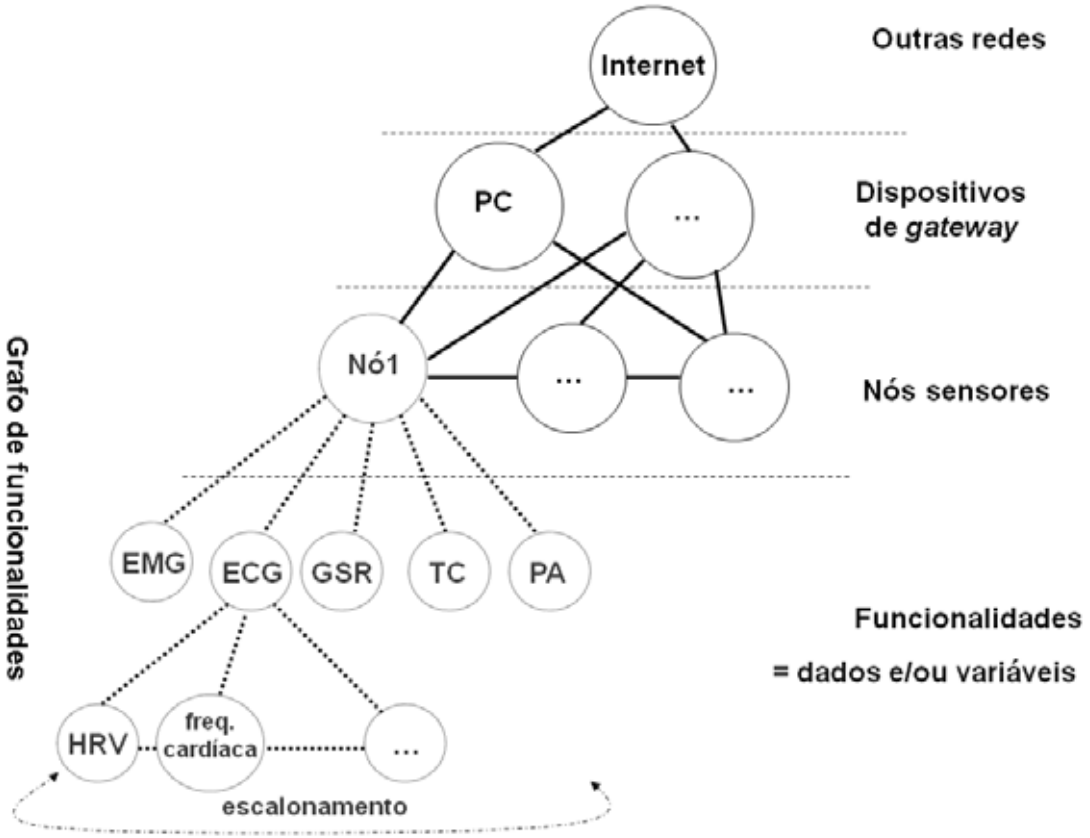


Figura 3.2 – Grafo de funcionalidades.

3.4 – CAMADA DE APLICAÇÃO: BWSNET CONFIGURATION TOOL

A Seção 1.3 apresentou a arquitetura SOAB do ponto de vista das atribuições de cada camada. Nesta seção são apresentados os requisitos funcionais dos sistemas que representam a camada de aplicação.

BWSNET Configuration Tool tem como objetivo oferecer um ambiente de configuração e programação da rede de sensores do corpo humano. A Figura 3.3 apresenta as funcionalidades do subsistema *BWSNET Configuration Tool*. Cumpre lembrar que o usuário apresentado nesta figura refere-se ao profissional da saúde.

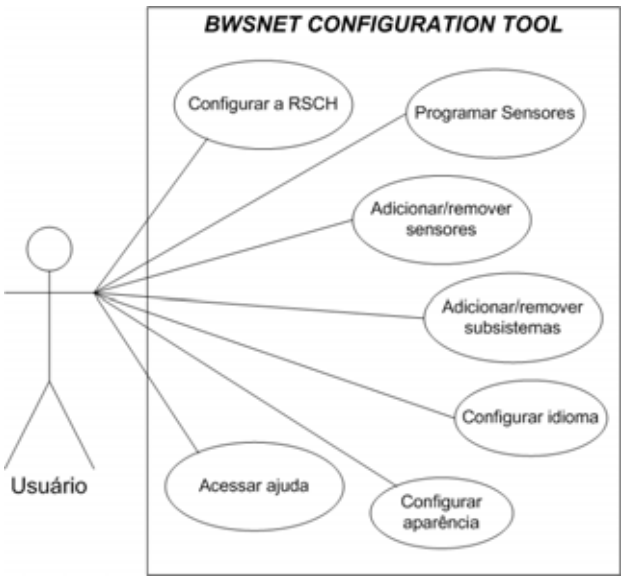


Figura 3.3 – Funcionalidades da camada1: *BWSNET Configuration Tool*

Configurar a rede de sensores do corpo humano (RSCH) significa definir e/ou modificar valores que serão associados a parâmetros que comandam o monitoramento de um paciente, sem que nenhuma parte do sistema seja reinicializada. Esses parâmetros definem uma aplicação e, ainda, devem ser alterados dinamicamente, isto é, durante a execução da própria aplicação. Para isso, um aplicativo específico foi projetado. A Figura 3.4 ilustra as funcionalidades da ferramenta para configuração da RSCH. A definição destas funcionalidades foi baseada no modelo de programação apresentado pelo MiLAN (CARAVALHO *et al*., 2004) e descrito na Seção 2.2.4.2. Durante a descrição de uma aplicação são definidos o grafo de variáveis baseadas em estados, o grafo de QoS dos

sensores e, ainda, selecionado o estado atual da aplicação. A atualização da RSCH pressupõe que essas informações devam ser repassadas ao núcleo do MiLAN (executado no *gateway*) para ajustes da RSCH em função da aplicação, a cada instante de tempo. Também, o grafo de funcionalidades (apresentado na Figura 3.2) requer mecanismos (ferramentas) para ajustes, em tempo de execução, das tarefas (funcionalidades) executadas pelos nós sensores. Por isso, o requisito "ajustar configurações dos sensores (tarefas)" é também uma funcionalidade requerida da ferramenta de configuração.

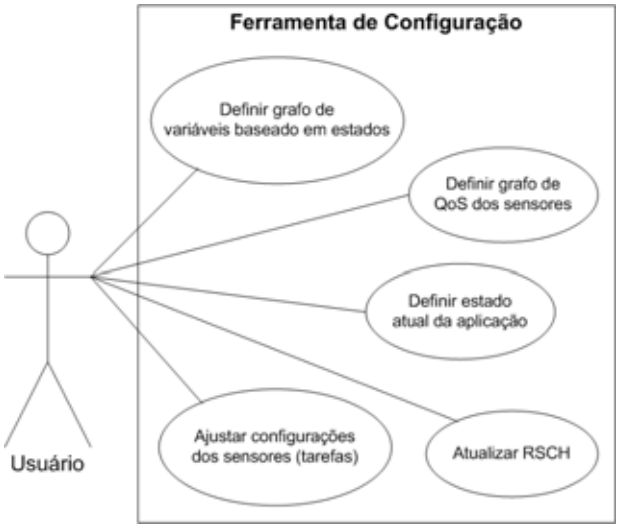


Figura 3.4 – Funcionalidades da camada1: ferramenta de configuração

Programar uma rede de sensores significa definir quais as funcionalidades, em forma de programas de computador, devem ser instaladas nos nós sensores. Como parte de *BWSNET Configuration Tool*, um aplicativo para programação dos sensores foi projetado. A Figura 3.5 ilustra as funcionalidades da ferramenta de programação dos sensores. A definição de quais programas devem ser instalados nos sensores parte da definição das tarefas (funcionalidades requeridas), seguida pela definição dos filtros digitais, seleção dos mecanismos para gerência de energia, seleção dos alarmes e outros. Após a definição desses mecanismos, o usuário pode avaliar também qual será o desempenho dos sensores antes de efetuar a programação. Para isso, um simulador foi projetado. A principal informação fornecida pelo simulador é o tempo estimado de funcionamento do sensor, em função da demanda por energia e por memória. Os cálculos são feitos com base no nível de energia (percentual de carga da bateria) estimado pelo próprio usuário. A Figura 3.6 ilustra as principais funcionalidades desse simulador. Ainda, quando o conjunto de componentes

eletrônicos (hardware) utilizado num sensor é modificado, o arquivo de configuração desse sensor precisa ser substituído. Essa funcionalidade foi projetada para ser executada por um técnico especializado (profissional da computação) ou pelo próprio sistema por meio de uma atualização remota (pela Internet).

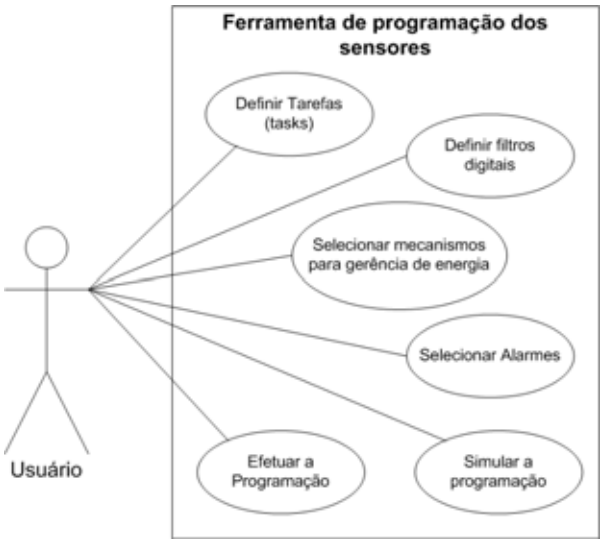


Figura 3.5 – Funcionalidades da camada 1: Ferramenta de programação dos sensores

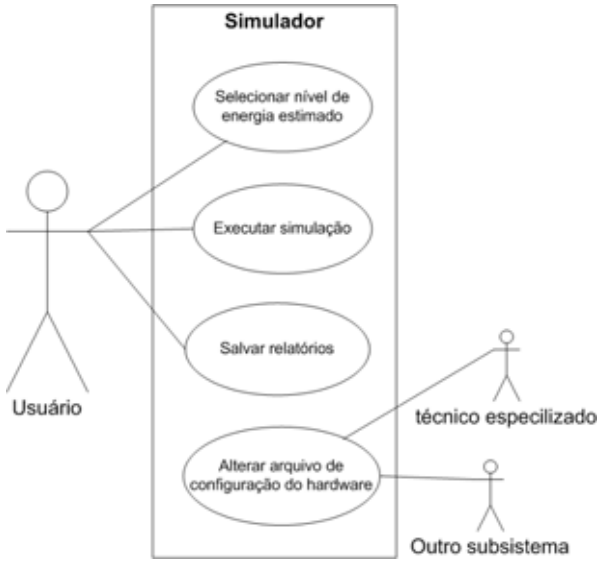


Figura 3.6 – Funcionalidades da camada 1: simulador da programação

Inicialmente, foram projetados dois tipos de sensores aptos a serem programados por meio de *BWSNET Configuration Tool*, o nó sensor de ECG e o nó sensor genérico. O nó sensor

de ECG é customizado (em termos de elementos gráficos de interface) para programação do eletrocardiograma, enquanto que, o nó sensor genérico pode ser programado para monitorar vários tipos de sinais. Por exemplo, a pressão arterial, o eletromiograma não-invasivo, a oximetria de pulso e outros.

Entretanto, podem haver situações em que o profissional da saúde deseje utilizar um novo sensor. Por exemplo, um sensor recém lançado no mercado. Pensando nisso, foi especificada a possibilidade de adicionar ou remover um novo sensor (veja Figura 3.3). Mas, para que esse novo sensor possa fazer uso de alguns dos mecanismos da arquitetura SOAB, precisa compatibilizar dados e funcionalidades em relação aos demais sensores propostos pela arquitetura. Um exemplo são os mecanismos para economia de energia. A Figura 3.7 apresenta o modelo de dados comum a todos os sensores compatíveis com a arquitetura SOAB. Este modelo de dados genérico foi proposto por Carvalho em (CARVALHO, 2005, p. 31). Neste projeto, este modelo de dados é utilizado como interface para compatibilização de atributos e operações executadas pelos sensores.

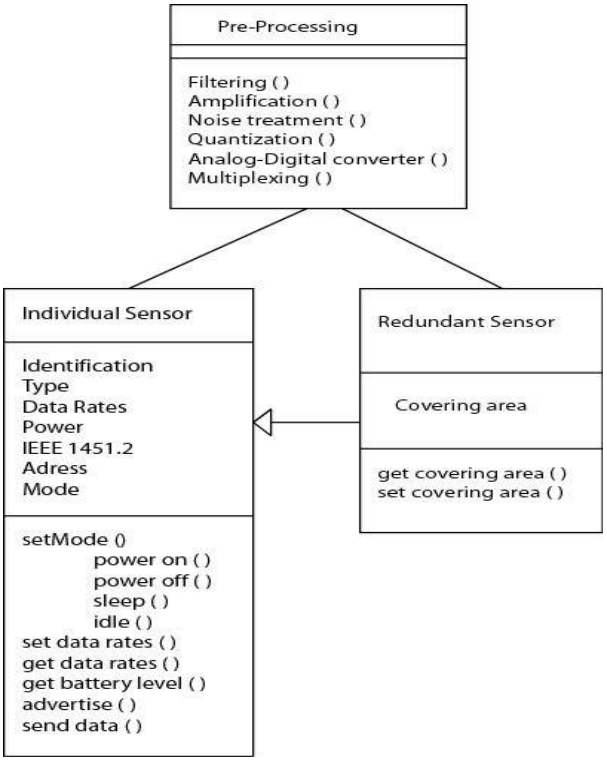


Figura 3.7 – Modelo de dados comum aos sensores compatíveis com a arquitetura SOAB (CARVALHO, 2005, p. 31).

A classe “Individual Sensor” tem atributos, tais como, identificação do sensor, tipo do sensor, taxas de transmissão de dados, dentre outros (Figura 3.7). Além disso, oferece operações que podem manipular o fluxo de informações e gerenciar o nível de bateria de cada sensor. A classe “Redundant Sensor” adiciona informações referentes à existência de redundância de sensores na classe “Individual Sensor”. Essas informações podem ser utilizadas para realocação de sensores e recursos em caso de falhas. A classe “Pre-Processing” inclui diferentes funções usadas para o processamento de sinais analógicos e digitais.

Um subsistema é um software adicional que o usuário pode necessitar durante o monitoramento de pacientes que estão utilizando redes de sensores. Por exemplo, um *plotter* para sinais eletrofisiológicos, uma interface para acesso a um mecanismo de buscas em bases de dados de informações médicas (por exemplo, o MedLine[13]), um sistema para tomada de decisões com base nos dados capturados pela rede e em consultas a outros sistemas de informação, um simulador ou outro programa qualquer. Pensando nisso, foi especificada a funcionalidade “adicionar/remover um subsistema” (veja Figura 3.3). Esse requisito visa expandir as funcionalidades da ferramenta, da camada de aplicação e, ainda, do sistema de forma geral, porque possibilita que um novo sistema possa ser incluído ou removido durante o uso de *BWSNET Configuration Tool*.

Em relação aos requisitos não-funcionais, todos os requisitos apresentados na Seção 2.2.2 foram considerados. Além desses, a portabilidade (Tabela 3.1) é uma característica importante para esse subsistema porque possibilita que o usuário substitua a plataforma de suporte (sistema operacional e hardware) quando achar conveniente. Essas características serão exploradas com mais detalhes no Capítulo 4. Nesse capítulo são apresentados detalhes dos protótipos e as tecnologias utilizadas para implementação.

3.5 – CAMADA DE INTERCONEXÃO: *MIDDLEWARE* BASEADO EM SERVIÇOS

A camada de interconexão tem como objetivo fornecer um meio pelo qual os serviços (funcionalidades) ofertados pela rede de sensores do corpo humano sejam acessados remotamente, isto é, à distância pela Internet. Deve promover a independência entre as

[13] http://www.ncbi.nlm.nih.gov/sites/entrez

camadas. Esta é uma característica importante porque facilita substituição e/ou a adição de novos serviços sem a necessidade de modificações nas demais camadas que interagem com esta. Por exemplo, uma nova interface de programação (orientada a outros perfis de usuários) pode ser projetada, implementada e utilizada sem que os sistemas das camadas subjacentes necessitem de modificações. Este tipo de facilidade também é chamado de reusabilidade (anglicismo do termo *reusability*, veja Tabela 3.1).

Pensando nisso, foi projetada uma camada de software para interconexão dos dispositivos de *gateway* com a Internet. Basicamente, esse sistema (chamado de *middleware* baseado em serviços) possibilita que a rede de sensores do corpo humano seja vista como uma entidade independente e fornecedora de serviços para a camada de aplicação. Na prática, o *middleware* encapsula diferentes protocolos de comunicação e possibilita que o acesso remoto aos serviços da rede de sensores seja simplificado e dinâmico. A dinamicidade está relacionada à capacidade de descoberta dos serviços durante a execução das aplicações (requisito funcional apresentado na Seção 2.3.3.2).

Em relação aos requisitos não-funcionais, o *middleware* deve se basear em padrões da Internet e não em soluções proprietárias. Ainda, resguardar a independência quanto à plataforma de suporte (Tabela 3.1). Isso facilita a interoperabilidade com outros sistemas de tecnologias diferentes e, principalmente, a extensibilidade, capacidade do *middleware* em acomodar novos serviços. Por exemplo, a camada de aplicação pode requerer acesso a múltiplas redes de sensores do corpo humano que podem ter sido implementadas por outros desenvolvedores com tecnologias diferentes.

Assim, o projeto de um *middleware* foi desenvolvido de acordo com padrões especificados pelo Web Services W3C (W3C, 2006) e pela a arquitetura SOA (*Service Oriented Architecture*) (WSA, 2004). Os padrões utilizados para implementação são descritos no Capítulo 4 juntamente com os protótipos desenvolvidos.

A arquitetura SOA define três principais entidades: *(i)* consumidor do serviço (em inglês, *service consumer*); *(ii)* provedor do serviço (em inglês, *service provider*) e *(iii)* um gerente do serviço (em inglês, *service broker*). A Figura 3.8 ilustra a arquitetura SOA.

O provedor do serviço é responsável por criar uma descrição do serviço, publicar essa descrição em um ou mais gerentes do serviço e tratar as invocações de serviço recebidas por um ou mais consumidores de serviço. Há ainda três operações: *(i)* publicação do serviço (em inglês, *publish*); *(ii)* descoberta do serviço (em inglês, *find*) e *(iii)* utilização do serviço (em inglês, *bind*).

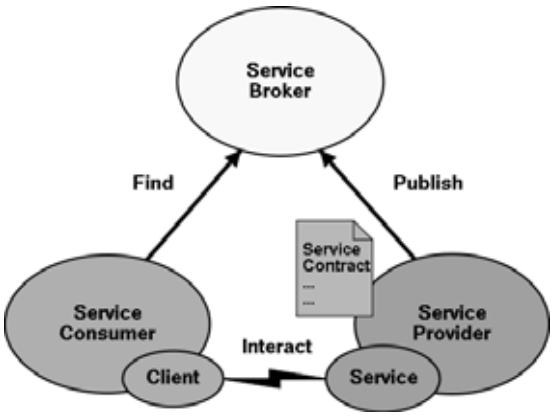


Figura 3.8 – *Service Oriented Architecture* (WSA, 2004).

A operação *"publish"* é responsável pelo registro ou anúncio do serviço. A operação *"find"* oferece ao interessado no serviço a possibilidade de descoberta e entendimento acerca de como operar o serviço. A operação *"bind"* é responsável por promover a integração entre o cliente (consumidor do serviço) e o servidor (provedor do serviço) (GRAHAM, 2002). Ao final, as interações entre o consumidor e o produtor podem acontecer de forma síncrona, por meio de chamadas remotas a procedimentos (em inglês, RPC – *Remote Procedure Call*), ou, ainda, de forma assíncrona pelo envio e recebimento de mensagens.

Na arquitetura SOAB os serviços oferecidos para a camada de aplicação são descritos por meio de documentos específicos para descrição de serviços, que podem ser publicados em servidores de gerência de serviço (*Service Brokers*). Nesses documentos são apresentadas as funcionalidades oferecidas, a localização (endereço) dos serviços, as interfaces de acesso e outras informações. Esses documentos seguem um padrão da Internet. Um exemplo é apresentado no Capítulo 4. De posse dessas informações a camada de aplicação (cliente) deve ser capaz de executar uma ordem de serviço, por exemplo, uma chamada a um procedimento remoto executado pela entidade que implementa o serviço. No caso de

SOAB, os serviços são implementados pelo BWSNET *Proxy* (*Service Provider*). Dependendo do que é requisitado, esse sistema pode acionar uma operação que será executada nos sensores, por intermédio do sistema operacional MedOS (apresentado na Seção 3.7). Ou, ainda, iniciar um processo local para responder a essa requisição. A Figura 3.9 ilustra como SOAB adere ao padrão SOA por meio das operações e artefatos disponibilizados pelo *middleware* baseado em serviços.

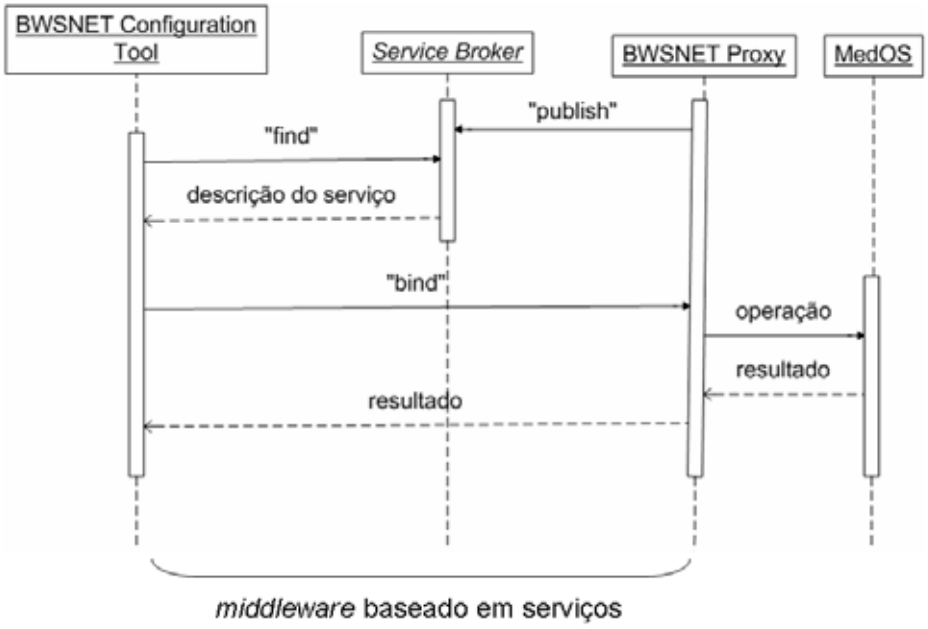


Figura 3.9 – Operações da SOAB segundo padrão SOA.

3.6 – SERVIÇOS DA RSCH: BWSNET *PROXY*

Os serviços referentes a uma rede de sensores do corpo humano são implementados nos dispositivos de *gateway* (veja Figura 3.1). Nesses dispositivos é executado um sistema chamado BWSNET *Proxy*. O BWSNET *Proxy* corresponde à terceira camada da arquitetura SOAB. A Figura 3.10 exibe o BWSNET *Proxy* e o principal conjunto de componentes desse sistema.

Atualmente, os serviços oferecidos pelo BWSNET *Proxy* são representados por dois componentes: o tradutor MedOS e o gerador de agente. Esses serviços possibilitam a configuração e a programação da RSCH e dos sensores.

O tradutor MedOS é responsável por analisar as mensagens encaminhadas pelo *middleware* relativas à configuração das aplicações e da rede de sensores, criar uma mensagem equivalente e compreensível para a camada subjacente e acionar o componente *Wrapper* para que as mensagens de configuração sejam enviadas a cada nó sensor. O tradutor MedOS pode repassar diretamente os comandos do usuário para configuração dos sensores. Ainda, utilizar o componente MiLAN para determinar quais ajustes são necessários a cada momento. De posse dessas decisões o tradutor MedOS deve então comandar o *Wrapper* para envio dos comandos aos sensores.

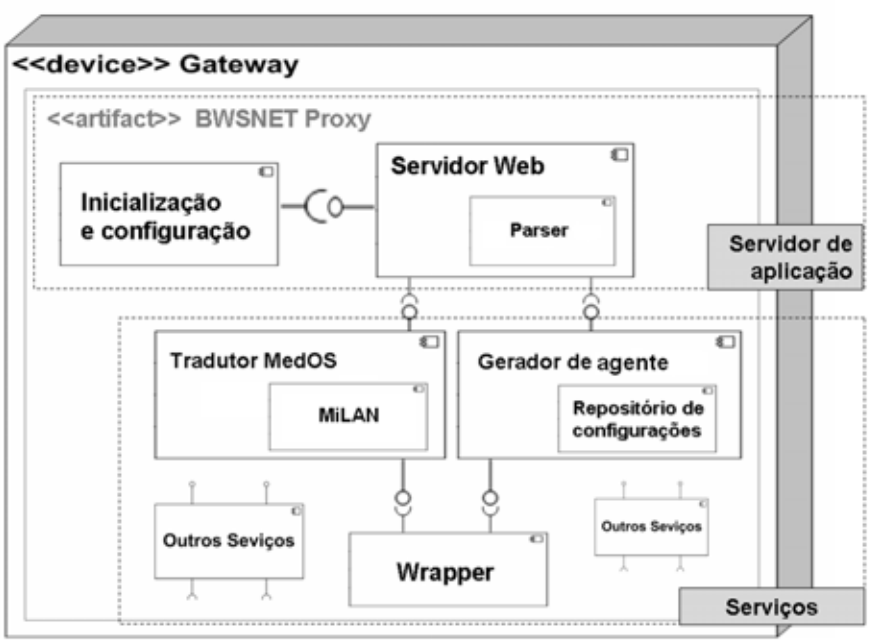


Figura 3.10 – Principais componentes do BWSNET *Proxy*.

O gerador de agente recebe uma seleção de funcionalidades (arquivo com a programação efetuada pelo usuário) por meio de uma mensagem. A partir daí, determina o comportamento inicial para cada nó sensor e comanda o *Wrapper* para a compilação, a instalação e a inicialização do software em cada nó sensor. O gerador de agente deve escolher dentro do repositório de configurações (repositório de agentes) qual a melhor programação (agente) para cada sensor, com base nas funcionalidades selecionadas pelo usuário. Esse processo corresponde à programação estática do nó sensor. Em inglês, também é conhecido como *deployment-time programmability*.

Outros serviços podem ser incorporados ao sistema como novos componentes do BWSNET *Proxy*. Por exemplo, a integração da RSCH com sistemas de informação de saúde do paciente (em inglês, *Healthcare Information System – HIS*). Da mesma forma, as funcionalidades descritas na Seção 2.2.3. Para isso, uma estrutura comum chamada servidor de aplicação [14] foi desenvolvida. O objetivo dessa estrutura é fornecer um ambiente de execução para serviços que consuma pouca memória e seja portável (independente de hardware e software). Portanto, que possa ser executado em diferentes tipos de dispositivos de *gateway* e que seja capaz de receber e tratar requisições no formato padronizado pelo *middleware*.

3.6.1 – Representação dos agentes baseada em autômatos

Neste trabalho, o comportamento inicial de cada nó sensor é determinado pelo agente escolhido pelo gerador de agente para comandar o hardware. Esse agente (software que será instalado nos sensores) é descrito por um autômato. Um conjunto de agentes forma uma família de agentes. As famílias de agentes são armazenadas no repositório de configurações (repositório de agentes).

Autômato ou máquina de estados finitos é um modelo matemático que pode ser utilizado para representar o comportamento de um sistema computacional. Basicamente, é composto por estados, que podem ser representados graficamente por círculos, e arestas que indicam as ações responsáveis pelas mudanças de estados. Para a representação gráfica de um autômato utiliza-se um diagrama de estados.

Formalmente, um autômato é representado pela quíntupla $<Q, \sum, \delta, q0, F>$, em que: "*Q"* corresponde a um conjunto finito de estados e "∑" a um alfabeto, isto é, um conjunto finito de símbolos de entrada. A letra "δ" é chamada função de transição. Recebe como argumento um estado e um símbolo de entrada e retorna um outro estado (no caso do autômato determinístico) ou um conjunto de estados (no caso do autômato não-determinístico). O elemento *"q0"* corresponde ao estado inicial, enquanto que, o elemento *"F"* corresponde ao conjunto de estados finais. Portanto, *"F"* é subconjunto de "*Q".* No

[14] Um servidor de aplicação ou em inglês, *application server*, é um *software* que disponibiliza um ambiente para a instalação e execução de certas aplicações. Os servidores de aplicação também são conhecidos como *software* de *middleware*.

diagrama de estados, usualmente, *"q0"* é indicado pelo símbolo "→", enquanto que, *"F"* é marcado por um círculo duplo. Como exemplo, a Figura 3.11 apresenta um autômato determinístico relacionado ao consumo de corrente do microcontrolador MSP430F149, apresentado na Seção 2.3.

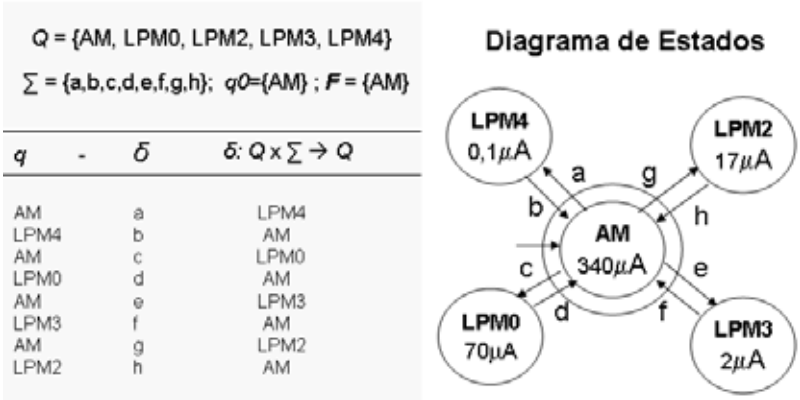


Figura 3.11 – Um autômato finito e seu diagrama de estados.

Em linhas gerais, esta máquina de estados (Figura 3.11) é também conhecida como PSM (em inglês, *Power State Machine*) (BENINI *et al*., 2000). Mudanças de estado têm custo relacionado ao consumo de energia e ao tempo gasto para o chaveamento do modo de operação. No caso do MSP430 é próximo de 6 μs. Mudanças de estado são disparadas por uma política pré-definida para gerenciamento de energia. PSM é uma representação muito utilizada para descrever mecanismos que promovem o gerenciamento dinâmico do consumo de energia. Um exemplo é o DPM apresentado na Seção 2.3.

Em (CARVALHO *et al*., 2003b) é apresentada uma extensão para o PSM. EPSM (*Extended Power State Machine*) é um modelo que inclui as necessidades das aplicações a cada instante de tempo (em inglês, *the state of computation*) e a influência de eventos externos na formulação da PSM. Para isso, inclui o elemento QoS (*Quality of Service*) na quíntupla que define a máquina de estados. A Equação 3.1 apresenta a EPSM.

$$M = <Q,E, q0, \delta, QoS> \quad (3.1)$$

"Q" corresponde ao conjunto de estados. Associado a cada elemento desse conjunto existe um elemento do conjunto QoS. QoS é relativo ao grau de qualidade que é associado ao

volume de informação que está sendo transmitido ou recebido a cada momento. *"E"* corresponde ao alfabeto de eventos, que pode conter eventos relacionados à bateria ou a eventos externos ao sistema. Eventos são responsáveis pelas mudanças das necessidades das aplicações. O elemento *"q0"* corresponde ao estado inicial, enquanto que, "δ" corresponde ao conjunto de transições de estado. Cada transição de estado é obtida por uma relação que associa eventos, dois a dois, a um determinado estado. Com isso, é possível determinar se haverá ou não uma mudança de estado e, caso haja, qual será o novo estado.

Para uma aplicação em RSCH, a proposição lançada com a EPSM diz que é possível economizar energia mudando dinamicamente o monitoramento do estado de saúde do paciente. Para isso, é necessário reajustar os parâmetros que comandam as interfaces de comunicação com objetivo de garantir a QoS para a aplicação. Até mesmo, desligando essas interfaces quando não estiverem em uso (CARVALHO *et al*., 2003b).

A Figura 3.12 mostra um diagrama de estados referente a um monitor de ECG. Essa aplicação tem como objetivo reduzir o consumo de energia resguardando as necessidades da aplicação a cada instante de tempo. As necessidades da aplicação são associadas às condições avaliadas do paciente. Tais condições são responsáveis por descrever um conjunto de eventos que influirá nas mudanças de estados. Essas condições são reajustadas dinamicamente por um agente externo (software) que analisa amostras dos dados capturados. A Tabela 3.2 apresenta o ponto de partida. Relaciona a qualidade do monitoramento com a condição do paciente avaliada inicialmente pelo especialista.

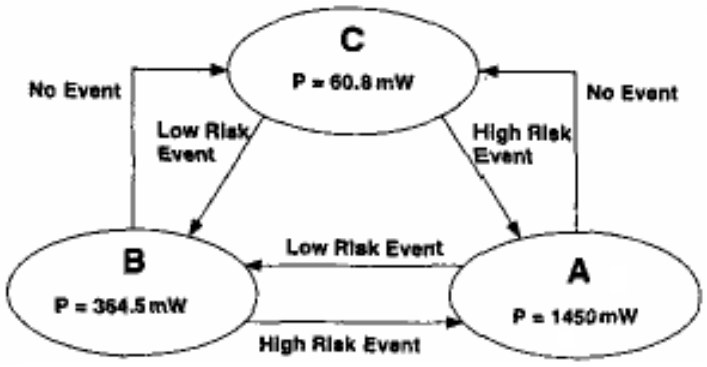


Figura 3.12 – EPSM para um monitor de eletrocardiograma
(CARVALHO *et al*., 2003b).

O conjunto de estados “Q” é composto por três estados: A, B e C. O estado “C” corresponde ao monitoramento mínimo, isto é, ao ECG num único canal, com taxa amostragem de 300 Hz. O estado “B” refere-se ao monitoramento parcial, ou seja, ECG em três canais e com amostragem de 300 Hz. O estado “A” equivale ao monitoramento com o máximo de qualidade dos sinais, isto é, doze canais de ECG com amostragem de 1000 Hz.

Tabela 3.2 – Qualidade do monitoramento de acordo com o estado clínico do paciente (CARVALHO *et al*., 2003b).

Qualidade do Monitoramento	**Min (%)**	**Parcial (%)**	**Max (%)**	**Consumo da interface de comunicação (mW)**
Paciente saudável	97	2.9	0.1	71.00
Paciente com uma doença de baixo risco (*Low Risk*)	95	4.75	0.25	78.70
Paciente com uma doença de médio risco (*Middle Risk*)	40	59.5	0.5	248.45
Paciente com uma doença de alto risco (*High Risk*)	30	65	5	327.67

Existe ainda o conjunto “QoS”, que, neste caso, determina o intervalo de tempo em que a interface de comunicação precisa estar ativada para cada estado. Durante o intervalo de inatividade as informações são armazenadas num *buffer* local. Para o estado “C” tem-se uma retenção de 10 segundos, para o estado “B” de 5 segundos e para o estado “A”de 1 segundo.

Outra implementação, que tem como base a Tabela 3.2, é mostrada pela Figura 3.13. Esta figura exibe o diagrama de estados do autômato responsável pelo comando de um nó sensor de ECG multifuncional. Aos estados estão associados valores para o consumo de corrente obtidos com base em manuais técnicos de componentes eletrônicos.

Como benefícios, a utilização de autômatos para modelagem de agentes para uso em redes de sensores produz uma especificação fácil de entender. Também, os modelos existentes para multiprogramação dos nós sensores: orientado a eventos e baseado em *threads* (apresentados na Seção 2.2.3.5) oferecem noção intuitiva de estados, comumente associados às tarefas. Isso reduz a distância entre a implementação e o modelo. Por exemplo, os estados ECG1, ECG3 e ECG6 (veja Figura 3.13) são mapeados em tarefas. Ainda, os autômatos consomem menos memória se comparados a outras metodologias existentes, como, por exemplo, o OSM (em inglês, *Object State Model*) (Karsten & Römer,

2005). OSM requerer uma linguagem para especificação diferente da utilizada para implementação.

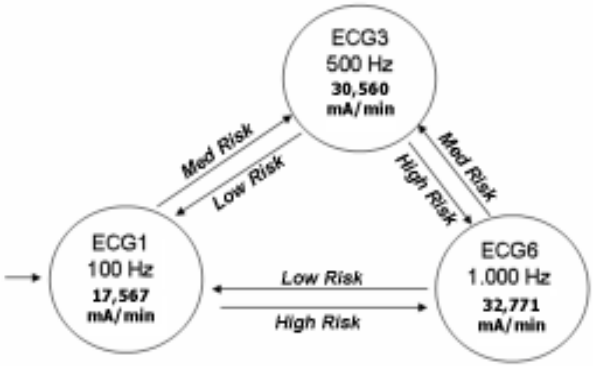


Figura 3.13 – Uma implementação do EPSM para um nó sensor de ECG multifuncional (BARBOSA *et al*., 2006).

3.6.2 – O gerador de agente

Atualmente, os modelos de programação desenvolvidos para as redes de sensores têm como objetivo possibilitar que os programadores desses sistemas possam desempenhar suas atividades utilizando ferramentas de programação com alto-nível de abstração. Portanto, sem se preocupar com muitos detalhes técnicos acerca dos nós sensores (hardware). Essa transparência, em geral, requer a utilização de compiladores de nível intermediário. Esses são desenvolvidos e inseridos entre a interface de programação (abstração de alto nível) e a tecnologia utilizada para programação dos nós sensores.

Como exemplos, o BeanWatcher (LINS *et al*., 2003) é uma ferramenta que possibilita a geração de programas para redes de sensores com base na descrição de componentes por meio de uma ferramenta gráfica. *Abstract Regions* (WELSH & MAINLAND, 2004) oferece primitivas de programação que escondem detalhes de baixo-nível (do hardware), dos sistemas comunicação, do endereçamento utilizado pela rede e do compartilhamento de dados. Em seguida, aplica políticas para economia de energia. VM* (KOSHY & PANDEY, 2005) é um *framework* [15], desenvolvido para geração automática de máquinas virtuais utilizadas como ambiente de execução de programas, dentro dos nós sensores.

[15] No desenvolvimento do software, um *framework* é uma estrutura de suporte definida em que um outro projeto de software pode ser organizado e desenvolvido. Tipicamente, um *framework* pode incluir programas de apoio, bibliotecas de código, linguagens de *script* e outros *software*.

Cada instância da máquina virtual VM* deve ser capaz de interpretar uma aplicação escrita na sintaxe Java. No ATaG (PATHAK & PRASANA, 2006) a programação é definida por meio de dois grafos: *(i) abstract task graph* (define a aplicação) e *(ii) network graph* (define a topologia de rede). Baseado nesses grafos, o compilador gera programas numa linguagem de programação específica que, posteriormente, é utilizada para a geração dos sistemas (programas executáveis).

Entretanto, todos os exemplos supracitados consideram que os programadores detenham algum conhecimento em Sistemas de Computação. Por isso, não avaliam os requisitos específicos do domínio das aplicações para as quais essas redes são utilizadas. Portanto, entende-se que promovam pouco ganho em eficiência. De fato, apenas aumentam o nível de transparência oferecido para o programador em relação ao hardware.

A Figura 3.14 apresenta um modelo de programação para redes de sensores do corpo humano centrado num compilador intermediário, chamado gerador de agente. Esse modelo é composto por cinco principais atividades. A seleção das funcionalidades refere-se à descrição de mecanismos e políticas necessárias. Cumpre lembrar que na arquitetura SOAB essa função é atribuída à camada 1 (BWSNET *Configuration Tool*). A atividade seguinte refere-se ao envio da programação para o gerador de agente. Isso deve ser executado pela camada 2 da SOAB (*middleware* baseado em serviços). Após a geração do programa para a plataforma alvo, é executada a compilação e a instalação do software no nó sensor. Por fim, é executada a inicialização do sistema. O gerador de agente tem como objetivo determinar a melhor configuração para o funcionamento de um nó sensor e, então, gerar os programas que devem ser compilados para o hardware que está sendo utilizado.

O gerador de agente é composto por três módulos: *parser*, seletor e gerador. O módulo *parser* é responsável por extrair informações de relevância para o módulo seletor. Essas informações são relativas a parâmetros, como, por exemplo, o objetivo da aplicação, a quantidade de tarefas, os tipos de tarefas, os alarmes selecionados, entre outros. Os valores lidos desses parâmetros são utilizados para preenchimento de uma tabela de valores, utilizada pelo módulo seletor.

Com base na tabela de valores (que representa os requisitos definidos pelo usuário), o módulo seletor deve escolher o agente que maximiza o potencial de cada aplicação. Neste

trabalho, maximizar o potencial de uma aplicação significa aumentar o tempo de funcionamento do sistema pela economia de energia elétrica. A economia de energia elétrica deve considerar as funcionalidades exigidas pelo usuário (aplicação), as capacidades do hardware e, ainda, os requisitos mínimos para cada funcionalidade (definidos com base no conhecimento especialista). Essas políticas são expressas por meio de autômatos que representam os agentes. O módulo seletor implementa a estrutura responsável pela escolha do melhor agente, dentro de um conjunto de possíveis agentes, chamado família de agentes. Na prática, esse módulo foi projetado para ser implementado por uma árvore de decisão [16].

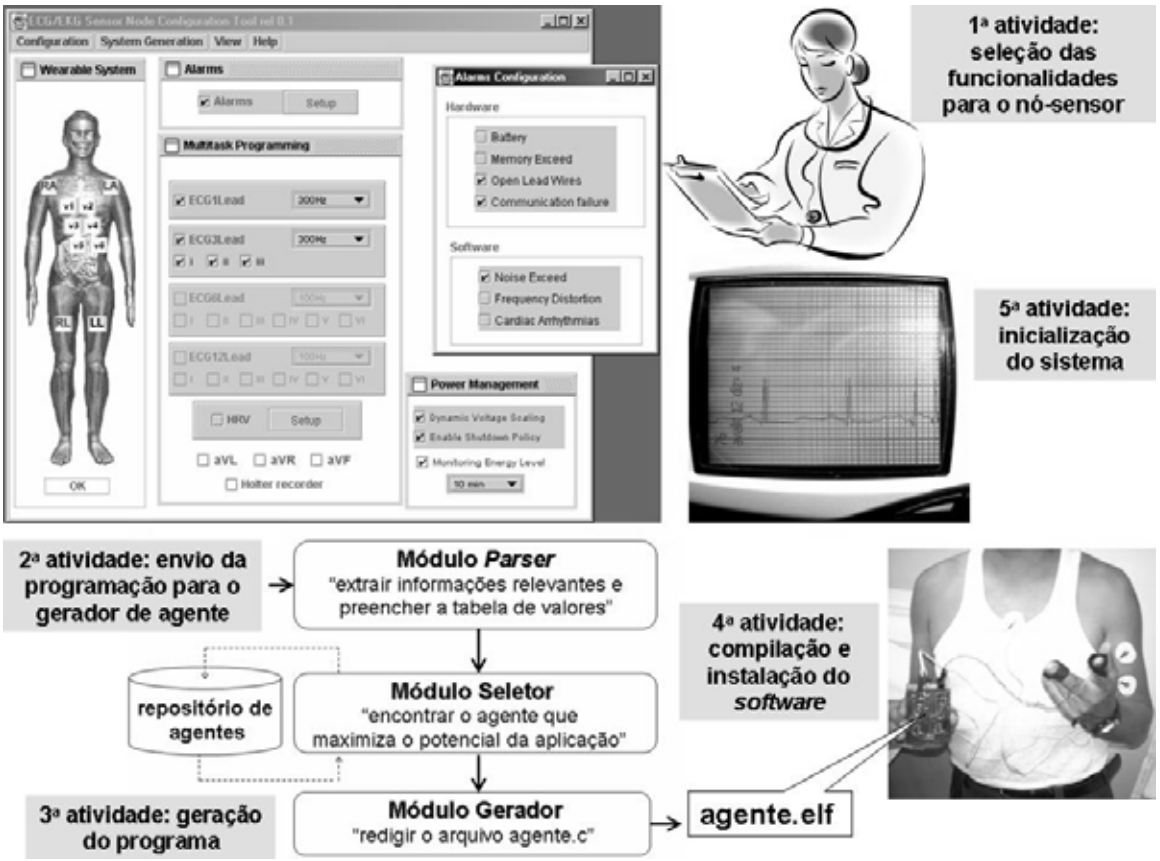


Figura 3.14 - O gerador de agente (Barbosa *et al*, 2007).

[16] Uma árvore de decisão é uma estrutura de dados, representada por um grafo acíclico e dirigido, que pode ser utilizada para sistematizar um processo de escolha. Numa árvore de decisão os vértices internos representam ações, as arestas representam os resultados de uma ação e as folhas representam os resultados finais (GERSTING, 2004). Uma árvore de decisão pode ser gerada automaticamente com base numa tabela de valores ou definida com base no conhecimento especialista.

Como exemplos, a Figura 3.15a exibe uma família de agentes para o nó sensor de ECG, enquanto que, a Figura 3.15b mostra um agente escolhido pelo módulo seletor com base na seleção das funcionalidades (descrição da aplicação) efetuada pelo profissional da saúde. Para o agente escolhido devem ser associadas políticas para o gerenciamento dinâmico do consumo de energia, como, por exemplo, a redução ou aumento da freqüência de amostragem do sinal e o desligamento de partes do nó sensor, como o rádio transmissor. Cumpre mencionar que essas políticas devem ser definidas com base no conhecimento especialista.

Por fim, o módulo Gerador pode redigir o programa, por exemplo, um arquivo em linguagem C que representa o agente escolhido. Para isso, uma descrição que contenha referência às bibliotecas de código necessárias deve ser utilizada. Essa descrição é armazenada no repositório de agentes, juntamente com a descrição de cada agente. No momento da seleção do autômato, essas informações são utilizadas pelo módulo Gerador para a construção do arquivo "agente.c" (veja Figura 3.14). A partir daí, ferramentas específicas para o hardware, como, compiladores, depuradores e instaladores, são executadas para a programação efetiva do nó sensor.

A solução adotada para o módulo Seletor almeja aumentar a eficiência do processo de programação, minimizando o tempo gasto e resguardando a eficácia. A definição de cada agente é muito influenciada pelo conhecimento especialista, portanto, um mecanismo que execute a construção automática de agentes poderá tornar lenta e imprecisa a programação dos nós sensores. Isso, em razão da quantidade de informações que devam ser consideradas e porque os agentes gerados não são avaliados previamente por um especialista. Entretanto, as políticas utilizadas para seleção dos agentes podem ser substituídas sem a necessidade de modificações no módulo seletor. Isso promove a manutenibilidade do modelo proposto (veja Tabela 3.1).

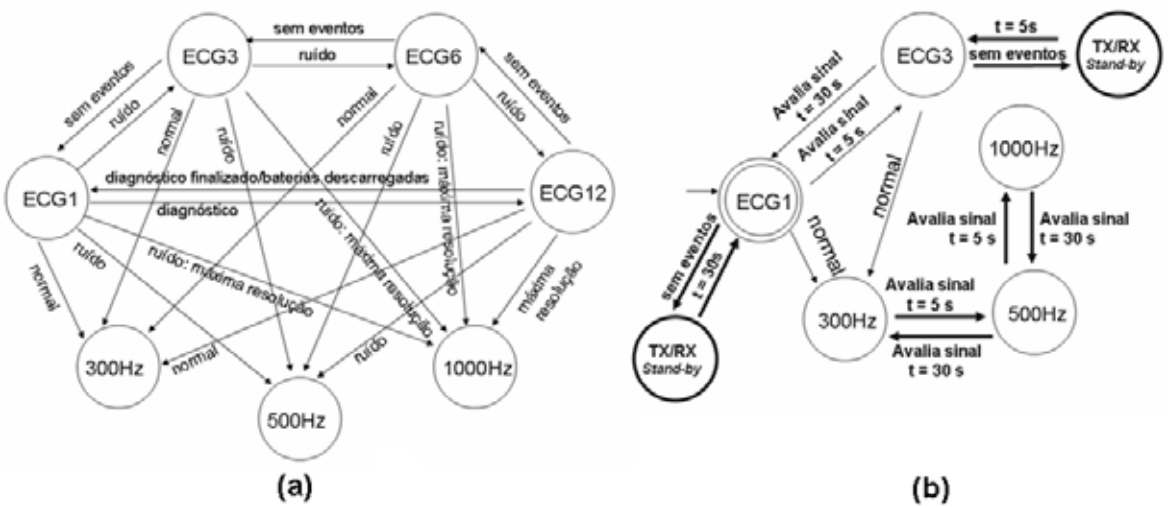


Figura 3.15. (a) Uma família de agentes para o nó sensor de ECG; (b) Exemplo de um agente escolhido pelo gerador de agente (Barbosa *et al*, 2007).

3.6.3 – O tradutor MedOS

O tratamento das mensagens do *middleware* dentro dos nós sensores requer capacidade de processamento para grandes cadeias de caracteres. Além disso, de capacidade de armazenamento considerável.

O componente tradutor MedOS (apresentado na Figura 3.10) analisa as mensagens enviadas pelo *middleware* relativas à configuração do sistema, cria uma mensagem equivalente e compreensível para a camada subjacente e aciona o componente *Wrapper* para que mensagens de configuração sejam enviadas a cada nó sensor.

Para isso, o Tradutor MedOS executa um analisador léxico sobre as mensagens do *middleware*. Isso tem como objetivo obter informações que possam ser mapeadas em comandos. Esses comandos são relativos a um conjunto de comandos (chamado protocolo MedOS) que cada nó sensor pode ser capaz de interpretar. Esses comandos possibilitam modificações dos agentes que foram instalados nos sensores, durante a programação dos mesmos. Cada agente suporta um conjunto de comandos.

Para uma determinada ação podem ser requeridos vários comandos. Esses podem ser encaminhados diretamente pelos usuários ou definidos e encaminhados com base nos

ajustes necessários para cada aplicação, a cada instante de tempo. Esses ajustes foram projetados para serem executados pelo componente MiLAN (veja Figura 3.10).

O componente MiLAN corresponde ao núcleo do MiLAN, apresentado na Seção 2.2.4.2. Com base em grafos com informações de configuração (recebidos dos usuários), esse componente determina quais ajustes nas configurações dos sensores são necessários a cada momento. Tem objetivo de manter a qualidade mínima de funcionamento da RSCH para cada estado da aplicação, a cada instante de tempo. Para isso, o MiLAN descobre as capacidades e a disponibilidade dos sensores para suporte a uma determinada aplicação. Em seguida, executa os ajustes necessários. A Figura 3.16 ilustra operações executadas pelo componente MiLAN. Essas operações foram descritas na Seção 2.2.4.2.

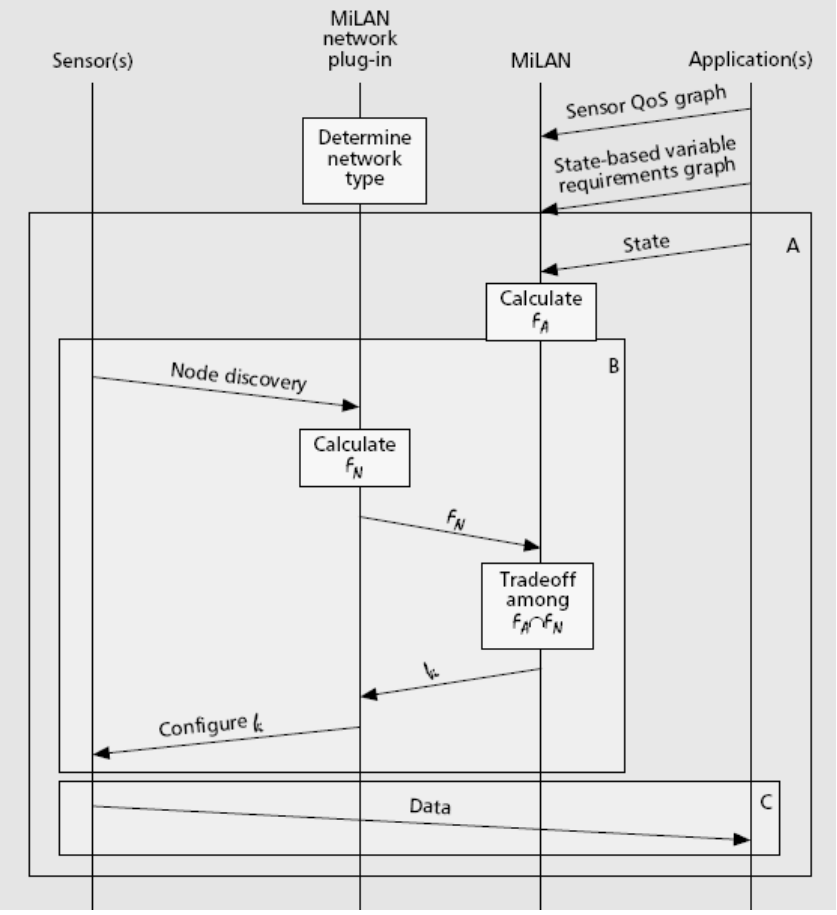


Figura 3.16 – Operações do MiLAN (Carvalho *et al*, 2004).

3.7 – SERVIÇOS DO HARDWARE: MEDOS

O MedOS é um sistema operacional projetado para executar nos nós sensores do projeto *Body-worn Sensor Networks* (BWSNET). O MedOS tem como objetivo fornecer mecanismos (sob forma de programas) para que os agentes (como o exemplo da Figura 3.15b) possam ser implementados e executados pelos nós sensores. Na prática, o MedOS é um conjunto de bibliotecas de código, especificadas e construídas com base num núcleo (*kernel*) mínimo, que oferece suporte a preempção de tarefas. A Figura 3.17 ilustra as principais funcionalidades dos MedOS.

Suportar a execução de agentes implica na existência de mecanismos para ajustes nas tarefas executadas pelos nós sensores. Também, na capacidade de interpretar comandos enviados pelo BWSNET *Proxy* e na existência de mecanismos e políticas para acesso ao hardware (*device drivers*). Os *device drivers* devem oferecer interfaces para programação dos sensores, da interface de comunicação e do microcontrolador.

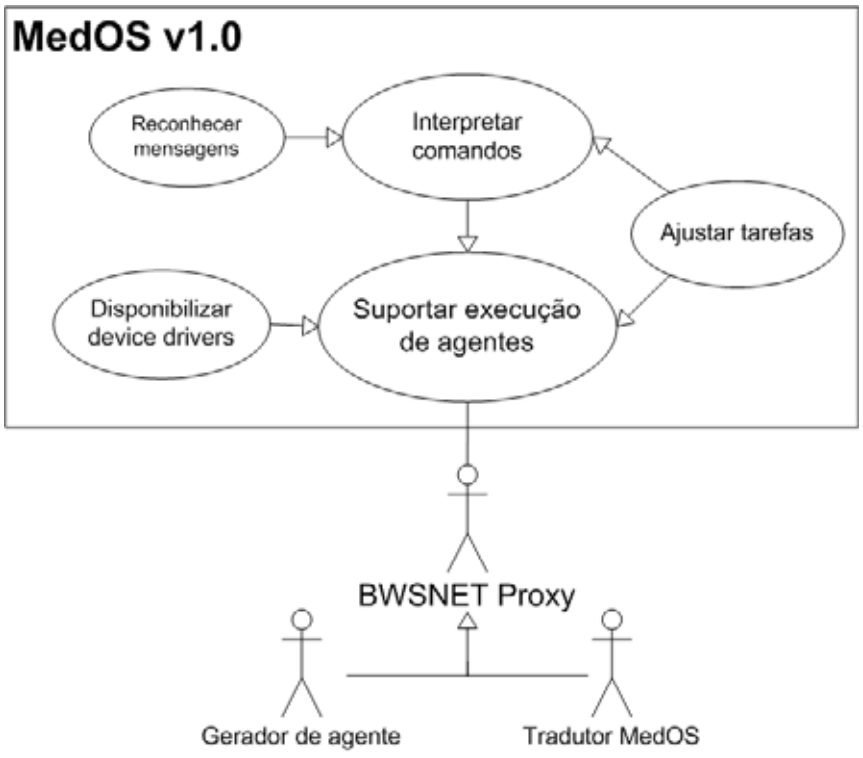


Figura 3.17 – Principais funcionalidades do MedOS.

As funcionalidades desempenhadas por cada nó sensor são representadas por estados. Para cada estado é usualmente criada uma tarefa no MedOS. A cada tarefa é associada um valor de prioridade e um estado de execução. Essas informações são utilizadas para estabelecer a ordem e a temporalidade determinada pelo autômato que descreve o funcionamento do agente instalado em cada sensor (ajustar tarefas). A manipulação dos valores de prioridades e dos estados de execução das tarefas possibilita cumprir a meta pré-estabelecida pelo autômato ou, ainda, modificar o comportamento do agente durante a execução do mesmo.

Interpretar comandos é uma funcionalidade que possibilita ajuste durante a execução do agente. É implementada pelo Interpretador de comandos, a tarefa de maior prioridade gerenciada pelo MedOS. Pode ou não ser incluído durante a programação do nó sensor. A cada comando enviado pelo BWSNET *Proxy*, o interpretador de comandos deve reconhecer a validade desse comando dentro do conjunto de comandos válidos (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.

3.7.1 – O FreeRTOS

O MedOS foi projetado para utilizar um núcleo (*kernel*) com suporte a preempção de tarefas. Esse núcleo deve oferecer interfaces que facilitem a inclusão de novas funcionalidades e, ainda, que ofereçam suporte a diferentes plataformas de hardware. Além desses requisitos, a aquisição de licenças para uso de software não é desejável. Assim, uma solução gratuita foi necessária no momento das primeiras especificações do MedOS. Após estudo de algumas possibilidades, o FreeRTOS[17] foi escolhido como núcleo do MedOS.

O FreeRTOS é um projeto que vem sendo desenvolvido com licença GNU *General Public License* (GPL). Tem como objetivo disponibilizar um conjunto de bibliotecas de código para suporte à multiprogramação em microcontroladores. Atualmente, o FreeRTOS encontra-se na versão 4.7. Pode ser utilizado em diferentes microcontroladores, entre os quais estão: o MSP430, o PIC18F, o ARM7, o ATmega128.

[17] http://www.freertos.org/.

A multiprogramação com o FreeRTOS pode ser alcançada de duas maneiras: *(i)* pela utilização de tarefas (*tasks*), estruturas independentes com suporte à preempção e a utilização de valores de prioridades. Cada *task* tem o seu próprio contexto: endereços de memória e pilha de execução; *(ii)* pela utilização de *co-routines*, estruturas que compartilham um único contexto para execução. *Co-routines* podem co-existir com as *tasks*, mas, são sempre preteridas quando há *tasks* aguardando para serem executadas.

O FreeRTOS oferece duas políticas para escalonamento de *tasks*. A primeira é uma política preemptiva. Com ela, a *task* de maior prioridade sempre é executada em detrimento às demais. *Tasks* de mesma prioridade compartilham o tempo do processador seguindo uma fila circular. Isto é, cada *task* recebe um intervalo fixo de tempo para execução. Ao final desse intervalo de tempo é substituída pela próxima *task* da fila. A outra política é a cooperativa. Nesse caso, a troca do contexto somente irá ocorrer se a *task* em execução estiver bloqueada ou executar a chamada ao sistema "taskYIELD()". *Co-routines* são sempre escalonadas pela política cooperativa em relação às demais *co-routines*. A Figura 3.18 exibe o diagrama de estados para *tasks, co-routines* e as respectivas chamadas de sistema responsáveis pelas mudanças de estado.

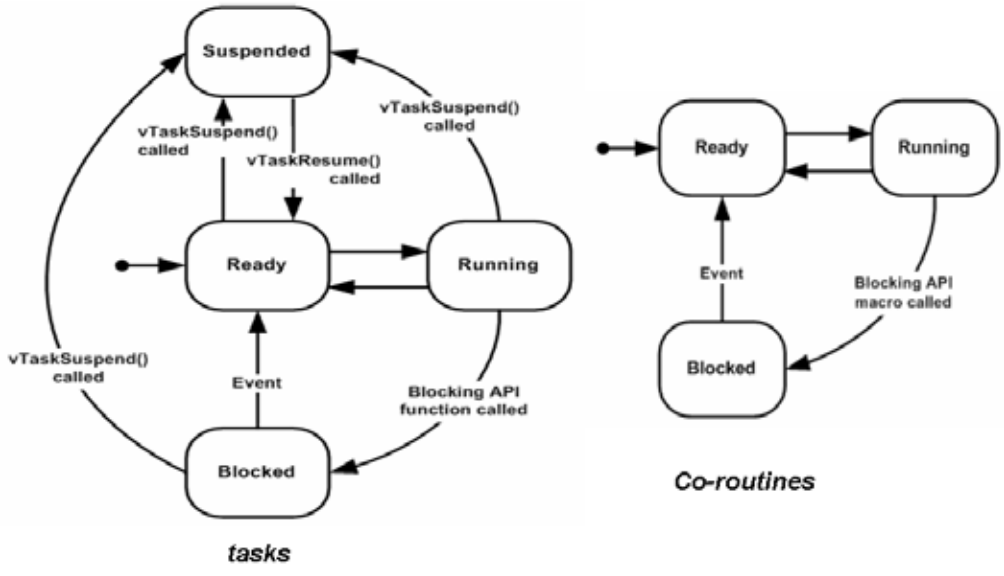


Figura 3.18 – Máquinas de estados suportadas pelo FreeRTOS (BARRY, 2006).

3.8 – O PROJETO DO HARDWARE

O projeto e o desenvolvimento do hardware fazem parte de um esforço conjunto do Grupo de Processamento Digital de Sinais (GPDS), do Laboratório de Tratamento de Superfícies e Dispositivos (LTSD) – ambos do Departamento de Engenharia Elétrica da Universidade de Brasília – e do departamento de Computação da Universidade Católica de Goiás. Os trabalhos realizados tiveram como propósito viabilizar sensores (nós) para monitoramento da saúde humana que pudessem ser utilizados principalmente para as provas de conceito requisitadas deste trabalho de tese. Para isso, foram desenvolvidos o projeto e protótipos de um nó sensor e, também, de circuitos eletrônicos para aquisição de sinais fisiológicos.

3.8.1 – A arquitetura do nó sensor

O projeto da arquitetura do nó sensor foi pautado nas funcionalidades requeridas do hardware: capacidade de aquisição e condicionamento dos sinais analógicos; capacidade de digitalização e transmissão desses sinais (preferencialmente sem fios); utilizar baterias e apresentar tamanho e peso reduzidos. Isso com objetivo de promover ergonomia. Dessa forma, um outro aspecto considerado foi a utilização do menor número possível de blocos funcionais. Ainda, foram requeridos: baixo consumo de energia elétrica e baixo custo para a prototipação (menor que US$ 100,00 por cada nó sensor). A Figura 3.19 apresenta a arquitetura desenvolvida e utilizada como base para os nós sensores.

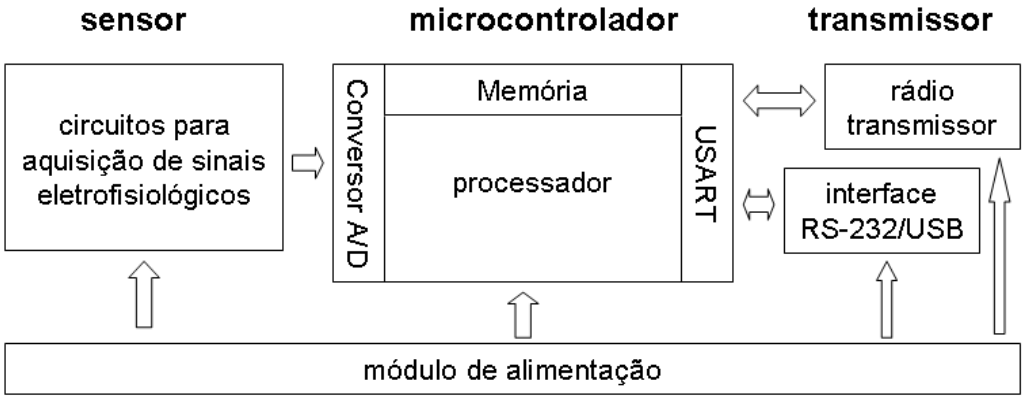


Figura 3.19 – Arquitetura do nó sensor.

O núcleo da arquitetura apresentada pela Figura 3.19 é microcontrolador. Para este projeto, o microcontrolador deve dispor: um processador, um conversor analógico/digital (A/D), capacidade de armazenamento (memória) e interface para comunicação serial (USART - *Universal Synchronous Asynchronous Receiver Transmitter*). Desses dispositivos, o conversor A/D e a memória merecem destaque.

A resolução igual a 12 bits para o conversor A/D é importante. Para muitos tipos de sinais a avaliação da forma de onda é o que determina o diagnóstico. Entretanto, dependendo da aplicação, como, por exemplo, para captura da pressão arterial, a resolução de 10 bits (maioria dos microcontroladores) é satisfatória.

Atualmente, microcontroladores de baixo custo (menor que US$ 10,00) possuem em média 32 KB (kilobytes) de memória RAM. Para muitas aplicações esse espaço é suficiente. Por exemplo, a captura de três canais do ECG amostrados a 1000 Hz com resolução de 12 bits, requer 4,5 KB de memória por segundo de retenção. Dessa forma, 32 KB de memória RAM é suficiente para a aplicação representada pelo agente da Figura 3.15b.

Para a transmissão dos dados foram especificados dois módulos:

- Uma interface para interconexão serial com fios. Pode ser implementada por meio do padrão RS-232 ou USB (sigla em inglês, *Universal Serial Bus*). A interface tem objetivo de facilitar a interconexão com dispositivos móveis (por exemplo, um telefone celular), quando já dispõem de interface para comunicação sem fios. O acoplamento em questão possibilita a economia de energia por não exigir a ativação de dois transmissores sem fios (o do sensor e o do dispositivo móvel) ativos ao mesmo tempo;
- Um rádio transmissor sem fios com alcance de até 100 metros, baixo consumo de energia e capacidade de operar a taxas de até 100 Kbps. Por exemplo, o agente representado pela Figura 3.15b pode requerer a transmissão de dados a taxas de 36 Kbps. Outro aspecto relevante refere-se à implementação da pilha de protocolos Bluetooth. É desejável que o rádio transmissor disponha de memória e capacidade de processamento. Isso é importante porque possibilita que o microcontrolador possa ser utilizado para as outras funcionalidades. Além disso, o rádio transmissor deve ser pequeno, leve e com custo acessível para a prototipação dos sensores.

O módulo de alimentação é composto por um circuito regulador de tensão, um circuito terra virtual e a fonte de energia (pilhas). O regulador de tensão é um circuito que tem como objetivo compatibilizar as tensões elétricas entre a fonte de energia e os demais módulos do nó sensor, evitando que eventuais surtos de corrente e de tensão possam danificar os componentes eletrônicos.

O circuito terra virtual é utilizado quando é necessário gerar uma referência para a tensão de alimentação. Em geral, isso é requerido por amplificadores de baixo custo, com entrada JFET (*Junction gate Field-Effect Transistor*). Por exemplo, amplificadores da família TL e INA (*Texas Instruments*), muito empregados em circuitos de instrumentação biomédica. Esses amplificadores requerem que a tensão de alimentação seja simétrica, isto é, com dupla polaridade (+VCC e –VCC) e, ainda, uma referência (0 V). A Figura 3.20 mostra o esquema elétrico do circuito terra virtual utilizado neste trabalho. A tensão de entrada fornecida pelo regulador de tensão (VCC) é dividida pelos resistores de 1 KΩ combinados em série. O *buffer* (configuração do amplificador operacional) é utilizado para evitar perda de potência do sinal. Também, é possível adicionar dois capacitores em paralelo aos resistores de 1KΩ com objetivo de minimizar ruídos gerados pelos outros elementos do módulo de alimentação. Ainda, é possível adicionar um transistor na saída deste circuito quando a corrente fornecida precisa ser aumentada. Uma alternativa ao terra virtual é utilização de circuitos integrados conversores de tensão. Esses são mais precisos, entretanto, têm custo muito maior.

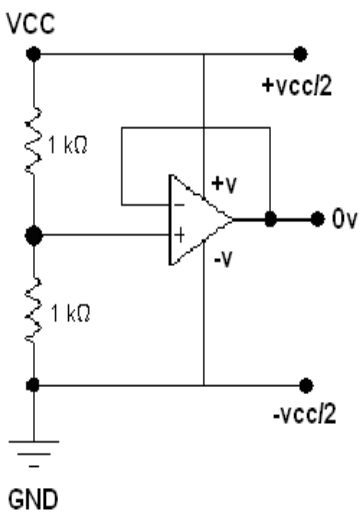


Figura 3.20 – Circuito terra virtual.

3.8.2 – Circuitos para aquisição de sinais eletrofisiológicos

Os circuitos projetados neste trabalho têm a seguinte estrutura básica: eletrodo específico para aquisição do sinal, amplificadores do sinal capturado e filtros analógicos.

3.8.2.1 – O circuito de ECG e EMG

O eletrocardiograma ou ECG é o registro gráfico da atividade bioelétrica do coração obtido na superfície corporal. É um registro dos potenciais elétricos gerados pelo coração ao longo do tempo. O EMG ou eletromiograma corresponde ao mesmo tipo de registro, entretanto, refere-se a um sinal elétrico proveniente da musculatura que possibilita investigar as funções musculares.

Há dois métodos principais de coleta do sinal eletromiográfico. O método invasivo, que requer a utilização de agulhas e/ou microeletrodos. É o método empregado na prática clínica, entretanto, causa dor e desconforto ao paciente. O outro método é conhecido como eletromiografia de superfície (EMG-S), pelo fato de utilizar eletrodos metálicos na superfície da pele, do tipo Ag-ClAg e gel salino condutor de eletricidade. Ao contrário do método invasivo, não é restrito aos médicos. É amplamente empregado por fisioterapeutas e por profissionais da área desportiva (VENEZIANO, 2006).

Os circuitos para aquisição de sinais de ECG e EMG-S são mostrados na Figura 3.21. Ambos têm basicamente a mesma estrutura, entretanto, têm filtros diferentes em relação às freqüências de corte. Em suma, esses circuitos são compostos por:

- Um amplificador de instrumentação com razão de rejeição de modo comum elevado (aproximadamente CMRR=110 dB). Isso é importante para minimizar ruídos que surgem eventualmente na entrada do circuito e acabam sendo adicionados ao sinal de interesse;
- Duas etapas de amplificação: a primeira (de pequeno ganho) tem a função de evitar saturação do sinal de interesse. É dimensionada pelos valores dos resistores que regulam o ganho no amplificador de instrumentação. Para o circuito apresentado na Figura 3.21 o ganho atribuído ao sinal de entrada (nesse estágio) é igual a 10 (dez). A segunda etapa de amplificação possibilita que o sinal de interesse alcance o nível de tensão adequado para que o conversor A/D possa executar a digitalização do

sinal. O ganho atribuído ao sinal de entrada nesse estágio é igual a 100. Por exemplo, o ECG capturado na superfície da pele apresenta amplitude aproximada de 1 mV. Ao passar pela primeira etapa de ganho alcança amplitude aproximada de 10 mV e ao passar pela segunda etapa de ganho alcança amplitude aproximada de 1000 mV;

- Duas etapas de filtragem: um filtro passa altas e um filtro passa baixas. Esses filtros são utilizados para minimizar os ruídos existentes. Também, para limitar o sinal numa determinada banda passante de interesse. Os principais ruídos são gerados pelas componentes DC encontradas no sinal de entrada (nas baixas freqüências) e a interferência da rede elétrica que gera ruído na freqüência de 60 Hz e nas harmônicas. Neste projeto, a banda passante utilizada pelo circuito de ECG é de 0,07-48 Hz. Para o EMG é de 20-497 Hz;
- Circuito da perna direita **(***driven-right-leg system*) que tem como objetivo colocar o paciente no mesmo potencial do circuito eletrônico. Isso evita problemas de descarga elétrica (prevenindo choques elétricos**)** e, principalmente, minimiza ruídos gerados pelos potenciais de modo comum, que aparecem quando o corpo do paciente não está aterrado ou no mesmo potencial que o de referência dos amplificadores.

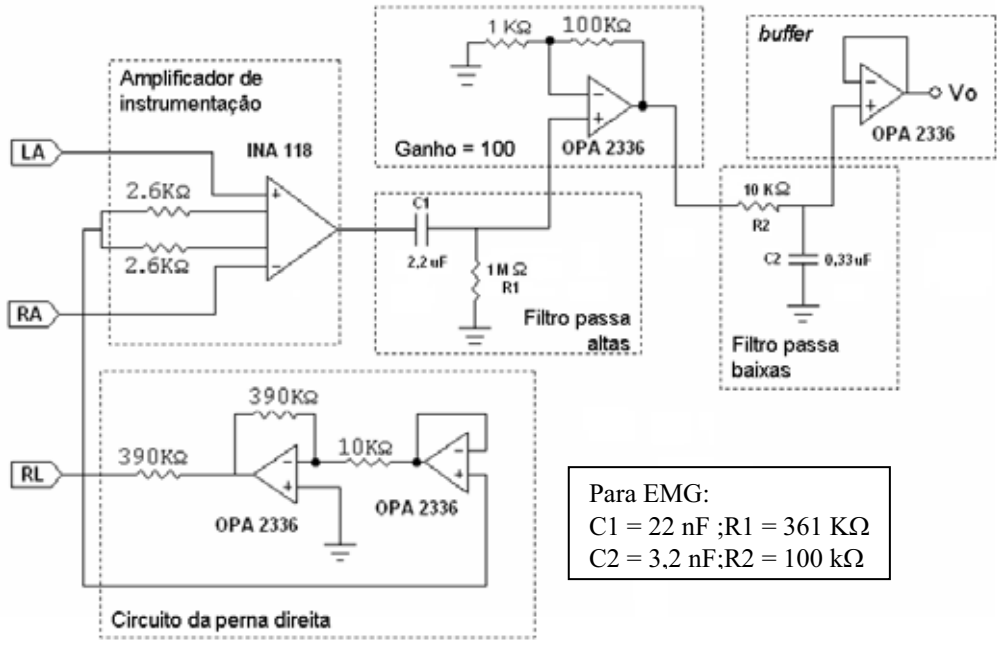


Figura 3.21 – Circuito para aquisição do ECG e do EMG.

O *buffer* introduzido na saída do circuito (Figura 3.21) tem como objetivo apenas evitar a perda eventual de potência do sinal que, no estágio seguinte, será digitalizado pelo conversor A/D.

3.8.2.2 – O circuito de resistência galvânica da pele

Uma fonte de informações para a avaliação do sistema nervoso é a resistência galvânica da pele ou GSR (*Galvanic Skin Resistance*). É a medida de condutância da pele obtida por dois eletrodos. Esse parâmetro é adquirido aplicando uma pequena corrente elétrica através de dois eletrodos conectados aos dedos de uma pessoa. GSR apresenta uma resposta que corresponde a variações no valor da condutância ao longo do tempo. Essa variação da condutância da pele é função da atividade das glândulas sudoríparas e do tamanho dos poros da pele. Um aumento da condutividade, por exemplo, pode ser causado pelo aumento da umidade da pele, pela atividade de secreção das glândulas sudoríparas ou, até mesmo, pelo conjunto desses fatores, que são provocados pela atuação do ramo simpático do sistema nervoso. A resistência galvânica da pele pode ser usada também como um medidor do nível de estresse ou um detector de mentiras, já que quanto mais relaxado, mais a pela estará seca e, portanto, maior a resistência.

O circuito desenvolvido para aquisição da resistência galvânica da pele é apresentado pela Figura 3.22. Trata-se de um divisor de tensão constituído por uma resistência conhecida e pela resistência da própria pele. Possui dois *buffers*. Com o *buffer* de entrada, a impedância de entrada vista pela tensão de alimentação é idealmente infinita. Com o *buffer* de saída a baixa impedância de saída isola o circuito da carga, evitando assim a perda de potência do sinal. Os amplificadores especificados não necessitam de alimentação simétrica e podem ser alimentados com o mesmo valor de tensão utilizado na entrada do circuito (VCC), fornecida pelo módulo de alimentação sem a necessidade do terra virtual.

Para minimizar o ruído provocado pela alta impedância de entrada (que funciona como uma antena que capta ruídos de alta freqüência) foi inserido um capacitor (0,1μF) em paralelo com a resistência da pele. Esse arranjo cria um filtro passa baixas e isso elimina as variações na saída do circuito para um valor constante de resistência da pele.

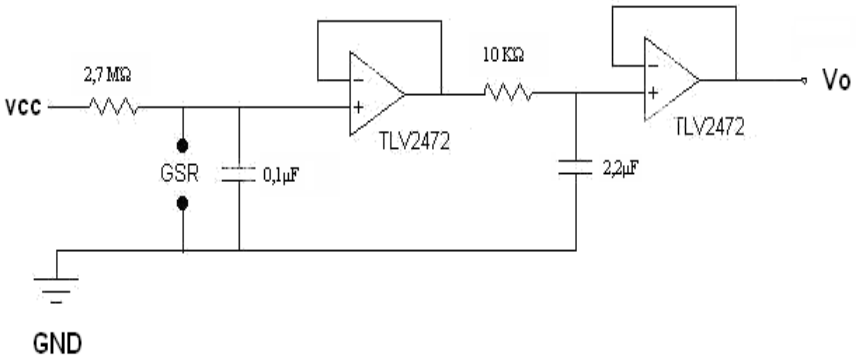


Figura 3.22 – Circuito para aquisição de sinais GSR (GUTIÉRREZ *et al*., 2006).

Neste circuito há também necessidade de um filtro passa baixas para minimizar a influência do sinal de 60 Hz da rede elétrica. A freqüência de corte do filtro pode ser baixa. Isso possibilita retirar uma maior parte do ruído de 60 Hz sem distorcer o sinal. O filtro passa baixas foi projetado para operar com freqüência de corte igual a 7,234 Hz. A resposta do circuito GSR (em kilo-ohms) é representada pela Equação 3.2.

$$GSR = \frac{2700 \times V_0}{VCC - V_0} \qquad \textbf{(3.2)}$$

De acordo com a Equação 3.2, uma tensão de entrada igual a 5 V sofre uma divisão entre um resistor de 2.700 KΩ (valor máximo de resistência que a pele pode alcançar) e a resistência da pele (GSR). Para o valor máximo de GSR, a saída do circuito deve ser de 2,5 V e para o outro extremo aproximadamente 0 V. Isso facilita a utilização da faixa de operação do conversor A/D embutido na maioria dos microcontroladores disponíveis atualmente.

Cumpre mencionar que no caso da resistência galvânica da pele, o valor absoluto nem é tão importante, uma vez que dependendo de onde for realizada a medida ele pode ser diferente. Na verdade, o que deve ser avaliado é a variação da resistência galvânica, ou seja, como ela está se modificando ao longo do tempo**.**

3.8.2.3 – O circuito para aquisição da temperatura cutânea

A análise do sinal de temperatura do corpo humano (*feedback* termal) está também mensurando o fluxo sangüíneo na pele. Quando os pequenos vasos na pele se dilatam, o fluxo sangüíneo e a temperatura aumentam, e quando esses vasos se contraem, o fluxo sangüíneo e a temperatura diminuem. Os vasos nos dedos são particularmente sensíveis ao estresse (vasoconstrição) e relaxamento (vasodilatação). Desta maneira, o *feedback* de temperatura dos dedos é uma ferramenta útil em treinamento de relaxamento, uma vez que quando os músculos estão contraídos e a pessoa está tensa, a temperatura diminui porque menos sangue alcança os dedos. É por isso que quando estão ansiosas com alguma situação, as pessoas têm a sensação de que as mãos estão frias. (Guyton & Hall, 2006).

Existem também estudos que relacionam as diferenças das temperaturas obtidas do lado direito e do lado esquerdo do corpo com a atividade cerebral. A capacidade de regular a temperatura corporal é exercida pelo hipotálamo. Ele é informado sobre a temperatura corporal por termorreceptores periféricos (sensores na superfície da pele) e por neurônios localizados na parte anterior do hipotálamo (que funcionam também como termorreceptores). Assim, o hipotálamo funciona como um termostato capaz de detectar variações de temperatura no sangue que por ele passa e ativar os mecanismos de perda ou de conservação do calor necessários à manutenção da temperatura normal. Esses mecanismos são ativados em dois centros separados. O de perda de calor (hipotálamo anterior) que desencadeia fenômenos de vasodilatação e sudorese e o de conservação de calor. O hipotálamo posterior é responsável pela vasoconstrição periférica e tremores musculares ou calafrios. Como curiosidade, uma lesão no centro de perda de calor do hipotálamo, por exemplo, em conseqüência de traumatismos cranianos pode causar uma elevação incontrolável da temperatura corporal. Isso é chamado de febre central e quase sempre leva a pessoa à morte (Guyton & Hall, 2006).

O projeto do circuito utilizado para aquisição da temperatura cutânea (TC) é apresentado pela Figura 3.23. Este projeto foi desenvolvido com base no circuito proposto por Bishop (2000). Tem como objetivo efetuar a leitura do valor da tensão sobre o termistor, cuja resistência varia de acordo com a temperatura. O valor de tensão obtido é amplificado e filtrado. Posteriormente, é correlacionado com um valor de temperatura que relaciona temperatura e a resistência do termistor.

Para este projeto foi especificado um termistor do tipo NTC (*Negative Temperature Coeficient*) de 10 KΩ. Esse dispositivo tem como característica a diminuição da resistência com o aumento da temperatura. Esse fenômeno ocorre porque ele é formado por materiais semicondutores. Com o aumento da temperatura (da energia), boa parte dos elétrons da camada de valência é liberada. Isso aumenta a condutividade do material e, portanto, diminui a resistência. A escolha desse tipo de sensor foi motivada pelo baixo custo e pela facilidade de compra (facilmente encontrado no comércio de componentes eletrônicos).

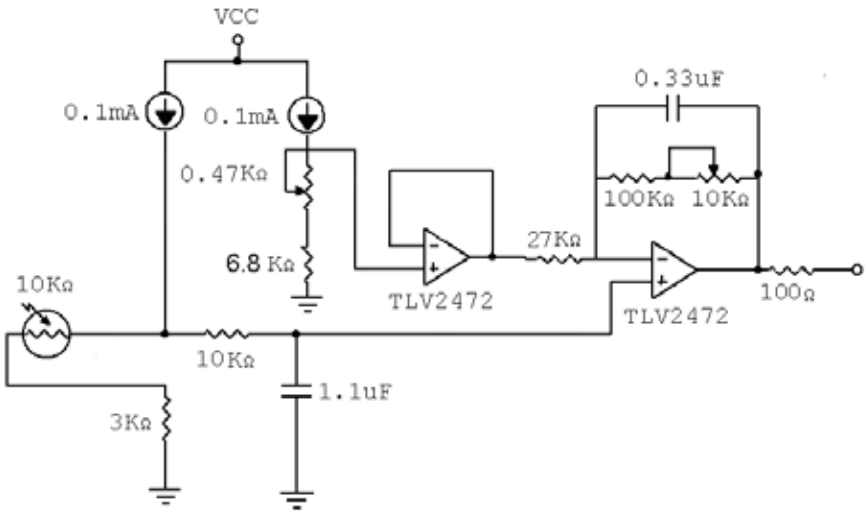


Figura 3.23 – Circuito para aquisição de sinais TC (GUTIÉRREZ *et al*., 2006).

Utilizando os recursos do laboratório de Física da UnB foi possível estimar a curva da temperatura em função da resistência. O procedimento experimental partiu de um recipiente com água no qual foram colocados um termopar e o termistor, cujo valor de resistência foi monitorado por um multímetro. Em seguida, a água foi aquecida. Para um determinado valor de temperatura a resistência correspondente foi anotada. O procedimento foi repetido várias vezes. Com isso, foi possível calcular a média aritmética entre as várias amostras obtidas para cada valor de temperatura. Também, foi considerado o valor da resistência da amostra da água utilizada. O valor da resistência da água utilizada nos testes foi aproximadamente 172 KΩ, independentemente, da temperatura. A Tabela 3.3 relata os dados obtidos.

Tabela 3.3 – Valores de resistência obtidos.

Temperatura (°C)	Resistência (Ω)	Temperatura (°C)	Resistência (Ω)
35	5779,0	65	1810,7
40	4820,9	70	1568,5
45	3842,2	75	1287,2
50	3247,5	80	1098,9
55	2658,2	85	929,9
60	2181,9	90	893,3

A função de Steinhart-Hart (apresentada pela Equação 3.3) caracteriza a resposta de qualquer termistor de acordo com a temperatura. Nela T é a temperatura (em graus Celsius), R é a resistência em Ohms (Ω) e α,β e φ são coeficientes.

$$T = \frac{1}{\alpha + \beta \times \ln(R) + \varphi \times \ln(R)^3} - 273.15 \qquad (3.3)$$

Com auxilio de um software e dos dados obtidos em laboratório (Tabela 3.3) foi possível estimar os valores dos coeficientes da equação de Steinhart-Hart. A Figura 3.24 exibe um gráfico com os resultados obtidos. Para melhor caracterizar a curva obtida, os valores de temperatura são fornecidos em graus Kelvin.

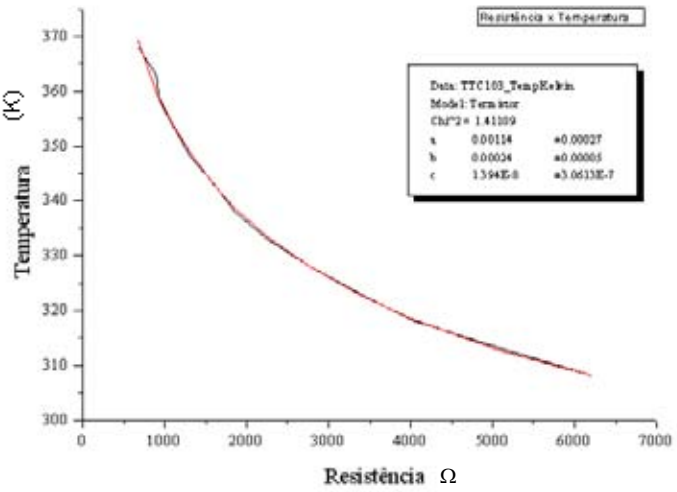


Figura 3.24 – Aproximação da função de Steinhart-Hart.

Foi adotada como faixa dinâmica de trabalho do circuito o intervalo entre 303°K (30°C) e 323°K (50° C). Esse intervalo é suficiente para aferir a temperatura cutânea. Por meio da

Equação 3.3 foi possível calcular os dois valores de resistência que limitam a faixa dinâmica de trabalho. Os valores encontrados foram 7730Ω (30°C) e 3330Ω (50° C).

Então, os potenciômetros (que aparecem na Figura 3.23) foram calibrados de modo que na saída do circuito o valor de tensão seja igual 0,330 V. Este valor representa a temperatura igual a 50° C (limite superior). O outro valor utilizado foi 2,52 V, que corresponde à temperatura de 30° C (limite inferior). A função que relaciona tensão e resistência é mostrada na Equação 3.4. Os valores dos coeficientes *a* e *b* foram calculados por meio de um sistema linear de duas equações e duas incógnitas. Os valores obtidos foram *a= 2009* e *b =2667.*

$$R = a \times tensão + b \qquad (3.4)$$

Pela Figura 3.23 pode-se verificar o emprego de uma fonte de corrente regulada. Ela é utilizada de modo a fornecer um valor constante para a corrente elétrica que flui pelo termistor. Dessa forma, o valor da tensão sobre o termistor deve variar linearmente em função da resistência (lei de Ohm). A utilização de uma fonte de corrente ao invés do divisor de tensão (solução clássica) possibilita maior precisão. Além disso, no caso do divisor de tensão, a corrente que passa pelo resistor varia de acordo com a resistência. Para valores pequenos da resistência, a corrente elétrica é muito alta. Isso poderia provocar o aquecimento do termistor e uma possível queimadura do paciente. Para implementar a fonte de corrente foi especificado o circuito integrado REF200 [18]. Esse componente possui duas fontes de corrente. Isso é necessário uma vez que a segunda fonte de corrente é utilizada para gerar a tensão utilizada para calibração da tensão de saída do circuito.

Para minimizar a interferência do ruído de 60 Hz da rede elétrica e limitar a banda passante do sinal, um filtro passa baixas com freqüência de corte igual a 14,46 Hz foi projetado. Para isso, foi especificado um capacitor de 1.1 μF e um resistor de 10 KΩ. Ainda, foi projetado um segundo filtro passa baixas na saída do circuito. Esse filtro reforça as características do primeiro e, também, minimiza ruídos introduzidos pelos componentes do circuito pela indução eletromagnética.

[18] http://focus.ti.com/docs/prod/folders/print/ref200.html

Para este circuito foi especificado o amplificador TLV2472 (*Texas Istruments*). Esse circuito integrado tem baixo consumo de energia elétrica e não necessita de alimentação simétrica.

3.8.2.4 – O circuito para aquisição da pressão arterial

Pressão arterial corresponde à força exercida pelo sangue contra a parede da artéria. Quando se diz que a pressão arterial é de 50 mmHg, isso significa que a força exercida é suficiente para empurrar uma coluna de mercúrio até o nível de 50 mm de altura e aí mantê-la. O bombeamento cardíaco é pulsátil, a pressão arterial flutua entre um nível sistólico (pressão máxima) e um nível diastólico (pressão mínima). Há ainda a pressão média em cada ciclo do batimento cardíaco que é ligeiramente inferior à média aritmética da sistólica e da diastólica. (Guyton & Hall, 2006).

A medida da pressão arterial pode ser direta ou indireta. A medida direta é invasiva, realizada por meio da punção de uma artéria e a inserção de uma agulha ou cateter conectado a um transdutor calibrado que transforma o sinal mecânico (pressão arterial) em sinal elétrico. A medida indireta é uma estimativa da pressão arterial utilizando o esfigmomanômetro (de coluna de mercúrio ou aneróide) e o estetoscópio para ausculta dos sons de korotkoff [19] (método auscultatório) ou para palpação simultânea (método palpatório) do pulso arterial (Guyton & Hall, 2006). Há também o método oscilométrico que consiste na identificação, quantificação e análise dos pulsos oscilométricos para determinação da pressão arterial.

No método oscilométrico, quando a pressão do manguito de oclusão que circunda o membro do paciente (normalmente o braço) é maior que a pressão arterial sistólica, ocorrem pequenas oscilações (pulsos oscilométricos) que aumentam em amplitude à medida que o manguito é esvaziado e a pressão cai abaixo da pressão sistólica. Com essa redução na pressão do manguito a amplitude aumenta até atingir um máximo, que pode permanecer constante ou cair abruptamente. A partir daí, as oscilações se reduzem com o esvaziamento do manguito. Pela análise dos pulsos oscilométricos é possível determinar a

[19] Uma seqüência definida e muito característica de sons que aparecem subitamente quando o manguito de oclusão é inflado, mudam suas características e, posteriormente, desaparecem.

pressão arterial média (máximo da amplitude do pulso oscilométrico) e estimar a pressão sistólica e a diastólica (CERULLI, 2000).

As principais vantagens do método oscilométrico são:

- Praticidade. Não necessita da ausculta, portanto, a medição pode ser efetuada por um observador não-treinado. Isso facilita a implementação de um agente (software) para automatização do processo;
- Não há risco de choque elétrico porque não há passagem de corrente elétrica pelo corpo do paciente;
- Sensores e componentes eletrônicos de baixo custo. Isso reduz os custos finais do equipamento.

Entretanto, há muita controvérsia sobre o uso e a precisão de equipamentos que usam o método oscilométrico, principalmente, por causa dos modelos computacionais e algoritmos utilizados para detecção da pressão sistólica e diastólica nos pulsos oscilométricos.

Neste trabalho, o circuito para aquisição da pressão arterial (PA) não-invasiva é apresentado na Figura 3.25. Este circuito foi projetado para utilizar o método oscilométrico e concebido com base no circuito desenvolvido por C. S. Chua (CHUA & HIN, 2005). O circuito em questão é composto por um sensor de pressão tipo Gauge MPX5050 GP, fabricado pela Freescale Semicondutor, e um módulo de amplificação e filtragem, utilizado para obtenção dos pulsos oscilométricos extraídos do sensor do sinal de pressão.

O sensor de pressão é acoplado diretamente ao manguito de oclusão que circunda o braço do paciente. Este sensor foi escolhido por apresentar boa precisão e saída já amplificada. Isso possibilita interfacear diretamente o sensor com o módulo conversor A/D do microcontrolador. Ainda, uma outra característica importante foi o custo acessível, cerca de US$ 8,00 por unidade. A função de transferência deste sensor é dada pela Equação 3.5.

$$\text{Tensão}_{saída}\,(\text{v}) = 5{,}0\ \text{V(alimentação)} * (0{,}018 * \text{Pressão}_{lida} + 0{,}04) + erro \qquad (3.5)$$

Os pulsos oscilométricos (aproximadamente 1Hz) são transportados pelo sinal de pressão (aproximadamente 0,04 Hz). Se o sinal de pressão não for completamente atenuado, a linha de base dos pulsos oscilatórios não será constante. Dessa forma, não será possível fazer as

comparações porque os pulsos poderão apresentar valores de amplitudes gerados com base em referências diferentes. A filtragem do sinal de pressão para obtenção dos pulsos oscilométricos utiliza um filtro passa altas com dois pólos que definem duas freqüências de corte distintas. As freqüências de corte utilizadas podem ser aproximadas pelas Equações 3.7 e 3.8.

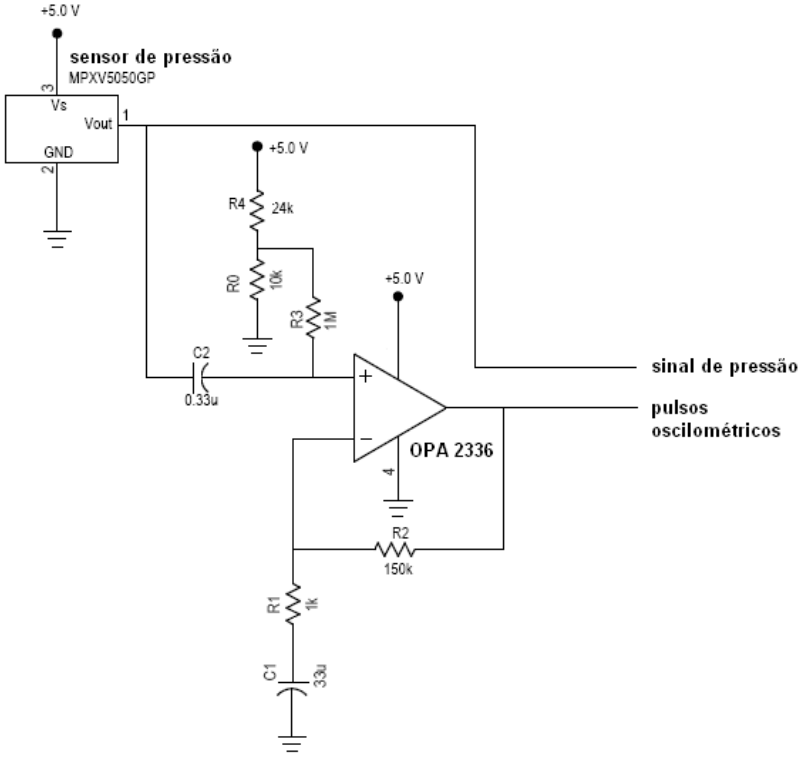


Figura 3.25 – Circuito para aquisição do sinal da pressão arterial não-invasiva.

$$fc1 = 1/ 2\Pi \, R1C1 \qquad (3.7)$$

$$fc2 = 1/ 2\Pi \, R3C2 \qquad (3.8)$$

Os pulsos oscilométricos variam de pessoa para pessoa. Em geral, de 1mmHg até 3 mmHg. Considerando a função de transferência do sensor e a atenuação de 10 dB gerada pelos filtros, os pulsos oscilométricos deverão apresentar amplitude de 3,8 mV até 11,4 mV. Isso é insuficiente para analise e correlação dos picos que definem a pressão média, máxima e mínima com o sinal de pressão obtido do manguito. Assim, um estágio de amplificação é necessário. Neste estágio, o fator de amplificação (ganho) especificado foi de 150.

Neste projeto, a escolha do amplificador OPA2336 foi baseada na qualidade da resposta (rapidez e precisão) deste componente, quando utiliza alimentação simples e tensões menores que 5 V. Ainda, por ser encontrado com facilidade no comércio eletrônico.

Entretanto, a utilização do amplificador LM324N no lugar do OPA2336 para o circuito mostrado na Figura 3.30 reduz custos. Para isso, é necessário um circuito auxiliar para compensação da tensão de *offset* de saída, que nesse tipo de amplificador operacional deve ser considerada. Na Figura 3.30, o circuito para compensação de *offset* é composto por um divisor de tensão, representado pelos resistores R4 e R0.

4 – PROTÓTIPOS E RESULTADOS

O objetivo deste capítulo é apresentar detalhes de implementação dos sistemas que compõem a arquitetura apresentada no Capítulo 3. Neste capítulo são descritas as tecnologias utilizadas e apresentados os protótipos desenvolvidos. Também, são relatados testes efetuados com os protótipos e os métodos utilizados nos testes.

4.1 – CAMADA DE APLICAÇÃO: *BWSNET CONFIGURATION TOOL*

Uma visão geral das classes que implementam *BWSNET Configuration Tool* é apresentada pela Figura 4.1. *BWSNET Configuration Tool* é composta por uma ferramenta para configuração das aplicações e por um conjunto de ferramentas para a programação dos sensores.

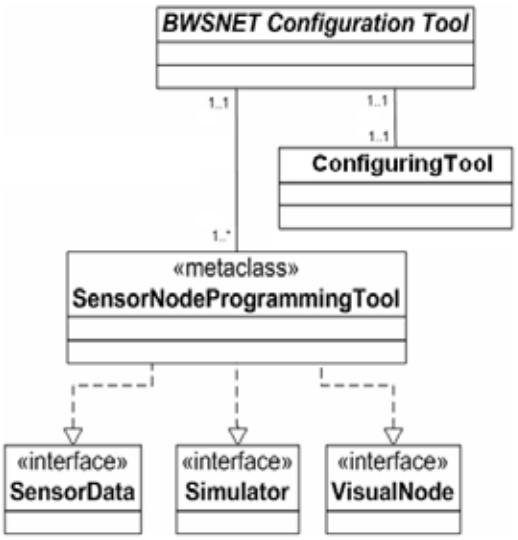


Figura 4.1 – *BWSNET Configuration Tool* (BARBOSA *et al*., 2006b).

4.1.1 – Ferramenta para programação dos sensores

A metaclasse "*SensorNodeProgrammingTool*" (apresentada na Figura 4.1) define o modelo de dados de cada nó sensor da rede. Para compatibilização das informações e funcionalidades necessárias ao modelo de programação proposto neste trabalho, cada sensor deve implementar as seguintes interfaces: *SensorData*, *Simulator* e *VisualNode*. Essas interfaces têm como objetivo uniformizar os requisitos apresentados na Figura 3.5.

Por exemplo, "*SensorData*" estende o modelo de dados definido para um nó sensor genérico, apresentado na Figura 3.7. Isso possibilita padronizar as informações acerca das capacidades e das funcionalidades de cada nó sensor, facilitando a inserção de novos nós sensores que não foram inicialmente planejados.

BWSNET Configuration Tool foi desenvolvida utilizando tecnologia Java [20]. A escolha dessa tecnologia ocorreu em função da portabilidade (independência de hardware e software) entre diferentes plataformas de suporte, da adequação à metodologia de desenvolvimento utilizada no projeto e, principalmente, em função da quantidade de bibliotecas de código e API (em português, Interface para Programação de Aplicações) disponíveis para uso gratuito. Essa característica facilita o trabalho do desenvolvedor porque requer menos tempo para programação dos protótipos.

Por exemplo, para suportar a inclusão de novos sensores e/ou novos subsistemas, *BWSNET Configuration Tool* implementa um mecanismo que utiliza o conceito de Reflexão Computacional (apresentado na Seção 2.2.2). Esse mecanismo tem como objetivo habilitar o sistema a carregar em memória um novo objeto (por exemplo, o programa que implementa um novo sensor) fazer sua inspeção e finalmente executá-lo a partir de uma invocação de métodos descobertos em tempo de execução. Para implementar esse mecanismo foi utilizada a API Java Reflection (Green, 2006). A Figura 4.2 mostra a tela implementada para adição e/ou remoção de novos sensores.

A inclusão de novos sensores e/ou subsistemas possibilita também que esses possam ser desenvolvidos utilizando várias tecnologias de programação, como, por exemplo, Applets, Java Applications (SUN, 2006b) e Thinlets (Bajazat, 2006). Ainda, a utilização de Thinlets possibilita que a interface do novo sensor possa ser montada e modificada com base num arquivo de configuração. Para padronização dos elementos gráficos que compõe as telas do novo sensor é recomendada a implementação da interface "*VisualNode*" (Figura 4.1). Essa interface define questões acerca da apresentação dos objetos, por exemplo, cores, localização dos botões, títulos (*labels*) e outros. Isso garante maior compatibilidade gráfica com *BWSNET Configuration Tool*.

[20] http://java.sun.com/

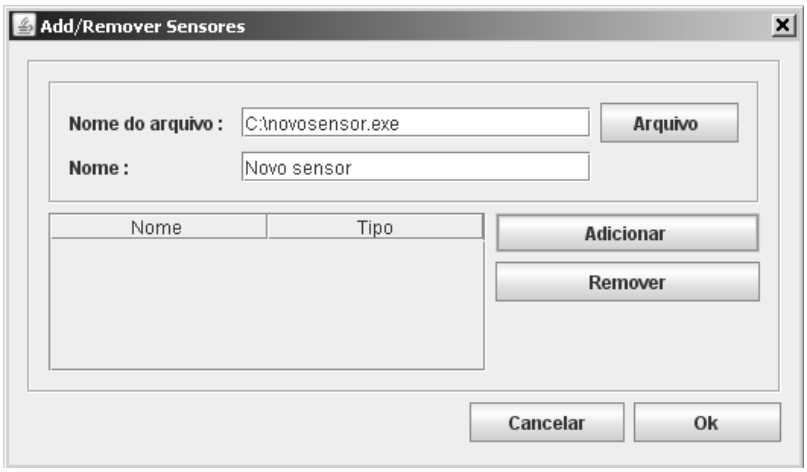


Figura 4.2 – Tela para adição e/ou remoção de sensores.

A Figura 4.3 apresenta o diagrama de seqüência que ilustra a invocação dinâmica de um novo sensor implementado por um objeto Java Application. Os objetos "cls", "retorno" e "m" representam meta-objetos. Ou seja, objetos que representam objetos externos ao sistema (novos programas) cujos métodos não são conhecidos em tempo de compilação. Essa dinâmica descreve como *BWSNET Configuration Tool* incorpora novos programas (que representam novos sensores e/ou subsistemas) sem a necessidade de reinicializar e recompilar esses programas. No exemplo apresentado pela Figura 4.3, um objeto Java Application é inicializado depois da descoberta do método "main()". Esse método estava contido na lista de métodos do novo objeto obtida durante a inspeção do mesmo.

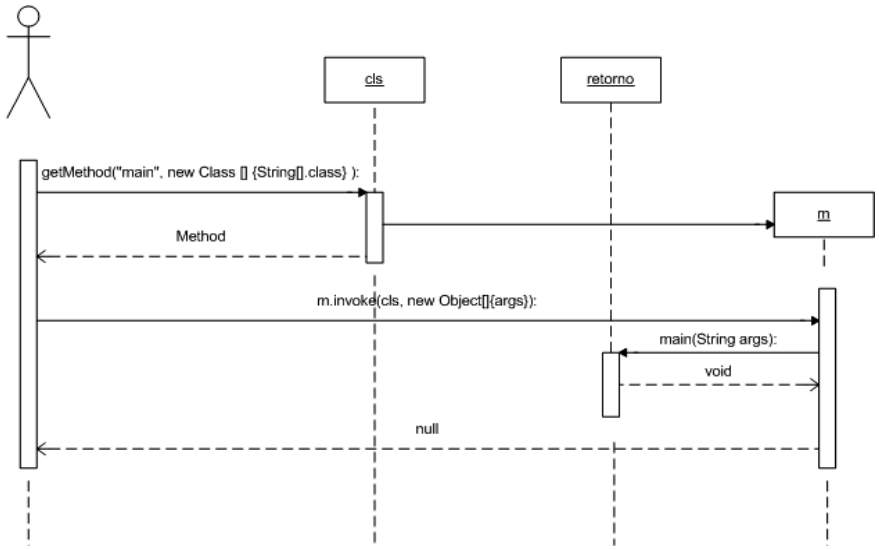


Figura 4.3 – Invocação Dinâmica de um novo sensor (BARBOSA *et al.*,2006b).

A programação dos sistemas é iniciada pela programação dos nós sensores. O nível de transparência oferecido aos usuários é configurável. O usuário pode simplesmente selecionar as tarefas requeridas dos sensores. Também, pode opinar pela inclusão de mecanismos e políticas para remoção de ruído, para gerenciamento de energia, para seleção dos alarmes, dentre outros. A Figura 4.4 ilustra a interface de programação de um nó sensor multifuncional para captura do eletrocardiograma (ECG).

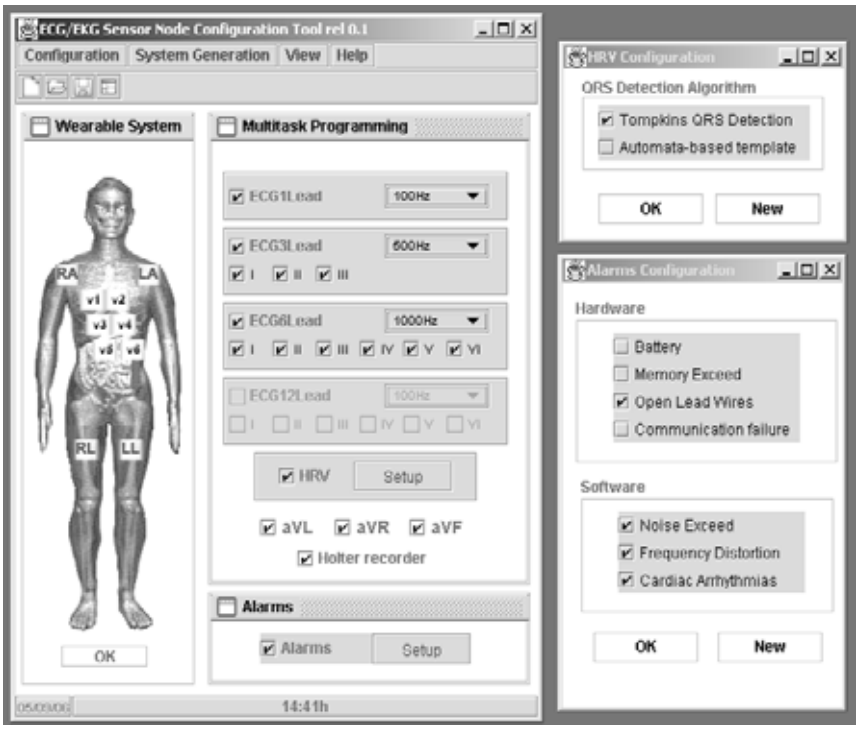


Figura 4.4 – Interface para programação do nó sensor multifuncional para captura do ECG (BARSOSA *et al.*, 2006b).

Por exemplo, se o usuário selecionar os eletrodos “RA”, “LA”, “LL” e “RL”, ele estará programando um sensor de ECG para oferecer até três diferentes funcionalidades. Pela paleta *Multitask Programming* (Figura 4.4) pode-se observar que aparecem selecionados o ECG com única derivação[21]operando com freqüência amostragem de 100 Hz, o ECG com três derivações operando com freqüência de amostragem de 500 Hz e o ECG com seis derivações operando com freqüência de amostragem de 1000 Hz.

[21] As derivações são identificadas pelos pontos (no corpo humano) onde são colocados os eletrodos. Cada par de eletrodos mede a diferença de potencial entre dois pontos. Na prática, cada derivação corresponde a um canal por onde são transmitidos os potenciais elétricos capturados por um par de eletrodos.

A partir do momento em que o usuário finaliza as escolhas, um arquivo no formato texto com as informações acerca da programação é gerado pelo sistema. Esse arquivo possibilita que os usuários possam catalogar os programas desenvolvidos e, principalmente, recuperar a programação de um sensor quando for preciso, sem a necessidade de efetuar um novo programa. Em seguida, ao comando do usuário (para efetuar a programação do sensor), o conteúdo desse arquivo é encaminhado pela camada subjacente (*middleware*) para o *gateway*, onde deve ser processado pelo gerador de agente. Em seguida, é efetuada a programação do sensor.

Para compatibilização com as tecnologias utilizadas pelo *middleware*, o arquivo de programação é gerado em formato XML (*EXtensible Markup Language*). Para isso, foi utilizada a API Java XMLEncoder. Essa API possibilita transformar um objeto Java num conjunto de bytes (serialização). Da mesma forma, possibilita recompor um objeto com base num arquivo XML, pela utilização de métodos específicos. Isso diminui o tempo para desenvolvimento de protótipos e minimiza a possibilidade de erros de programação porque o desenvolvedor não precisa desenvolver programas adicionais para processamento do texto XML. A Figura 4.5 exibe o formato e o conteúdo de um trecho do arquivo de programação do sensor. Nele são apresentadas as seleções efetuadas pelo usuário e mostradas na Figura 4.4.

ecg.xml*

```
<java version="1.5.0_06" class="java.beans.XMLDecoder">
 <object class="XMLECGData">
  <void property="BWSNET">
   <string>active</string>
  </void>
  <void property="DSV">
   <string>inactive</string>
  </void>
  <void property="ECG1">
   <string>100Hz</string>
  </void>
  <void property="ECG3">
   <string>100Hz</string>
  </void>
  <void property="ECG3I">
   <string>active</string>
  </void>
  <void property="ECG3II">
   <string>active</string>
  </void>
  <void property="ECG3III">
   <string>active</string>
  </void>
  <void property="ECG6">
   <string>active</string>
  </void>
  <void property="ECG6I">
   <string>active</string>
```

Figura 4.5 – Arquivo de programação do sensor de ECG.

Além de possibilitar que os usuários possam programar os sensores, *BWSNET Configuration Tool* prevê que os usuários possam simular a programação efetuada (veja Figura 3.5). Para isso, é necessário que seja disponibilizado um simulador, uma ferramenta que possibilite aos usuários avaliar os programas desenvolvidos antes que a programação do nó sensor seja de fato iniciada. Isso evita que o sistema seja programado desnecessariamente. A Figura 4.6 mostra a tela principal do simulador implementado para o nó sensor de ECG.

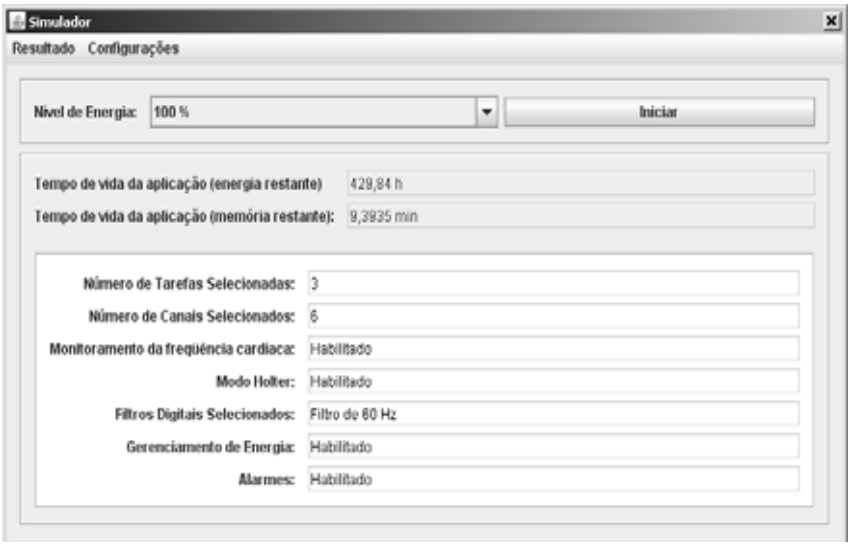


Figura 4.6 – Simulador para o nó sensor de ECG.

Na prática, o simulador tem como objetivo fornecer uma estimativa de desempenho do sensor com base na programação que o usuário pretende efetuar. Essa estimativa é relativa ao tempo de vida da aplicação (tempo de funcionamento do sensor) avaliada pelo consumo de energia do hardware e pela quantidade de memória requerida pela aplicação. Como informações de entrada para os cálculos, o simulador utiliza o arquivo de programação do sensor (Figura 4.5) e o nível de energia atual, estimado pelo próprio usuário. Além das estimativas de energia e memória restantes, o simulador fornece como resultado ao usuário um relatório (um resumo) da programação que está sendo avaliada.

Mudanças no hardware que implementam um determinado tipo de sensor não podem inutilizar o simulador. Para isso, é necessário um mecanismo que possibilite alterações nas políticas que implementam os cálculos executados pelo simulador. Essas políticas podem ser descritas num arquivo de configuração do hardware. Sempre que houver mudanças no

hardware, o simulador deve ser capaz de substituir as antigas políticas. Para isso, foi implementado um mecanismo baseado no conceito de Reflexão Computacional que possibilita a atualização do simulador à distância, sem a necessidade de recompilação do código fonte. Isso foi implementado a exemplo do que ocorre com as "atualizações automáticas" de alguns sistemas de software. Isto é, a atualização do simulador pode ser executada à distância (pela Internet) e completamente transparente para o usuário. A Figura 4.7 ilustra um trecho do arquivo de configuração utilizado para simular o hardware que implementa o nó sensor de ECG.

```
...
public class SimulatorExec {
 private int energyLevel;
 private Object simulator;
 private Map<String,Object> tasks;
 private double vi = 0.0; //energia estimada
 private double memoria = 1024; //memória em Kbytes
 private double aux,aux2;
 private double resultadoFinal;
 private int adc = 12; // resolução do conversor AD
...
 private void calc(){
    String deadline = "";
    String memoryDeadline = "";
    Object[] configs = (Object[])tasks.get("configs");
        try {
             vi = 120;//máximo 120 h (cinco dias)
             vi = energyLevel * vi;
             vi = vi / 100;
             //considera ajuste dinâmico de voltgem
             if ( ((Boolean)tasks.get("dynamicVoltage")).booleanValue() ){
                              vi = vi + (0.2 * vi);
                              memoria = memoria - 0.5;
                         }
            // considera politicas de desligamento habilitadas
                         if ( ((Boolean)tasks.get("enableShutdown")).booleanValue() ){
                              vi = vi + (0.5 * vi);
                              memoria = memoria - 0.5;
                         }
....
```

Figura 4.7 – Arquivo de configuração do hardware usado pelo simulador.

4.1.2 – Ferramenta de configuração

A ferramenta para configuração possibilita que os usuários possam definir as aplicações com base na definição de variáveis e fontes de dados disponíveis. Também, é possível declarar a importância da informação obtida por cada um desses elementos no contexto de cada aplicação, e, ainda, definir a situação (estado) atual de cada aplicação. Isso possibilita que o sistema possa se ajustar acerca da necessidade (disponibilidade) de recursos a cada momento. Em linhas gerais, a ferramenta de configuração é uma interface para configuração da rede e dos sensores concebida com base no MiLAN, apresentado na Seção 2.2.4.2. Para o desenvolvimento da ferramenta de configuração foi utilizada a API

jgraph[22]. Essa API é de uso gratuito e oferece componentes gráficos que facilitam o desenvolvimento. Um exemplo do uso de jgraph foi para implementação da caixa de ferramentas. Essa abstração é utilizada pelos usuários para definição dos grafos que representam uma aplicação.

A Figura 4.8 apresenta a tela principal da ferramenta de configuração utilizada para definição de uma aplicação para o monitoramento do estresse. Essa aplicação foi apresentada por Carvalho (Carvalho *et al*., 2004) e é composta por três variáveis baseadas em estados: pressão arterial, freqüência cardíaca e freqüência respiratória. Essas variáveis controlam cinco outras variáveis: pressão arterial, freqüência cardíaca, freqüência respiratória, oxímetro e ECG. Essas últimas são compostas pelos sensores: pressão, fluxo sangüíneo, oxímetro de pulso, oxímetro, ECG e sensor de respiração (resp). A importância (peso) que cada variável e/ou sensor exerce sobre o outro é definido pelo valor da aresta que varia entre 0 e 1.

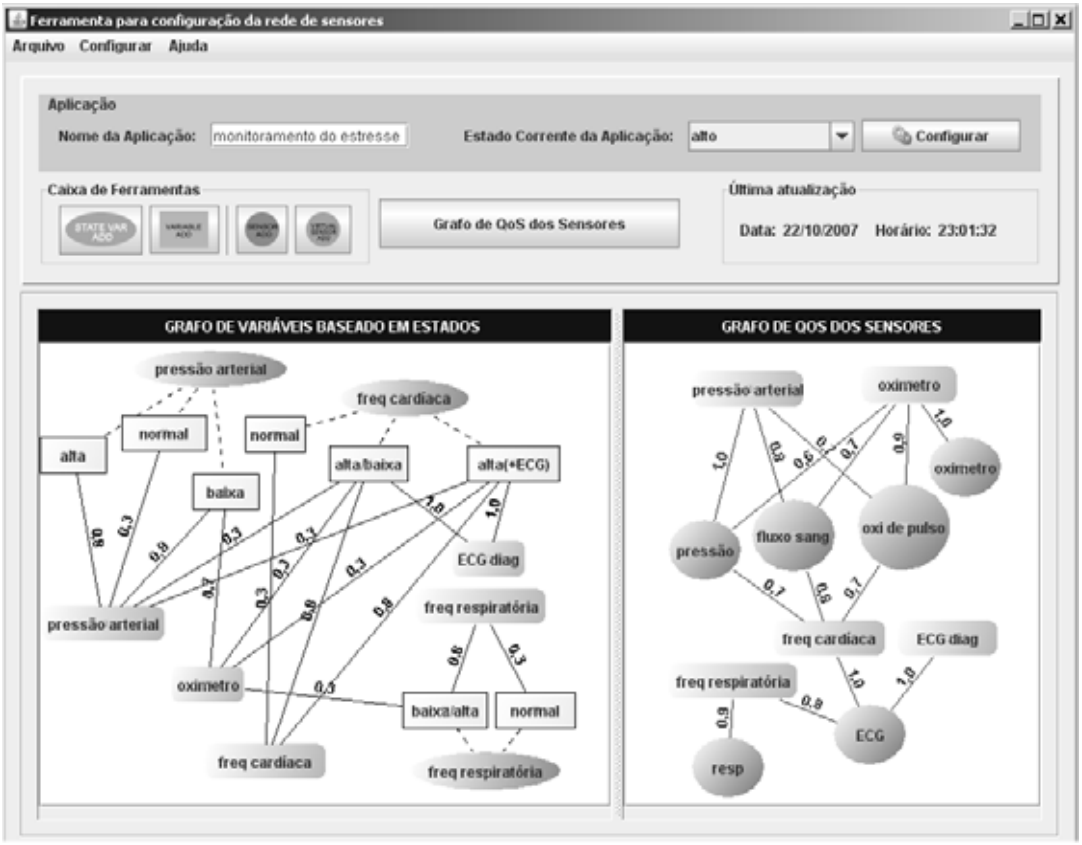


Figura 4.8 – Tela principal da ferramenta de configuração.

[22] http://www.jgraph.com/

A ferramenta de configuração amplia o modelo proposto pelo MiLAN que descreve uma aplicação, porque possibilita também que o usuário faça ajustes acerca das tarefas que estão sendo executadas pelos nós sensores. Essas tarefas foram especificadas pela ferramenta de programação durante a programação dos sensores. Na prática, isso possibilita que o usuário possa efetuar modificações a respeito do grafo de funcionalidades, apresentado na Figura 3.2. A Figura 4.9 mostra a tela de configuração para o nó sensor de ECG que aparece no grafo de QoS dos sensores (Figura 4.8). As tarefas que aparecem na Figura 4.9 devem ser definidas pela ferramenta de programação, apresentada na Figura 4.4.

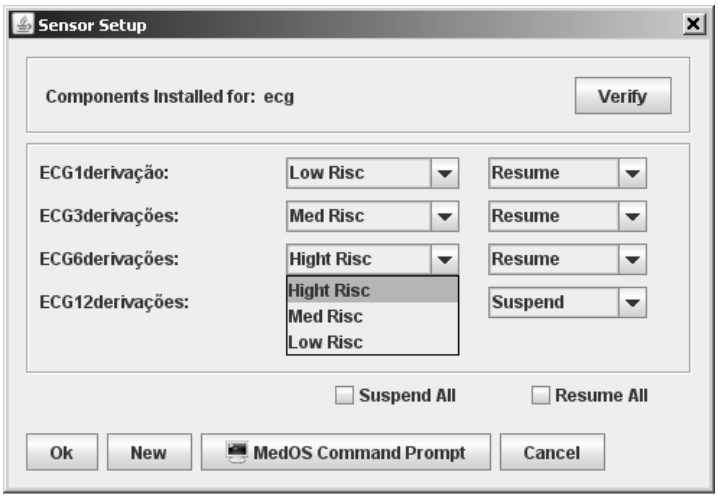


Figura 4.9 – Tela para configuração do sensor de ECG.

A tela "*Sensor Setup*" (Figura 4.9) ilustra os parâmetros configuráveis. O usuário pode alterar os valores de prioridade de cada tarefa e, também, suspender e reinicializar tarefas. Há ainda um terminal (*prompt*) de comandos que pode ser utilizado para verificação (*debugger*) do sensor. Esse terminal possibilita o envio direto de comandos para o sensor por intermédio do interpretador de comandos do MedOS, apresentado na Seção 3.7.

4.2 – CAMADA DE INTERCONEXÃO: *MIDDLEWARE* BASEADO EM SERVIÇOS

Na arquitetura SOAB, o *middleware* é responsável por fornecer as interfaces e mecanismos que implementam o acesso remoto aos serviços oferecidos pela rede de sensores do corpo humano. Pensando nisso, foi implementado um *middleware* baseado no paradigma RPC (em português, Chamada Remota de Procedimento). O RPC pressupõe que as interações

entre as partes sejam síncronas, com a passagem de parâmetros e a existência de valores de retorno e tratadores de erros. A solução implementada neste trabalho está em acordo com os protocolos especificados pelo comitê Web Services W3C e pela a arquitetura SOA (*Service Oriented Architecture*), apresentada na Seção 3.5.

4.2.1 – Os *webs services*

De acordo com Delicato (2005), *web services* são definidos como programas modulares, independentes e autodescritivos que podem ser descobertos e invocados através da Internet ou de uma Intranet corporativa. *Web services* são acessados por meio dos protocolos da Internet, por exemplo, o HTTP (*Hypertext Transfer Protocol*). A sintaxe utilizada pelos *web services* é baseada na linguagem XML (*EXtensible Markup Language*), desde o formato dos dados até a descrição dos serviços.

A tecnologia *web services* é montada numa pilha de protocolos composta de três níveis. O primeiro nível é chamado protocolo de rede, corresponde a um protocolo da camada de aplicação do modelo TCP/IP, por exemplo, o HTTP. O segundo nível é um mecanismo de codificação de dados baseado no formato XML (XML Schema) e o terceiro nível refere-se às mensagens XML trocadas para utilização do serviço.

Web services usam normalmente o protocolo SOAP (*Simple Object Access Protocol*) para interconexão do serviço (troca de mensagens). SOAP foi desenvolvido para ser simples, independente da tecnologia, independente da linguagem, independente do modelo de objetos e independente do protocolo de comunicação (camada de transporte) utilizado. SOAP especifica um mecanismo para definir a unidade de comunicação, um mecanismo para lidar com erros, um mecanismo de extensão que possibilita evolução do sistema e um mecanismo para representar tipos de dados em XML (Delicato, 2005).

XML é também a base para a descrição dos serviços. Além de disponibilizar o serviço, o provedor deve fornecer informações que são gerenciadas pelo servidor de gerência de serviços (*service broker,* apresentado na Seção 3.5) para que o consumidor possa, de fato, fazer uso das funcionalidade ofertadas. Para isso, utiliza-se uma linguagem padrão, também baseada em XML, chamada WSDL (*Web Service Description Language*).

De fato, existem várias alternativas para implementar o *middleware*. Todas apresentam vantagens e desvantagens. Por exemplo, a utilização de *sockets* exige que o desenvolvedor de sistemas conheça a sintaxe, a semântica e o significado das mensagens trocadas entre as partes envolvidas, também, onde essas mensagens aparecem no código-fonte, para o caso de futuras modificações. Além disso, para que uma solução com base nesse mecanismo possa ser padronizada para uso na Internet é necessário que o protocolo e o serviço sejam homologados pelo IETF (*Internet Engineering Task Force*), sob forma de RFC (*Request for Comments*). Há também outras questões técnicas que margeiam a solução adotada neste trabalho. Uma delas, por exemplo, refere-se aos números de porta utilizados para identificar as aplicações. Esses podem estar bloqueados por um *firewall* e, isso, dificulta e, até mesmo, impede a intercomunicação dos sistemas.

Outras alternativas, por exemplo, o RMI (*Remote Method Invocation*) da SUN, o .NET (lê-se ponto net) da Microsoft ou alguma implementação do padrão CORBA estão amarradas a tecnologias, paradigmas de programação ou, ainda, reféns das funcionalidades disponibilizadas por algum fabricante. Isso dificulta o crescimento (expansão) do sistema e reduz a capacidade de operação independente de plataforma.

A utilização de *web services* possibilita que a rede de sensores seja vista como uma entidade que fornece serviços para vários usuários com necessidades diferentes e dinâmicas, isto é, independentemente da camada 1: a interface para programação e configuração. Essa solução prevê ainda a incorporação de novas funcionalidades sob a forma de novos serviços que podem ser implementados por terceiros e adicionados ao sistema. A linguagem XML e o protocolo SOAP são padrões de fato da Internet. Isso fornece ao *middleware* capacidade de interoperabilidade e facilidade para crescimento. Aplicações podem ser escritas em diferentes linguagens de programação e interagir com a rede pelo suporte já existente de ferramentas para a geração automática de mensagens SOAP com base em documentos WSDL.

4.2.1.1 – *Web services* e SOAB

Como prova de conceito, foram implementados dois *web services.* Um para programação do nó sensor de ECG e o outro para envio de informações acerca da configuração da rede.

O primeiro deve ser executado para programação do sensor de ECG. O segundo deve ser executado sempre que houver necessidade de ajustes do sistema quando esses ajustes envolvam mudanças nos grafos e nos valores de prioridades que influem sobre o desempenho de uma aplicação.

A Figura 4.10 mostra o componente (*web service*) implementado para programação do nó sensor de ECG. Nele verifica-se a existência de três interfaces que correspondem a três operações que poderão ser executadas remotamente.

Figura 4.10 – Componente implementado para programação do nó sensor de ECG.

A operação "*compile*" é responsável por enviar para camada subjacente (o *gateway*) uma ordem de serviço para que a imagem do sistema[23] seja compilada. Após a compilação, essa imagem deve estar pronta para ser instalada no sensor. A instalação (em inglês *deployment*) do sistema é executada pela operação "*deploy*". A interface "*run*" possibilita que os usuários decidam o momento em que o sensor deve ser inicializado após a programação. Funciona como um botão "iniciar" implementado por software.

Para cada uma das operações do componente staticECGService, três diferentes mensagens SOAP foram criadas. Uma para invocação da operação (mensagem IN) e duas outras para respostas. Uma para caso a operação seja efetuada com êxito (mensagem OUT) e outra para informar sobre falha na execução da operação de invocação (mensagem outfault).

Toda descrição desse serviço pode ser acessada por meio do arquivo "ecgWSDL.xml". Para criação desse arquivo foi utilizada a linguagem WSDL versão 2.0 (W3C, 2006). A

[23] A imagem do sistema corresponde a todas as funcionalidades selecionadas pelo usuário. Isso constitui um conjunto de programas que ao ser compilado assume a forma de um único arquivo. Na prática, esses programas foram selecionados pelo usuário durante a programação do sensor e repassados para o componente por meio de um arquivo de configuração, gerado pela ferramenta de programação.

Figura 4.11 exibe um trecho do arquivo "ecgWSDL.xml" em que a operação "*compile*" é descrita.

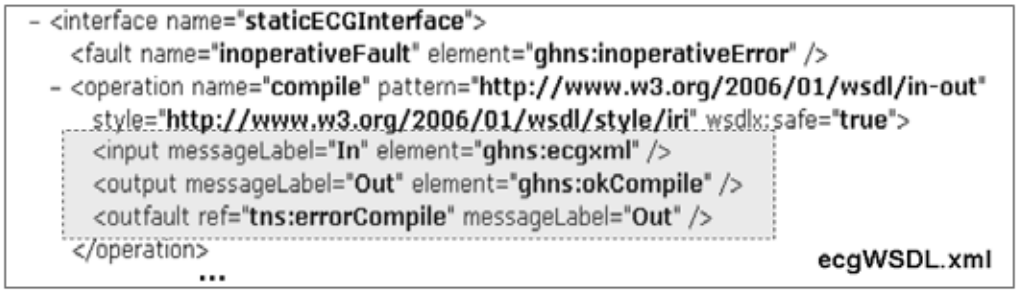


Figura 4.11 – Descrição da operação *compile*.

Por exemplo, uma requisição para a operação "*compile*" recebe como parâmetro "ecgXML". Este parâmetro representa um conjunto de informações referentes à seleção escolhida pelo usuário, por exemplo, número de tarefas, filtros selecionados, dentre outros. Este conjunto de informações corresponde ao conteúdo do arquivo de programação gerado por *BWSNET Configuration Tool* e apresentado na Figura 4.5. Como resposta à operação "*compile*" tem-se a mensagem "*okCompile*" em caso de sucesso e a mensagem "*errorCompile*" caso a operação não tenha sido bem sucedida.

Todos os protótipos do *middleware* foram construídos utilizando tecnologia Java. Como exemplo, podem ser destacados os programas para criação e interpretação das mensagens SOAP. Para aumentar a portabilidade não foi utilizado nenhum *framework*[24] durante a programação desses programas, como, por exemplo, o Xerces (APACHE, 2006). Todas as mensagens SOAP e as mensagens HTTP utilizadas para suporte aos *web services* foram programadas sem o uso de bibliotecas de código específicas. Isso facilita que instâncias do *middleware* possam ser compiladas e utilizadas em vários dispositivos, por exemplo, um PDA, um telefone celular ou outro dispositivo qualquer com suporte para Java. A Figura 4.12 apresenta um trecho do código-fonte que implementa as mensagens de resposta para a operação "*deploy*".

[24] Especificamente em orientação a objeto, *framework* é um conjunto de classes com objetivo de reutilização de um projeto, provendo um guia para uma solução de arquitetura em um domínio específico de software.

```
public static String soapDeployResponse(boolean status)
{String response = null;

 if (status == true) {
      response = printHeadersPost(true) + "\n\r" +
                          "<?xml version='1.0' ?> " +
                          "<env:Envelope xmlns:env=\"http://www.w3.org/2003/05/soap-envelope\"> " +
                          "<env:Header> </env:Header> " +
                          "<env:Body> " +
                          "<deployResponse env:encodingStyle=\"http://www.w3.org/2003/05/soap-encoding\" " +
                          "xmlns:deployResponse=\"http://localhost:8080/ECG/\"> " +
                          "<statusdeploy xmlns:statusdeploy=\"http://localhost:8080/ECG/\"> " +
                          "okDeploy " + "</m:statusdeploy> " +"</deployResponse> " +"</env:Body> " +
                          "</env:Envelope> " + "\n\r";
                             }

 if (status == false) {
             response = printHeadersPost(false) + "\n\r" +
                          "<?xml version='1.0' ?> " +
                          "<env:Envelope xmlns:env=\"http://www.w3.org/2003/05/soap-envelope\" " +
                          "xmlns:rpc=\"http://www.w3.org/2003/05/soap-rpc\"> " +
                          "<env:Body> " +  "<env:Fault> " +  "<env:Code> " +  "<env:Value> " +
                          "env:Sender " +  "</env:Value> " + "</env:Code> " + "<env:Reason> " +
                          "<env:Text xml:lang='en-US'> " +  "errorDeploy " +  "</env:Text> " +
                          "</env:Reason> " + "</env:Fault> " + "</env:Body> " +"</env:Envelope> " + "\n\r";
             }
return response;
}
```

Figura 4.12 – Implementação das mensagens de repostas para a operação *deploy*.

Em (DELICATO, 2005) é apresentado um *middleware* reflexivo para redes de sensores de propósito geral baseado na arquitetura SOA e na tecnologia *web services*. Esse sistema tem como foco disponibilizar uma visão abstrata da rede de sensores como fornecedora de serviços para as aplicações. Como componentes desse *middleware* foram implementados: um serviço de descoberta, um serviço de inspeção e/ou adaptação, um serviço de gerência de recursos e um serviço de configuração. Esse último é responsável pela seleção do protocolo de disseminação dos dados e da topologia da rede. Nesse trabalho, a arquitetura SOA é centrada em nós sorvedouros. A esses nós cabem o fornecimento do serviço para o meio externo. Na prática, a autora propõe que uma aplicação deve ser vista como consumidora dos serviços da rede de sensores disponibilizados por meio dos nós sorvedouros.

Em (WELSH *et al*., 2006b) é apresentado um *gateway* para interfacear uma rede de sensores do corpo humano com sistemas de informações médicas, por meio de *web services* compatíveis com a especificação Health Level-7 (HL7) versão 3. O cliente cria uma consulta (em inglês, *Query*). Em seguida, o sistema empacota a consulta numa mensagem HL7 XML e envia ao servidor (*gateway*). O servidor analisa a consulta e a repassa para a rede de sensores. O objetivo desse sistema é possibilitar que informações coletadas pela rede de sensores possam ser integradas com um sistema de informação, em

tempo real. Por exemplo, valores obtidos do sensor de oximetria de pulso são imediatamente integrados com um prontuário eletrônico do paciente. Nesse sistema, o *gateway* age como um tradutor entre a rede de sensores e um sistema de informação que utiliza o padrão HL7v3.

4.3 – CAMADA DE SERVIÇOS DA RSCH: *BWSNET PROXY*

O objetivo desta seção é apresentar os protótipos implementados que compõe o sistema BWSNET *Proxy*, apresentado na Seção 3.6.

4.3.1 – O gerador de agente

A Figura 4.13 apresenta o diagrama de classes que implementa o gerador de agente descrito na Seção 3.6.2. Cumpre lembrar que o gerador de agente é composto por três módulos: o módulo Parser, o módulo Seletor e o módulo Gerador. A classe "P_Recuperar_XML" relaciona com as classes "P_Estrutura" e "P_Montar_Tabela" por meio de uma associação do tipo agregação. Esse tipo de associação tem como objetivo assinalar que há obrigatoriedade de complementação das informações de um determinado objeto para com seus objetos-parte. Para extrair as informações da programação efetuada pelo usuário, a classe "P_Recuperar_XML" executa a leitura do arquivo XML. O conteúdo desse arquivo é recuperado num objeto. Esse objeto é convertido para um objeto do tipo "P_Estrutura". Isso é necessário para facilitar a manipulação dos dados desse objeto. Em seguida, o conteúdo desse objeto é manipulado pela classe "P_Montar_Tabela" para montagem da tabela de valores (descrita na Seção 3.6.2). As classes "P_Recuperar_XML", "P_Estrutura" e "P_Montar_Tabela" implementam o módulo Parser.

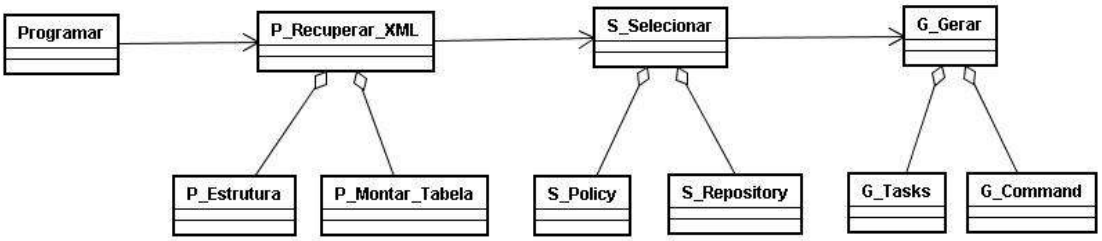


Figura 4.13 - Diagrama de classes do gerador de agente.

A classe “S_Selecionar” relaciona com a classe “P_Recuperar_XML”, com a classe “G_Gerar” e com a classe “Programar” por meio de uma associação simples. Isso implica que há apenas a troca de informações (por meio da invocação de métodos) para a continuidade de execução dos objetos.

A classe “S_Selecionar” relaciona com a classe “S_Policy” por meio de uma agregação. A classe “S_Repository” implementa os agentes que poderão ser selecionados pela classe “S_Selecionar”. Para seleção dos agentes, a classe “S_Selecionar” usa as informações enviadas pela classe “P_Recuperar_XML” e pela classe “S_Policy”. Essa solução separa o mecanismo para seleção de agentes (implementado na classe “S_Selecionar”) das políticas para seleção de agentes (implementado na classe “S_Policy”). Na prática, isso facilita mudanças futuras porque modificações nas políticas não requerem necessariamente modificações nos mecanismos. A classe “S_Selecionar” juntamente com a classe “S_Policy” implementam o módulo Seletor. A classe “S_Repository” implementa o repositório de agentes.

A classe “G_Gerar” se relaciona com as classes “G_Tasks” e “G_Command” também na forma de agregação. A classe “G_Tasks” e a classe “G_Command” fornecem os moldes (*templates*) para os agentes. Isto é, os trechos de código (em linguagem C) que poderão ser utilizados pela classe “G_Gerar” para composição do programa que será instalado nos sensores. A Figura 4.14 ilustra a implementação de um estado de um agente por meio de um molde de tarefa. A classe “G_Gerar” juntamente com as classes “G_Tasks” e “G_Command” implementam o módulo Gerador. A Figura 4.15 mostra um diagrama que representa a seqüência de ativação das classes (objetos) que compõe o gerador de agente e seus módulos.

```
public class G_Tasks {

        ...

        String task3= "static void task3( void *pvParameters )\n" +
                      "unsigned char convertedECG [10];\n"+
                      "for (;;)\n{\n"+
                      "itoa (getECG(600),convertedECG,10);\n"+
                      "usart0Puts(\"\\r \");\n"+
                      "usart0Puts(convertedECG);\n"+
                      "usart0Puts(\"\\r\");\n";

        ...
```

Figura 4.14 – Implementação de um estado de um agente por meio de um molde de tarefa.

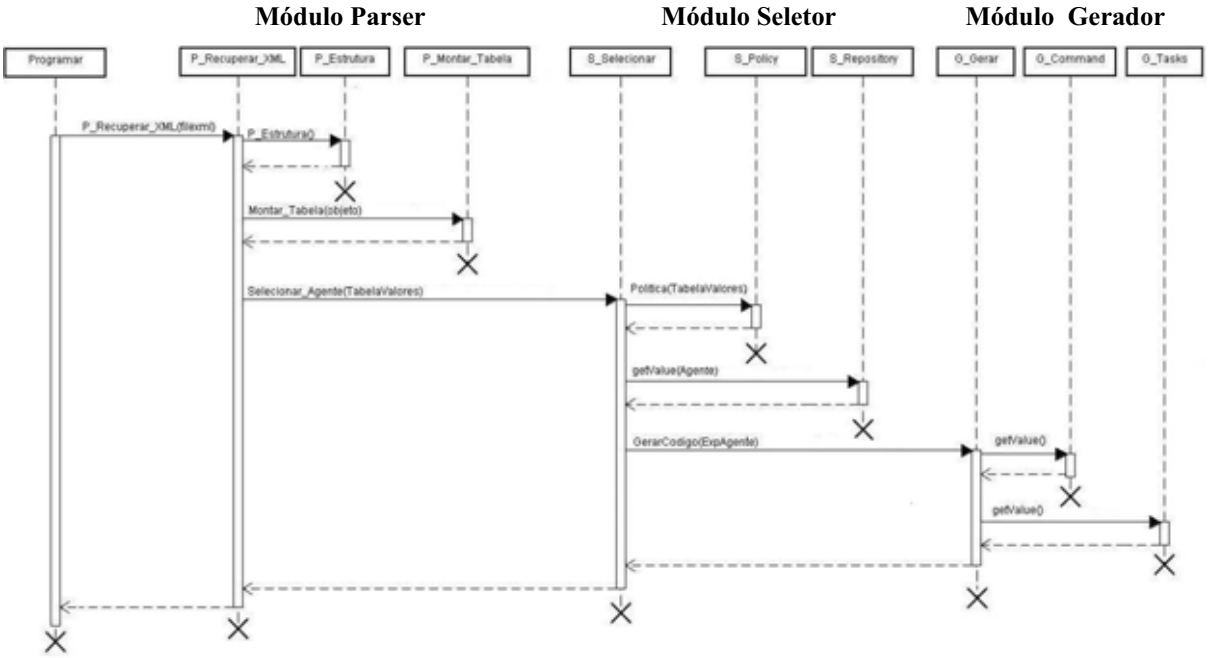


Figura 4.15 – Ativação dos programas do gerador de agentes

Todos os módulos do gerador de agente foram programados utilizando tecnologia Java. Neste caso, o principal benefício do uso dessa tecnologia é a disponibilidade de bibliotecas de código que oferecem funcionalidades específicas. Isso facilita o trabalho do desenvolvedor. Um exemplo disso é a API Java Beans. A classe "P_Estrutura" utiliza essa API por meio da classe XMLDecoder para gerar o objeto de dados com base num arquivo XML (arquivo com a programação do usuário). A Figura 4.16 ilustra a utilização da API Java Beans. O objeto "decoder" invoca o método "readObject". O resultado é uma referência para um objeto que contém dados do arquivo XML. Em seguida, essa referência é convertida para o tipo "P_Estrutura". Isso possibilita manipular os atributos presentes no objeto.

```
...
public class P_Recuperar_XML {

  public P_Recuperar_XML(String filexml) {

        try { String filexml = "ECG.xml";|
              XMLDecoder decoder = new XMLDecoder (new FileInputStream (filexml));
              P_Estrutura x = (P_Estrutura) decoder.readObject();

...
```

Figura 4.16 – Trecho de código que implementa a aquisição de dados efetuada pelo módulo Parser.

A classe “S_Selecionar” apresentada na Figura 4.13 implementa o mecanismo para seleção do melhor agente. A definição do “melhor” agente deverá seguir os critérios definidos pela política de seleção de agentes, definida com base no conhecimento especialista. A Figura 4.17 ilustra um trecho do código que implementa o mecanismo que ativa uma política para seleção de agentes. Para implementação desse mecanismo foi utilizada a API Java Reflection. Com isso, é possível que a política selecionada substitua a antiga política sem a necessidade de recompilação dos programas que estão em execução.

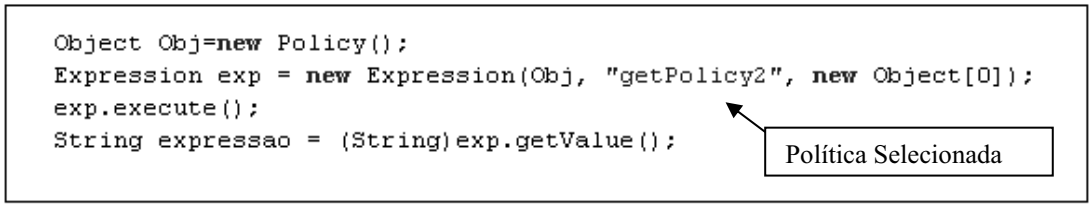


Figura 4.17 – Mecanismo para seleção de políticas utilizadas pelo módulo seletor.

A descrição dos agentes é efetuada por meio de autômatos e transformada em expressões regulares para a programação em Java. Uma expressão regular é um método formal que pode ser usado para criar uma definição e, em seguida, para checar se um texto qualquer segue a expressão definida. Java disponibiliza API para manipulação de expressões regulares por meio do pacote *java.util.regex*. Isso facilita o desenvolvimento de mecanismos (por exemplo, *parsers*) para verificação e validação dos autômatos que representam os agentes. A Figura 4.18 ilustra a implementação de dois agentes por meio de duas expressões regulares descritas como “policy1” e “policy2”. Os atributos “entPolicy1” e “entPolicy2” recebem os valores que definem as mudanças de estados. Neste caso, representados pelos símbolos “ECG1”, “ECG3”, “ECG6” e “ECG12”.

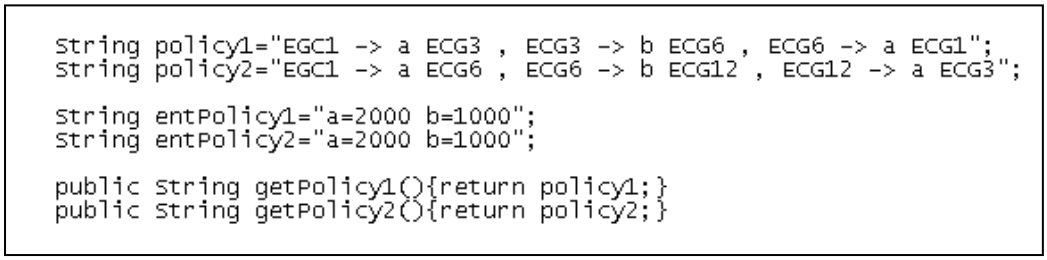


Figura 4.18 – Descrição dos agentes por meio de expressões regulares.

4.3.2 – O *wrapper*

O gerador de agente recebe um programa de usuário (arquivo XML) por meio de uma mensagem SOAP. A partir daí, gera o programa que será compilado e instalado nos sensores. A compilação e a instalação desse programa requerem a utilização de ferramentas específicas do hardware em uso. Para isso, o BWSNET *Proxy* aciona o componente *Wrapper* (apresentado na Seção 3.6). Uma das funcionalidades do *Wrapper* é encapsular os comandos necessários para compilação, instalação e inicialização dos programas nos nós sensores. Esse encapsulamento possibilita utilizar os demais componentes do sistema, desenvolvidos em Java, para acionar comandos necessários para programação e configuração dos sensores.

Na prática, o *Wrapper* executa a interface entre o BWSNET *Proxy* com ferramentas desenvolvidas para um hardware específico. A compilação, instalação e inicialização do programa gerado pelo gerador de agente finalizam a programação do nó sensor. Esse processo foi ilustrado pela Figura 1.3 e pela Figura 3.14. A Figura 4.19 mostra um trecho do código-fonte utilizado pelo *Wrapper* para comandar a instalação (*deployment*) do programa contido no arquivo "agente.out" no sensor de ECG.

```
public boolean deploy() throws IOException
{
boolean resp = true;
String Command = null;
Runtime runtime = Runtime.getRuntime();
Command = "deploy.bat";

Process proc = runtime.exec(Command);

InputStream inputstream = proc.getInputStream();
InputStreamReader inputstreamreader =  new InputStreamReader(inputstream);
BufferedReader bufferedreader = new BufferedReader(inputstreamreader);

String line;
 while ((line = bufferedreader.readLine()) != null) {System.out.println(line);}

     try {
         if (proc.waitFor() != 0)
         {
          if (proc.exitValue() != 0){resp = false;}
         }

         } catch (InterruptedException e) {System.err.println(e);}

return resp;

}
```

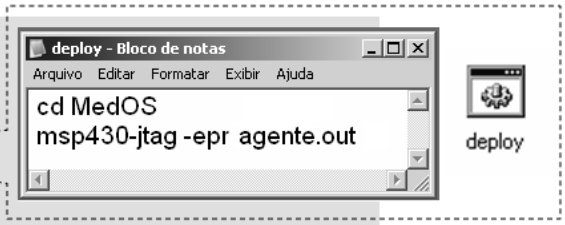


Figura 4.19 – Detalhes de implementação do componente *Wrapper*.

Um outra funcionalidade do *Wrapper* é o envio de comandos do Tradutor MedOS (apresentado na Seção 3.6) para os sensores. A Figura 4.20 ilustra o envio de um comando para reinício do nó sensor por meio da porta serial do PC. Para isso, foi utilizado pacote *javax.com*. Esse pacote contém APIs que facilitam o desenvolvimento de mecanismos que acessam os dispositivos de entrada e saída, por exemplo, a porta serial do PC.

```
import java.io.*;
import java.util.*;
import javax.comm.CommPortIdentifier;
import javax.comm.SerialPort;
import javax.comm.PortInUseException;
import javax.comm.UnsupportedCommOperationException;
...
SerialConnection connection = new SerialConnection();
boolean aberto=false;
String Mensagem = "r"; //mensagem r = MedOS restarted

 if(aberto == false)
               {connection = new SerialConnection();
                connection.openConnection("COM3");
                aberto=true;}

     connection.Escreve(Mensagem);
     System.out.println("Comando enviado ");
 ...
```

Figura 4.20 – Envio de um comando para o sensor por meio do componente *Wrapper*.

4.4 – CAMADA DE SERVIÇOS DO HARDWARE: MEDOS

A Figura 4.21 mostra os principais artefatos do MedOS e as dependências. O MedOS foi implementado sobre o FreeRTOS, apresentado na Seção 3.7.1. A Figura 4.21 ilustra como o artefato "agente.c" (criado pelo gerador de agente) relaciona com os demais artefatos do MedOS. O artefato "MedOSConfig.h" é um arquivo de configuração utilizado pelo pré-compilador para seleção e configuração de funcionalidades implementadas pelo FreeRTOS. Por exemplo, o parâmetro "INCLUDE_vTaskDelay=1" indica que a biblioteca de código que implementa a funcionalidade "vTaskDelay" deve ser incluída durante a compilação do sistema. Essa funcionalidade possibilita que tarefas possam ser suspensas por um período definido de tempo. O artefato "MedOS.h" encapsula (unifica) as bibliotecas de código utilizadas para programação do agente por meio de uma única diretiva de compilação. Isso possibilita simplificar as estruturas de decisão que implementam o gerador de agente, apresentado na Seção 4.3.1.

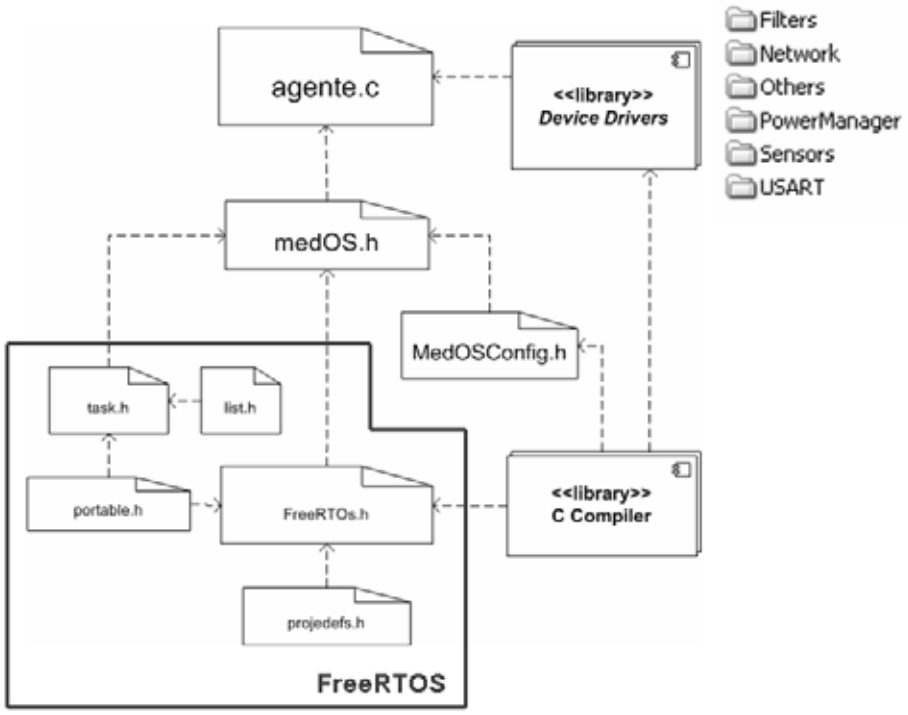


Figura 4.21 – MedOS: artefatos e suas dependências

O MedOS utiliza as bibliotecas de código do MSPGCC (MSPCC, 2007). O MSPGCC é uma IDE (em português, Ambiente de Desenvolvimento Integrado) para programação de

microcontroladores da família MSP430. É de código aberto e uso gratuito. Essa ferramenta é composta basicamente por: um compilador C (compatível com o padrão ANSI C89) com suporte à sintaxe orientada a objetos (C++), uma ferramenta para verificação do código gerado (um *debugger*) e uma ferramenta para instalação e inicialização dos programas no microcontrolador.

4.4.1 – *Device Drivers*

Cumpre lembrar que a principal funcionalidade do MedOS é facilitar a geração e suportar a execução de agentes. Então, o MedOS disponibiliza bibliotecas de código de alto nível (em linguagem C) que facilitam a utilização dos componentes da arquitetura do nó sensor em função de uma aplicação. Por exemplo, a Figura 4.22 mostra um trecho de código que implementa uma parte do *device driver* para captura do sinal de ECG. Este programa configura e aciona o conversor A/D presente no microcontrolador para uma aplicação específica. Dessa forma, o gerador de agente não precisa lidar com instruções especificas do hardware. Isso aumenta a portabilidade (independência do hardware) e diminui a complexidade das estruturas de decisão que implementam o gerador de agente.

```
#include <MedOSConfig.h>
#if (INCLUDE_ecg == 1)

getECG( int FA)
 {
  if (FA == 600)
      { static  ecg;
        ADC12CTL0 &= ~ENC;
        P6SEL |=0x01;
        ADC12CTL0 =  ADC12ON + SHT0_15 + REFON + REF2_5V;
        ADC12CTL1 =  SHP + SHS_0 + ADC12DIV_7 + ADC12SSEL_0 + CONSEQ_0;
        ADC12MCTL0 = INCH_0 + SREF_0;
        ADC12CTL0 |= ENC + ADC12SC;
        while (ADC12CTL0 & ADC12SC);
        ecg =  ADC12MEM0 * 0.62515;
      return ecg;}
  ...
}

#endif
```

Figura 4.22 – Um exemplo de *device driver*.

Outro conjunto de *device drivers* implemementado pelo MedOS refere-se ao gerenciamento do consumo de energia em microcontroladores MSP430. Esse conjunto de funcionalidades recebeu o nome de *PowerManager*. É implementado pelos artefatos *"powermanager.c"* (arquivo de implementação) e *"powermanager.h"* (arquivo de

cabeçalhos e definições). Ambos são encontrados no subdiretório "*PowerManager*" da pasta "*DeviceDrivers*" (Figura 4.21).

O *PowerManager* foi desenvolvido com objetivo de oferecer uma API de alto nível para que o gerador de agente pudesse selecionar e aplicar nos agentes as estratégias DPM e DSV (apresentadas na Seção 2.3). Para isso, foram criadas rotinas para chaveamento dos modos de operação e para ajuste da freqüência de operação do microcontrolador MSP430. Essas rotinas foram avaliadas por ensaios em laboratório.

Os microcontroladores MSP430 possuem seis modos diferentes de operação: um modo ativo (AM) e cinco modos de baixo consumo: LPM0, LPM1, LPM2, LPM3 e LPM4. Esses modos de operação são controlados pelos *bits* CPUOFF, OSCOFF, SCG0 e SCG1 do registrador R2. Como já foi mencionado na Seção 3.6.1, o tempo de chaveamento entre os modos de operação é de aproximadamente 6 µs. A Figura 4.23 ilustra como *PowerManager* possibilita simplificar a programação dos agentes.

Programação tradicional	**usando a API *PowerManager***

`//habilita interrupção`
`_EINT();`
`//seta o bit CPUOFF no registrador R2`
`BIS_SR(CPUOFF);`
	`enter_modePM (LPM0);`

Figura 4.23 – Programação do modo de operação LPM0 pela API *PowerManager*.

Uma simples redução da freqüência de operação pode reduzir também o consumo de energia porque em alguns circuitos integrados o consumo é diretamente proporcional ao quadrado da tensão e varia linearmente com a freqüência. A Figura 4.24 exibe um gráfico (fornecido pelo fabricante) que relaciona a freqüência de operação com a tensão de alimentação do microcontrolador MSP430. Existe ainda uma outra relação que determina o consumo de energia em função da drenagem de corrente. A Equação 4.1 apresenta essa relação. O MSP430 é capaz de obter ganhos em economia de energia mesmo sem o ajuste dinâmico da tensão porque o consumo de corrente (no modo ativo) varia linearmente em função da freqüência de operação utilizada.

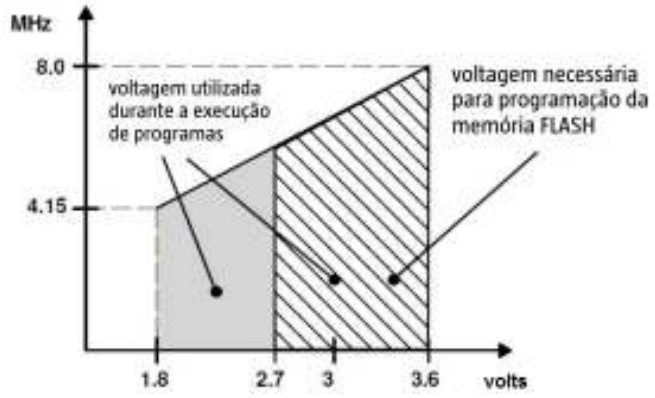


Figura 4.24 – Freqüência de operação *versus* voltagem de alimentação para o MSP430 (modificado – TI, 2006).

Cons. corrente (modo ativo) = Cons. corrente [1MHz] x freq. de operação [MHz] (4.1)

PowerManager disponibiliza duas interfaces de programação para ajuste da freqüência de operação. A interface *frequency_control* possibilita o ajuste baseado numa tabela de valores de freqüência (Tabela 4.1). Essa interface possui dois parâmetros DCOx e RSELx que controlam a freqüência de operação do DCO (sigla em inglês, *Digitally Controlled Oscillator*). O DCO é um oscilador implementado internamente no MSP430 e pode ser gerenciado por comandos no software. Esse ajuste de freqüência tem como vantagem a resposta rápida.

Tabela 4.1 – Seleção da freqüência de operação do MSP com base na seleção de valores para o DCO (PEREIRA, 2005, p. 119).

RSELx	**DCOx**							
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	88,8 kHz	97,3 kHz	106,4 kHz	116,9 kHz	128,7 kHz	141,9 kHz	156,5 kHz	176,8 kHz
1	140,9 kHz	154,5 kHz	168,9 kHz	185,4 kHz	203,8 kHz	224,6 kHz	247,3 kHz	279,7 kHz
2	221,2 kHz	242,3 kHz	264,8 kHz	290,9 kHz	319,7 kHz	352,3 kHz	387,5 kHz	439,1 kHz
3	370,3 kHz	405,3 kHz	442,9 kHz	486,5 kHz	534,8 kHz	589,8 kHz	649,5 kHz	736,9 kHz
4	625,7 kHz	685,1 kHz	748,9 kHz	823 kHz	904,7 kHz	998,9 kHz	1,1 MHz	1,25 MHz
5	1,04 MHz	1,13 MHz	1,24 MHz	1,36 MHz	1,49 MHz	1,66 MHz	1,83 MHz	2,08 MHz
6	1,64 MHz	1,79 MHz	1,96 MHz	2,16 MHz	2,38 MHz	2,64 MHz	2,92 MHz	3,31 MHz
7	2,56 MHz	2,79 MHz	3,06 MHz	3,37 MHz	3,72 MHz	4,13 MHz	4,59 MHz	5,19 MHz

Outra interface (chamada *frequency_fine*) possibilita o ajuste fino da freqüência de operação. Essa interface possui um parâmetro para definir o valor da freqüência desejada em Hertz. Essa interface implementa a política para ajuste da freqüência apresentada por

(PEREIRA, 2005, p. 352). Essa política parte da menor freqüência de operação do microcontrolador e vai incrementando até chegar à freqüência desejada para funcionamento. Na prática, uma função chamada *clock* executa os ajustes nos registradores DCOCTL e BCSCTL1 e configura o *timer* A para gerar interrupções. A rotina que trata interrupções compara o valor da freqüência desejada com o valor atual da freqüência do microcontrolador. Caso não sejam iguais, executa a função *up_DCO* que incrementa o campo MODx no registrador DCOCTL. Esse ciclo se repete até alcançar a freqüência desejada. A Figura 4.25 ilustra a implementação de *up_DCO* por meio de um trecho do código-fonte.

```
unsigned char up_DCO(void)
{
unsigned char result = 1;
   if (ajusteMODx<31) ajusteMODx++; else
      {ajusteMODx = 0;
       if (ajusteDCOx<224) ajusteDCOx+=32; else
          {ajusteDCOx = 0;
             if (ajusteRSELx<7) ajusteRSELx++;
                  else result = 0;
          }
       }
       DCOCTL = ajusteDCOx + ajusteMODx;
       BCSCTL1 = ajusteRSELx;

return (result);
```

Figura 4.25 – Trecho de código que implementa *up_DCO*.

4.4.2 – Interpretador de comandos

O interpretador de comandos é o mecanismo que possibilita modificar a execução do agente instalado no sensor sem a necessidade de reiniciar o hardware. Esse processo de modificação é interativo, ou seja, os resultados das operações executadas sobre os agentes podem ser imediatamente enviados dos sensores para os usuários. Isso viabiliza a configuração do sistema no âmbito dos sensores.

O interpretador de comandos é implementado como uma tarefa de maior prioridade gerenciada pelo MedOS. É criado automaticamente pelo gerador de agente e incluído no código que implementa o agente. Entretanto, caso o usuário prefira, o interpretador de comandos pode não ser incluído durante a programação dos sensores. Dessa forma, os sensores não podem ser reconfigurados. Assim, qualquer modificação no agente implicará numa nova programação do sensor.

Os comandos de usuário para configuração dos sensores são enviados para BWSNET *Proxy* e, em seguida, são repassados para os sensores. A cada comando enviado pelo BWSNET *Proxy*, o interpretador de comandos deve reconhecer a validade desse comando. Para isso, utiliza uma listagem de comandos permitidos para o agente que está instalado no sensor. Essa listagem de comandos é chamada de protocolo MedOS. Cada comando implica numa ação. Em geral, as ações estão relacionadas ao ajuste das tarefas que implementam os estados dos agentes. A Figura 4.26 ilustra uma instância do interpretador de comandos para um nó sensor de ECG.

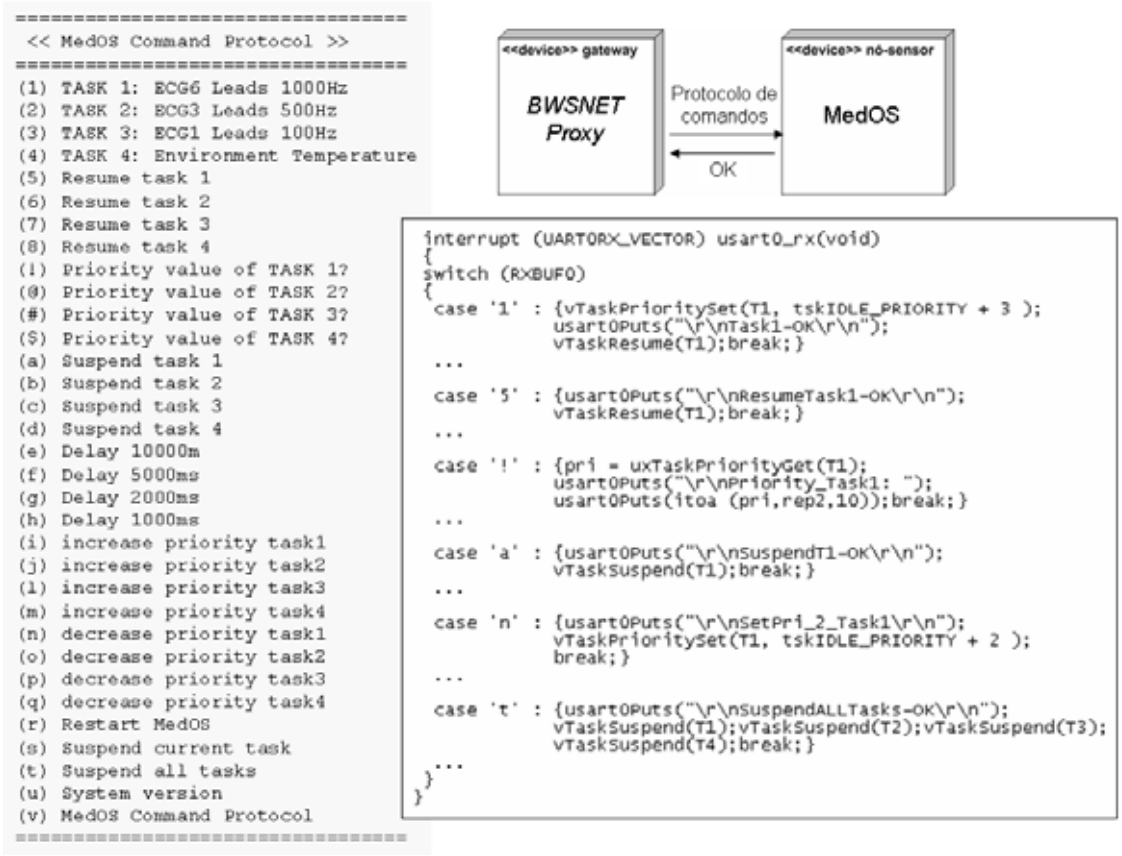


Figura 4.26 - Uma instância do interpretador de comandos para um sensor de ECG.

Para cada mensagem enviada ao sensor são executadas uma ação e uma mensagem de confirmação. As mensagens de confirmação são utilizadas para detecção de falhas. O significado de cada ação pode ser visto pela listagem dos comandos na Figura 4.26 (à esquerda). Uma ação pode ser composta por um ou vários comandos. Os comandos são programados como chamadas ao sistema. A implementação de algumas ações pode ser vista na Figura 4.26 (à direita), num trecho do código-fonte em linguagem C.

4.5 – O HARDWARE

Existem alguns sistemas vestíveis (hardware) disponíveis no mercado para aplicações clínicas. Por exemplo, monitores de pressão arterial, marcadores de freqüência cardíaca, pedômetros e outros. Entretanto, esses sistemas não possibilitam alterações no software já instalado. Diante desse fato, optou-se pela implementação do hardware para as provas de conceito propostas nesta Tese.

4.5.1 – O módulo de comunicação por rádio freqüência

O Bluetooth foi a tecnologia escolhida para implementar o rádio transmissor que compõe a arquitetura do nó sensor apresentada na Figura 3.20. Os critérios utilizados para escolha dessa tecnologia foram expostos na Seção 3.3. Para implementar o rádio transmissor foi adquirido o módulo BlueSMiRF v1.0 (SP, 2006). O BlueSMiRF v1.0 possibilita implementar um canal de comunicação *full-duplex* entre o nó sensor e qualquer outro dispositivo que tenha uma interface de rede Bluetooth. As principais características técnicas do BlueSMiRF são apresentadas na Tabela 4.2.

Tabela 4.2 – Características do BlueSMiRF.

Características	BlueSMiRF v1.0
Freqüência de operação	2,4 GHz ISM
Modulação	GFSK (*Gaussian Frequency Shift Keying*)
Alcance	Até 100 metros
Taxas de transmissão	2400-115200 bps
Largura de banda	200 Kbps (*fast mode*) 60 Kbps (*regular data mode*)
Potência de transmissão	-9 dBm ~ 15 dBm (configurável)
Tensão de alimentação	3-6 volts
Consumo de corrente: inicialização (conexão)	48 mA
Consumo de corrente: operação normal	33 mA (enviando uma seqüência constante) 24,8 mA (aguardando)
Consumo de corrente: *Stand-by*	2 mA (reinicialização por comandos) 3 mA (reinicialização automática)
Latência de dados	2~20 ms
Dimensões físicas	(43x15mm) 2g

Na prática, o BlueSMiRF é um modem concebido com base em um núcleo SoC (*System-on-a-Chip*). Esse núcleo implementa grande parte da pilha de protocolos Bluetooth no próprio hardware. Isso inclui serviços como autenticação, nomeação, descoberta e roteamento. O principal benefício dessa implementação é a liberação de recursos do

microcontrolador. Por exemplo, possibilita que a memória e o processador embutidos no microcontrolador sejam utilizados apenas pela aplicação corrente.

Para gerência do dispositivo, o BlueSMiRF disponibiliza um conjunto de comandos, com sintaxe muito parecida aos antigos comandos Hayes. Isso facilita a manutenção e o monitoramento do canal de comunicação. Por exemplo, o comando "ATSPF,15,+<CR>" possibilita ajustar a potência de transmissão do rádio para 15 dBm (equivalente a 32 mW). O comando "ATLSTO,10<CR>" ordena que o rádio descarte um pedido de conexão se o início da conexão não acontecer em até 10 segundos.

4.5.2 – Microcontroladores utilizados

Inicialmente, os nós sensores foram construídos com base no *kit* Olimex MSP-P149 (OLIMEX, 2006). O microcontrolador MSP430F149 (TI, 2006) é o núcleo desse *kit.* As principais características técnicas do microcontrolador MSP430F149 são apresentadas na Seção 2.1 e na Tabela 2.1. A Figura 4.27 mostra o protótipo do nó sensor implementado com base no MSP-P149.

O MSP-P149 foi escolhido pelo baixo custo (cerca de US$ 32,00) e por possuir dimensões físicas reduzidas (100x80 mm). Portanto, pouco obstrutivas. Isso é importante para a construção de sistemas vestíveis. Ainda, possibilita a inserção dos módulos que implementam os demais componentes da arquitetura numa mesma placa de circuito impresso. Além disso, o MSP-P149 implementa: a interface JTAG (em inglês, *Joint Test Action Group*) para instalação (*deployment*) do software; um circuito condicionador de tensão e a interface RS-232, para, caso seja necessário, a interconexão por fios.

Além da plataforma baseada no MSP430, outra plataforma foi desenvolvida. Essa baseia-se no microcontrolador PIC18F2550. As principais características técnicas desse microcontrolador são apresentadas na Tabela 2.1. Se comparado ao MSP-P149, o PIC18F2550 custa aproximadamente 75% menos. Também, possui interface USB (*Universal Serial Bus*) embutida no microcontrolador. Isso possibilita substituir a interface RS-232 por outra mais eficiente e com dimensões físicas menores. Outra característica interessante do PIC é que esse circuito integrado pode ser encontrado no encapsulamento DIP (*Dual In-line Package*). Isso possibilita a execução de ensaios preliminares em *pront-*

o-board, portanto, sem a necessidade de adquirir um *kit* específico para testes. Entretanto, o consumo de energia do PIC18F2550 (operando no modo ativo) é até sete vezes maior que o do MSP430F149. A Figura 4.28 ilustra uma implementação do nó sensor com base no PIC 18F2550.

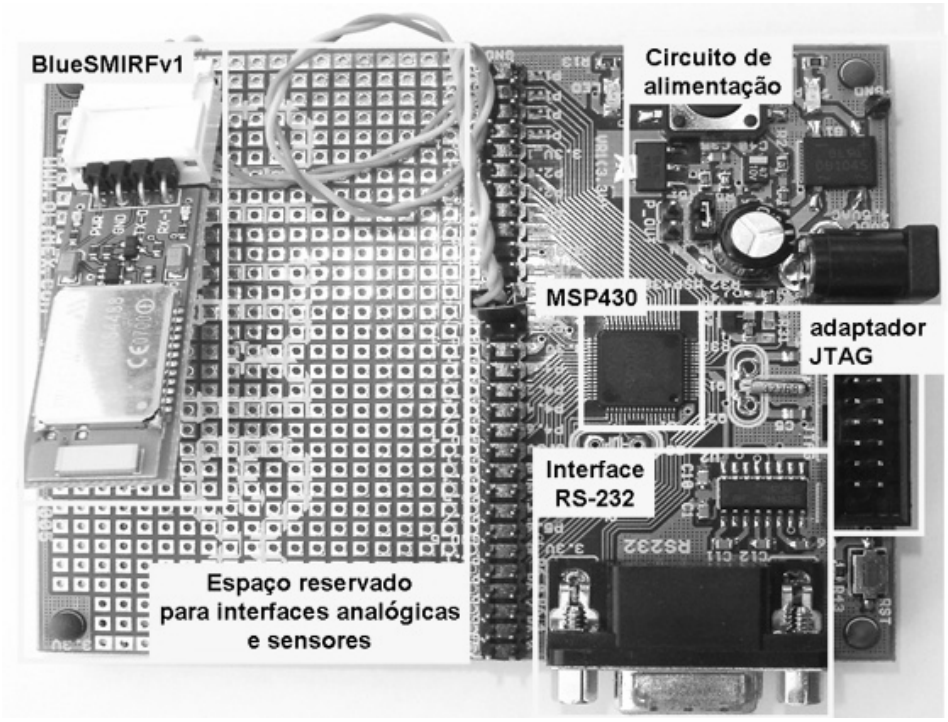


Figura 4.27 – Implementação do nó sensor baseada no *kit* MSP-P149.

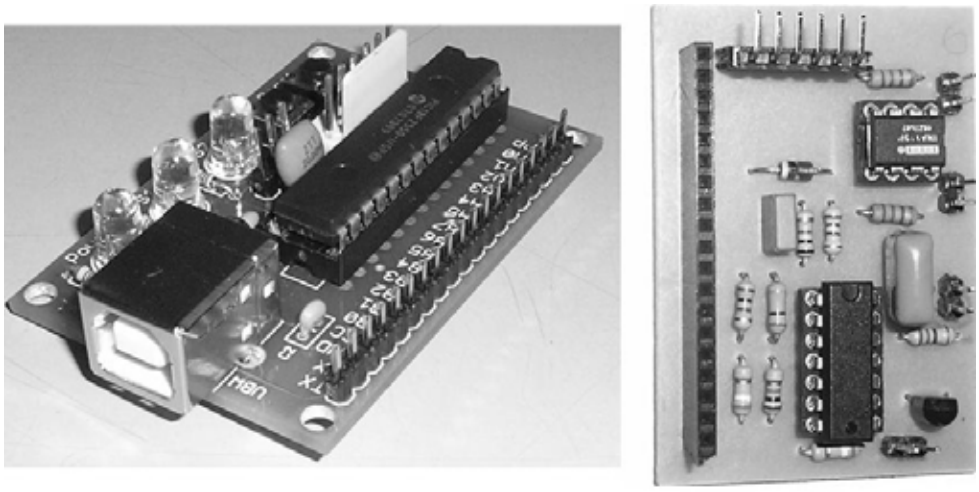


Figura 4.28 – Implementação do nó sensor baseada no PIC18F2550.

4.5.3 – As interfaces e circuitos analógicos implementados

Neste trabalho foram implementados circuitos eletrônicos para o monitoramento do eletrocardiograma (ECG), do eletromiograma não-invasivo (EMG), da temperatura cutânea (TC), da resistência galvânica da pele (GSR) e da pressão arterial (PA). Alguns dos protótipos desenvolvidos foram construídos com componentes SMD (*Surface Mount Devices*). Essa tecnologia possibilita redução do tamanho e do peso desses circuitos. A Figura 4.29 exibe um dos protótipos desenvolvido com tecnologia SMD para aquisição dos sinais: ECG, EMG, TC e GSR. Esse protótipo possibilita a implementação de um nó sensor multifuncional.

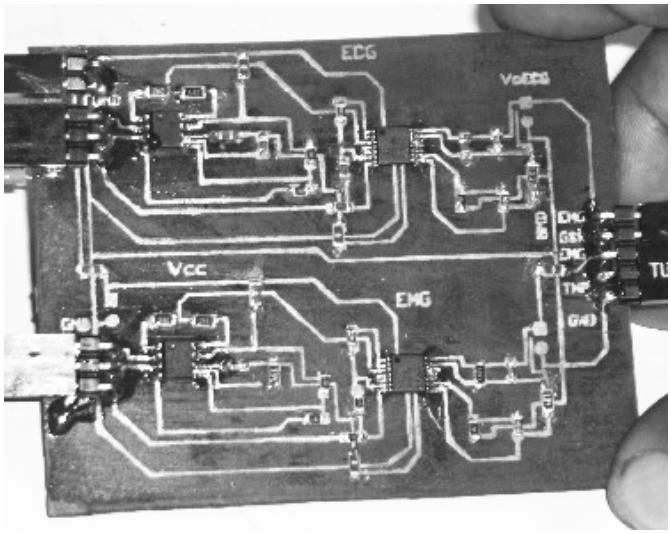


Figura 4.29 – Protótipo em placa de circuito impresso (GUTIÉRREZ *et al*., 2006).

Durante o desenvolvimento dos circuitos analógicos alguns componentes eletrônicos foram avaliados com propósito de reduzir custos. Ao final, foram implementados outros circuitos para captura do ECG e da pressão arterial com base nos amplificadores operacionais TL064 e OPA2336 da Texas Instruments. A Figura 4.28 (à direita) ilustra um circuito para aquisição do ECG desenvolvido com base no TL064. Para utilização do TL064 foi necessária a inclusão do TLE2425. O TLE2425 é um circuito terra virtual implementado por um circuito integrado. Nos testes efetuados em laboratório, esse componente ofereceu melhorias significativas em relação ao uso do próprio TL064 para o circuito terra virtual. Essas melhorias foram relativas ao tempo de resposta e estabilidade no funcionamento do circuito. A Figura 4.30 ilustra o protótipo desenvolvido para aquisição do sinal da pressão

arterial com base no amplificador operacional OPA2336 (à direita) e no sensor MPX5050GP (à esquerda).

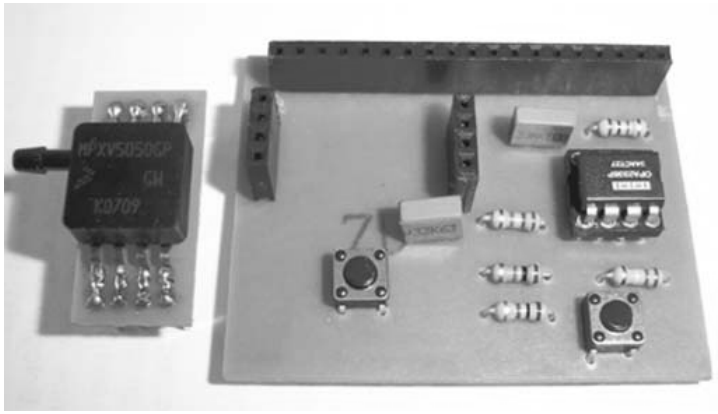


Figura 4.30 – Protótipo desenvolvido para captura da pressão arterial.

Como resultados dos circuitos mostrados na Figura 4.29 são apresentadas imagens das formas de onda obtidas pelo osciloscópio digital. Todos os sinais foram obtidos de maneira não invasiva por meio de eletrodos conectados a um voluntário.

A Figura 4.31 ilustra uma amostra do sinal da resistência galvânica da pele (GSR) capturada de um voluntário em situação de relaxamento. Neste exemplo, a resistência mostrada é aproximadamente 342 kΩ (*KiloOhms*). Cumpre lembrar que a variabilidade desse sinal no transcorrer do tempo pode indicar, por exemplo, o nível de estresse em que uma pessoa se encontra. Ensaios repetidos comprovaram a variação desse sinal.

A Figura 4.32 exibe o sinal que representa a temperatura cutânea (TC). O valor obtido corresponde a um valor de tensão sobre o termistor. Por meio da Equação 3.3 e da Equação 3.4 obteve-se o valor da temperatura. De acordo a amplitude do sinal mostrado na Figura 4.32 (1,6140 V), o valor da temperatura é igual a 36,37 °C (Graus Celsius).

A Figura 4.33 mostra a forma de onda obtida para o sinal de EMG. Para obtenção deste sinal foram colocados eletrodos no bíceps do braço esquerdo de um voluntário. Foram utilizados dois eletrodos, separados por dois centímetros. Um terceiro eletrodo (eletrodo de referência) também foi utilizado. Esse foi afixado no pulso direito do voluntário.

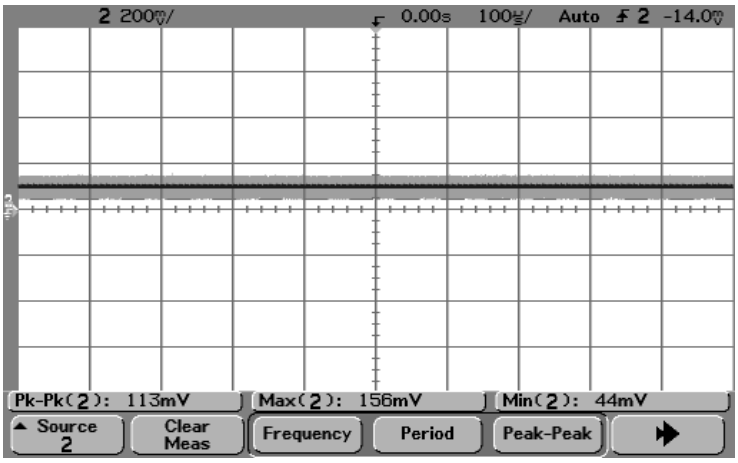


Figura 4.31 – Forma de onda gerada pelo circuito de GSR (GUTIÉRREZ *et al*., 2006)

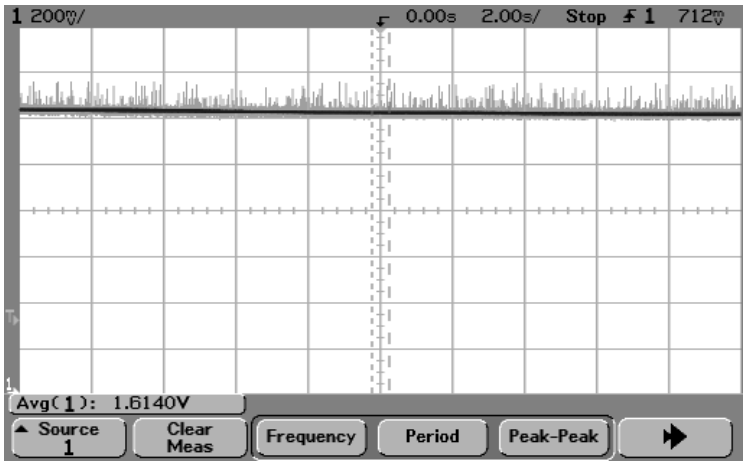


Figura 4.32 – Forma de onda capturada pelo circuito de TC (GUTIÉRREZ *et al*., 2006).

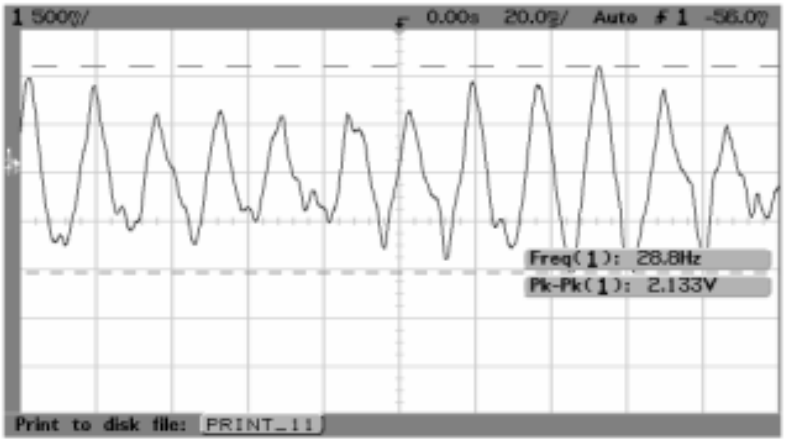


Figura 4.33 – Sinal capturado pelo circuito de EMG (GUTIÉRREZ *et al*., 2006).

A Figura 4.34 mostra a forma de onda obtida do nó sensor de ECG apresentado na Figura 4.28. Nesse exemplo, os sinais foram digitalizados com resolução de 10 bits (resolução máxima do PIC18F2550). A imagem que apresenta a forma de onda foi gerada com base em amostras coletadas de um voluntário (de maneira não invasiva) e plotadas por meio do software Matlab.

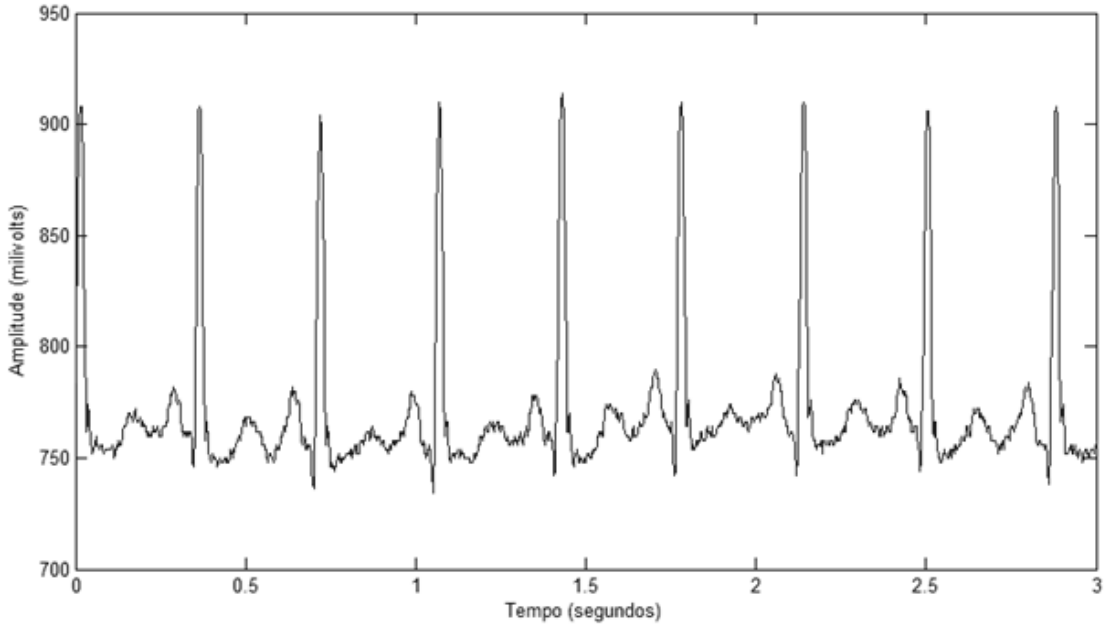


Figura 4.34 – Forma de onda gerada pelo circuito de ECG.

4.6 – TESTES APLICADOS NOS PROTÓTIPOS DA SOAB

A execução de testes de software é uma etapa importante para garantia de qualidade. De acordo com Staa (Staa, 2000, p. 517), o que ocorre é que, de um modo geral, nenhuma técnica de certificação, verificação, validação, aprovação ou teste pode assegurar que o software não possua faltas, ou que o software satisfaça a todos os requisitos de qualidade. No entanto, a escolha cuidadosa de um conjunto de instrumentos de controle auxilia na verificação de um nível de qualidade satisfatório.

Outro aspecto importante refere-se ao desenvolvimento ou aplicação de métodos que possam ser utilizados para testes de novos sistemas. Isso é particularmente importante quando o objeto dos testes é um sistema (software) desenvolvido para provas de conceito. Nesse caso, existe pouco ou nenhum consenso acerca do método que deve ser utilizado e nem de resultados que evidenciam a aplicação de outros métodos e, também, que sirvam

para comparações.

Os testes desenvolvidos e aplicados nos protótipos da arquitetura SOAB tiveram os seguintes objetivos:

- Encontrar erros de programação.
- Avaliar o correto funcionamento dos sistemas;
- Avaliar o desempenho dos sistemas relativo ao consumo de memória e ao tempo de processamento;
- Avaliar o desempenho dos nós sensores relativo à utilização de mecanismos para economia de energia. Esses mecanismos foram descritos na Seção 4.4.1;
- Avaliar o desempenho relativo ao tempo de resposta para efetuar a programação de um sensor à distância pela Internet.

4.6.1 – Testes de caixa aberta

Foram desenvolvidos testes de "caixa aberta (ou caixa branca)" (Staa, 2000, p. 529) para cada novo componente implementado. Esses testes tiveram como propósito identificar possíveis erros de programação, avaliando aspectos como: estruturas condicionais, fluxo de dados e caminhos lógicos. Os relatórios (resultados) desses testes foram incorporados ao código-fonte dos programas como parte dos comentários.

O método empregado nos testes de caixa aberta baseia-se em testes de unidade (ou testes unitários). O teste de unidade testa o menor dos componentes de um sistema de maneira isolada. Cada uma dessas unidades define um conjunto de estímulos (por exemplo, invocação de métodos), e de dados de entrada e saída associados a cada estímulo. Em seguida, são executados os testes.

Em geral, são utilizadas ferramentas para automatizar os testes unitários. Neste trabalho foi utilizado o Junit [25] para implementação dos testes unitários. O Junit é um *framework* gratuito para desenvolvimento de testes de artefatos que foram escritos em linguagem Java.

[25] Veja http://www.junit.org/

O Junit possibilita inserir assertivas[26] nas classes que implementam os componentes e, assim, executar testes de artefatos. Como exemplo, a Figura 4.35 ilustra um teste aplicado a um artefato que compõe o gerador de agente. Esse teste verificou a capacidade da classe "S_Policy" em retornar um atributo relacionado ao parâmetro utilizado para invocação de um método. Para isso, foi implementada a classe "S_PolicyTest". Nela é inserido o método "assertEquals" que retorna o resultado de uma comparação. Executado o teste, o relatório é apresentado pela tela do Junit.

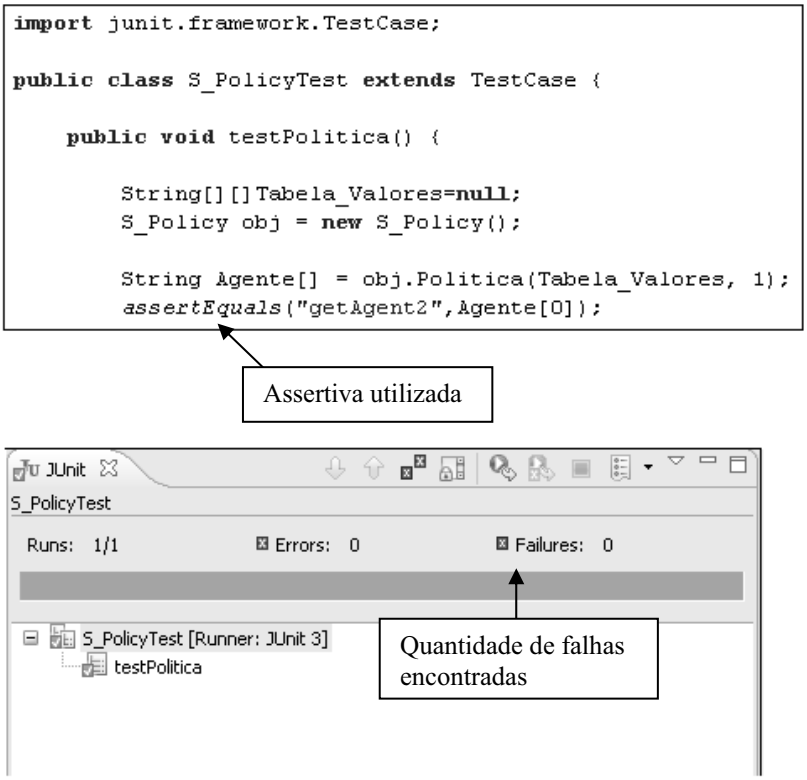


Figura 4.35 – Um exemplo de teste unitário utilizando o Junit.

[26] Assertivas são relações envolvendo os dados e os estados manipulados pelos programas e que devem estar satisfeitas em determinados pontos do programa. São utilizadas para testar programas como forma de verificar automaticamente o funcionamento pelo menos parcial do processamento (Staa, 2000, p. 421).

4.6.2 – Testes de caixa fechada

Quanto à portabilidade, a definição de uma arquitetura em camadas funcionais e independentes facilita a modificabilidade, subparâmetro definido em (ISO, 2001) e apresentado na Tabela 3.1. As tecnologias de desenvolvimento empregadas, como, por exemplo, Java e SOAP trazem como legado a capacidade de promover a independência do hardware e do software. Para avaliar o funcionamento dos sistemas que compõem SOAB foram empregados testes de "caixa fechada (ou caixa preta)" (Staa, 2000, p. 529) em mais de uma plataforma de suporte. Plataforma de suporte refere-se ao hardware e sistema operacional utilizados.

Os testes de caixa fechada enfocam nos resultados que o sistema deve gerar com base num conjunto de entradas, isto é, avalia as funcionalidades propostas na especificação do sistema. Portanto, não há necessidade de conhecer o código que implementa os componentes, ao contrário dos testes de caixa aberta. Dados de entrada são fornecidos, o teste é executado e o resultado obtido é comparado a um resultado esperado. Então, para cada componente ou subsistema é necessário elaborar um conjunto de casos de teste. Casos de teste são situações criadas para exercitar o software sob diferentes condições. Em geral, informam os possíveis valores de entrada do componente, o fluxo de execução e o resultado esperado. A Tabela 4.3 ilustra um exemplo de caso de teste utilizado para testar a tela para adição de um novo sensor, apresentada na Figura 4.2.

Foram executados testes de caixa fechada em versões dos protótipos que implementam SOAB em plataformas Windows e Linux. Os resultados obtidos dos testes foram satisfatórios em relação aos resultados esperados em ambas as plataformas, resguardando a diferença entre os tipos de fontes gráficas disponibilizadas pelo Linux em relação aos tipos disponibilizados pelo Windows. Outros aspectos avaliados durante a execução desses testes foram:

- Interoperabilidade entre os quatro principais sistemas que implementam as camadas da SOAB. Isso foi implementado por meio de casos de testes que enfocaram a interação entre essas camadas;
- Facilidade de modificação de elementos de cada camada em função da independência entre as mesmas. Isso foi implementado por meio de casos de testes que enfocaram a

manutenção (modificação e/ou substituição) de componentes. Os resultados desses aspectos foram também satisfatórios em relação ao esperado.

Tabela 4.3 – Exemplo de caso de teste.

Identificação		Tela para adição de um novo sensor
Itens a testar		Processamento correto da inserção de um novo sensor em BWSNET *Conf. Tool*
Entradas	Campo	Valor
	Nome do arquivo	<diretório> novosensor.exe
	Nome	novo sensor
Saídas Esperadas	Campo	Valor
	nome	Novo sensor
	tipo	Arquivo executável do windows
Ambiente		Linux
Procedimentos		Preencher campos de entrada e acionar o botão “adicionar”
Dependências		nenhuma

4.6.3 – Testes de desempenho

Eficiência é sinônimo de economia de recursos e/ou de tempo (IEEE, 1993; ISO, 2001). Para avaliar esse parâmetro foram aplicados testes de desempenho. Esses testes foram aplicados com propósito quantificar:

- A economia de energia relativa à utilização do nó sensor em diferentes modos de operação;
- O tempo de execução e o consumo de memória dos componentes que implementam os sistemas da arquitetura SOAB;
- O tempo de resposta para efetuar a programação dos sensores à distância pela Internet.

4.6.3.1– Avaliação do consumo de energia do nó sensor

Foram realizados testes com a biblioteca *PowerManager* (apresentada na Seção 4.4.1) com objetivo de verificar e avaliar as mudanças nos modos e na freqüência de operação do microcontrolador MSP430 e, então, quantificar o consumo de energia. Cumpre lembrar que essa biblioteca de funções compõe os *device drivers* que fazem parte do MedOS, apresentado na Seção 4.4. O MedOS implementa uma das camadas da arquitetura SOAB. Para execução desses testes foi projetado e montado um circuito de testes. A Figura 4.36 ilustra o circuito e o protótipo utilizados nos testes.

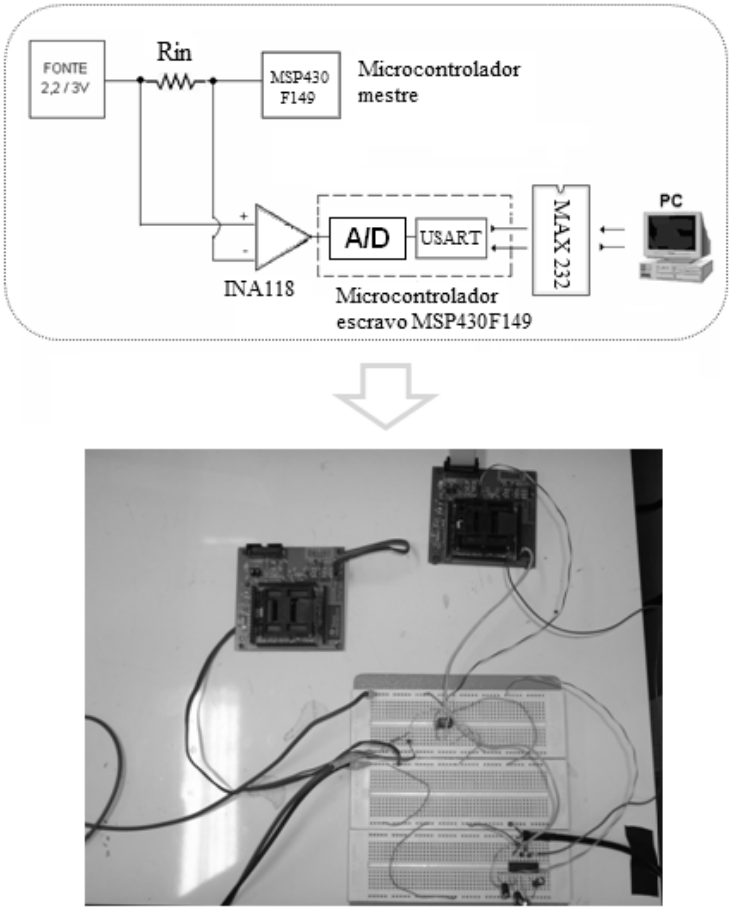


Figura 4.36 – Circuito de testes e protótipo utilizado para avaliar *PowerManager*.

Na prática, o objetivo desses experimentos foi medir a corrente elétrica drenada pelo microcontrolador MSP430F149 (microcontrolador mestre) quando ele estivesse operando em diferentes modos de operação e em diferentes freqüências de operação. Para isso, foi utilizando um resistor (R_{in}) em série com o MSP430. Isso possibilitou quantificar a queda de tensão no resistor e, assim, aplicar a Lei de Ohm para obtenção dos valores da corrente elétrica. Foram usados diferentes valores para R_{in} visto que a impedância do microcontrolador é diferente em cada modo de operação. Também, foi considerado que os valores utilizados para R_{in} deveriam ser pequenos se comparados às impedâncias do microcontrolador. Essa prática impediu que o valor da resistência de R_{in} interferisse na impedância total do circuito. A Figura 4.37 (à esquerda) ilustra, por meio de um gráfico, os principais resultados obtidos. Também, a mesma figura (à direita) ilustra como o agente apresentado na Figura 3.15b pode ser estendido pelo uso de *PowerManager* para economia de energia (*CPU Power-Saving Modes*).

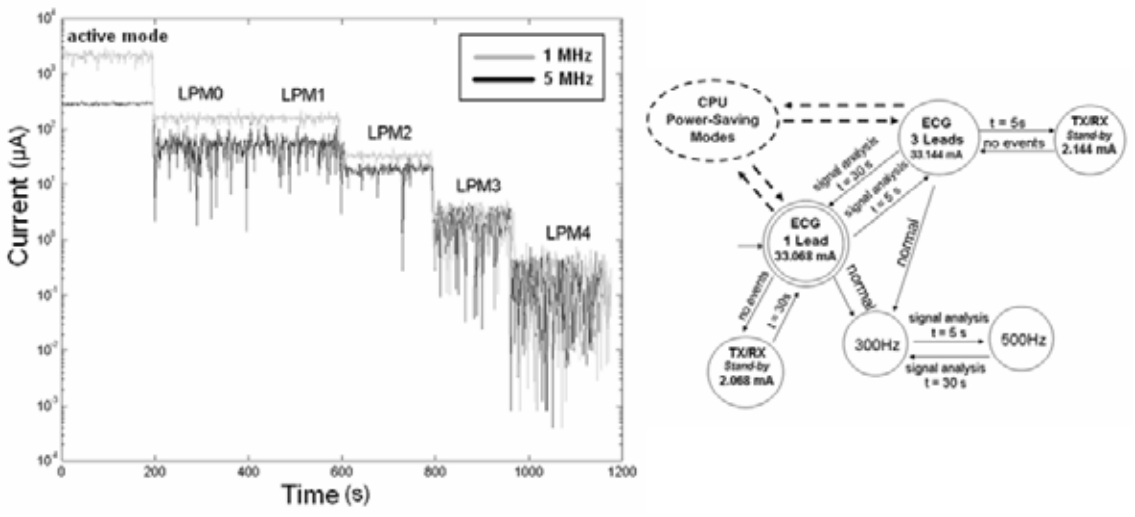


Figura 4.37 – Desempenho e aplicação de *PowerManager* (Barbosa *et al*, 2007b).

4.6.3.2 – Avaliação do tempo de processamento e do consumo de memória.

Para avaliação do tempo de processamento e do consumo de memória dos componentes de software foram efetuados testes com uma ferramenta para esboço do perfil da aplicação (em inglês, *profiling*). A ferramenta utilizada é chamada TPTP (sigla em inglês, *Test and Performace Tool Plataform*). Essa ferramenta é integrada ao Eclipse [27] e fornece relatórios com informações sobre a utilização dos recursos da plataforma de suporte, como, por exemplo, memória e CPU.

Para realizar os testes foram selecionados dois tipos de perfil (*profiling*): *Basic Memory Analysis* e *Execution Time Analysis*. A Figura 4.38 ilustra o relatório gerado para o consumo de memória do gerador de agente, apresentado na Seção 4.3.1. Os valores apresentados na Figura 4.38 e na Figura 4.39 foram obtidos com base na execução de TPTP sobre plataforma de suporte composta por microcomputador PC AMD Athlon 2800+ com 1024 MB (Megabytes) de memória RAM e o sistema operacional Windows (versão XP Profissional).

O perfil *Basic Memory Analysis* possibilita a geração de relatórios acerca do consumo de memória por atributo, método e classe. Da Figura 4.38, a coluna "Live Instances" indica a

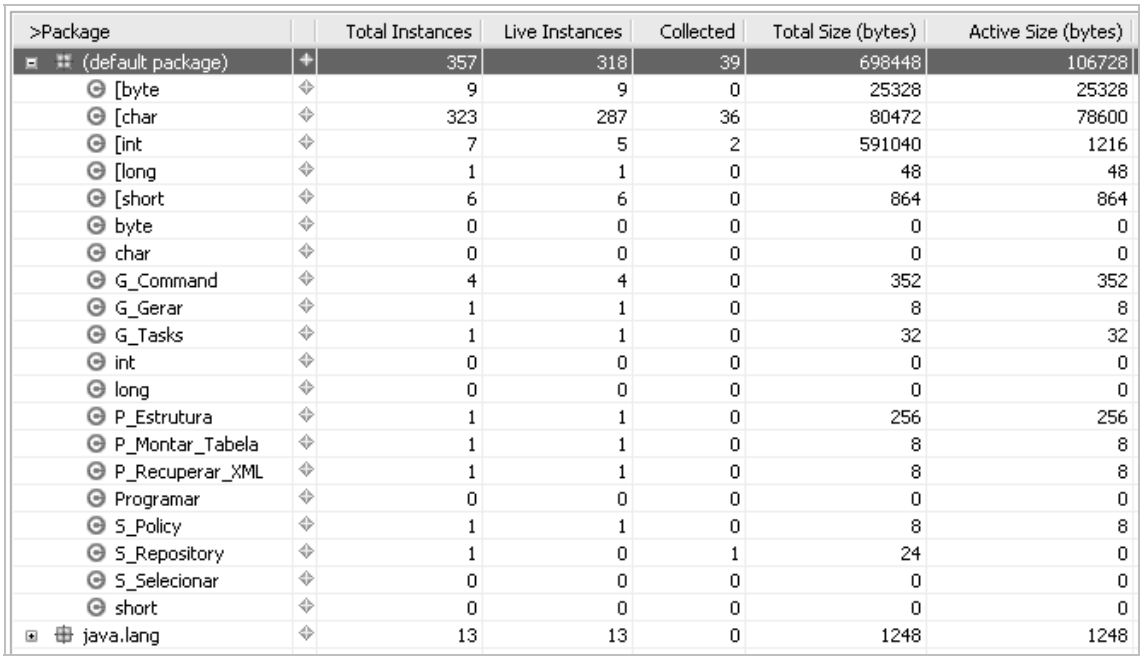
[27] O Eclipse é uma ferramenta para o desenvolvimento de aplicações, disponível gratuitamente em: http://www.eclipse.org/.

quantidade de instâncias criadas com base num determinado pacote, classe ou método, quando o coletor de lixo (*garbage collector*) não está ativado. O coletor de lixo é um programa executado pela máquina virtual Java para remover da memória rastros de objetos que não possuem mais referência. A coluna "Total Instances" indica a quantidade total de instâncias criadas durante a execução de um determinado pacote, classe ou método. A coluna "Total Size" indica a quantidade de memória (em bytes) ocupada por um pacote, classe ou método, considerando todas as instâncias criadas, incluindo aquelas que foram removidas pelo coletor de lixo. A coluna "Active Size" indica o total de memória (em bytes) de todas as instâncias ativas.

O perfil *Execution Time Analysis* possibilita a geração de relatórios acerca do tempo de execução. A Figura 4.39 apresenta também resultados gerados para a execução do gerador de agente. A coluna "Base Time" indica o tempo gasto (em segundos) para executar a invocação de um método em particular, excluindo o tempo gasto para invocação de outros métodos que podem ser acionados pelo método que está sendo avaliado. A coluna "Cumulative Time" indica o tempo acumulado (em segundos), isto é, o tempo para executar todos métodos que podem ser acionados pelo método que está sendo avaliado. Se uma invocação de método não aciona outros métodos a columa "Cumulative Time" terá os mesmos valores da coluna "Base Time". Por fim, a coluna "Calls" indica o número de chamadas executadas pelo método selecionado. A coluna "Avarage Base Time" indica o tempo médio (em segundos) por chamada. Corresponde ao valor da coluna "Base time" dividido pelo número de chamadas (*calls*).

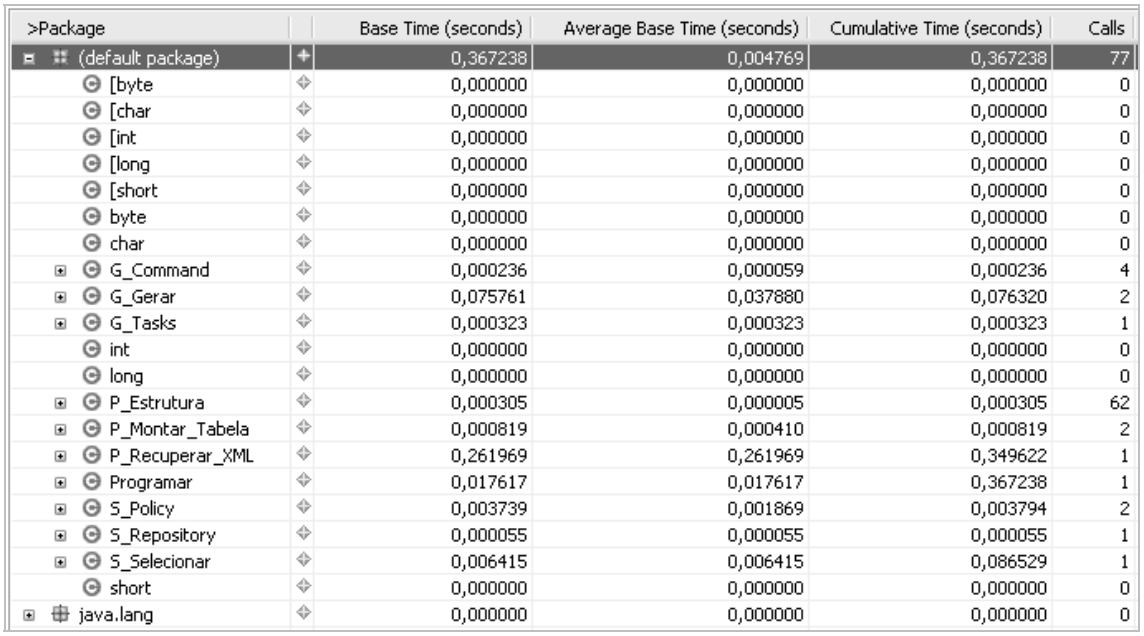
4.6.3.3 – Avaliação do tempo de resposta

Para avaliar o desempenho acerca da programação dos sensores à distância (pela Internet) foi executado um teste para obtenção de valores relacionados ao tempo de reposta. O tempo de resposta é uma métrica que pode estar relacionada à satisfação dos usuários em relação às intervenções executadas pela Internet. Por exemplo, de acordo com (BOUCH, 2001), valores acima de 10 segundos poderiam causar desinteresse quanto ao serviço por parte da maioria dos usuários do comércio eletrônico.



>Package	Total Instances	Live Instances	Collected	Total Size (bytes)	Active Size (bytes)
(default package)	357	318	39	698448	106728
[byte	9	9	0	25328	25328
[char	323	287	36	80472	78600
[int	7	5	2	591040	1216
[long	1	1	0	48	48
[short	6	6	0	864	864
byte	0	0	0	0	0
char	0	0	0	0	0
G_Command	4	4	0	352	352
G_Gerar	1	1	0	8	8
G_Tasks	1	1	0	32	32
int	0	0	0	0	0
long	0	0	0	0	0
P_Estrutura	1	1	0	256	256
P_Montar_Tabela	1	1	0	8	8
P_Recuperar_XML	1	1	0	8	8
Programar	0	0	0	0	0
S_Policy	1	1	0	8	8
S_Repository	1	0	1	24	0
S_Selecionar	0	0	0	0	0
short	0	0	0	0	0
java.lang	13	13	0	1248	1248

Figura 4.38 – Consumo de memória do gerador de agente.



>Package	Base Time (seconds)	Average Base Time (seconds)	Cumulative Time (seconds)	Calls
(default package)	0,367238	0,004769	0,367238	77
[byte	0,000000	0,000000	0,000000	0
[char	0,000000	0,000000	0,000000	0
[int	0,000000	0,000000	0,000000	0
[long	0,000000	0,000000	0,000000	0
[short	0,000000	0,000000	0,000000	0
byte	0,000000	0,000000	0,000000	0
char	0,000000	0,000000	0,000000	0
G_Command	0,000236	0,000059	0,000236	4
G_Gerar	0,075761	0,037880	0,076320	2
G_Tasks	0,000323	0,000323	0,000323	1
int	0,000000	0,000000	0,000000	0
long	0,000000	0,000000	0,000000	0
P_Estrutura	0,000305	0,000005	0,000305	62
P_Montar_Tabela	0,000819	0,000410	0,000819	2
P_Recuperar_XML	0,261969	0,261969	0,349622	1
Programar	0,017617	0,017617	0,367238	1
S_Policy	0,003739	0,001869	0,003794	2
S_Repository	0,000055	0,000055	0,000055	1
S_Selecionar	0,006415	0,006415	0,086529	1
short	0,000000	0,000000	0,000000	0
java.lang	0,000000	0,000000	0,000000	0

Figura 4.39 – Análise do tempo de execução (*Execution Time Analysis*).

O teste aplicado é também conhecido pelo jargão da Computação como “teste de carga” ou, em inglês, *benchmarking*. Neste trabalho, o tempo de resposta corresponde ao tempo para completar o ciclo de programação do sensor, que inclui:

- O tempo gasto para que o BWSNET *Proxy* receba a requisição do serviço;

- O tempo de processamento das mensagens de serviço. Isso corresponde ao tempo para processamento das mensagens SOAP e para acionamento do gerador de agente;
- Tempo para execução do *Wrapper*. Cumpre lembrar que esse módulo é responsável pela execução dos comandos (programas) para compilar, instalar e inicializar o sensor;
- Tempo para que o cliente receba uma mensagem acerca do resultado da programação.

Para simular a interação entre a ferramenta de programação (BWSNET *Configuration Tool*) e o BWSNET *Proxy* foi utilizado o software Httperf 0.9 (MOSBERGER e JIN, 1998). O Httperf é um gerador de requisições HTTP capaz de fornecer relatórios acerca do desempenho de aplicações, como, por exemplo, o tempo de resposta. Esse software foi executado numa plataforma de suporte composta pelo sistema operacional Linux (Kernel versão 2.4) e um PC Pentium IV 2.4 GHz com 512 MB de memória RAM. O BWSNET *Proxy* foi executado em uma plataforma de suporte composta pelo sistema operacional Windows (versão 2000 profissional) e um PC Pentium III 700 MHz com 128 MB de memória RAM. Essa configuração foi escolhida com objetivo de representar um dispositivo de menor capacidade computacional cujo desempenho pudesse ser aproximado do desempenho de um dispositivo móvel. Para o enlace, foi utilizada uma rede ETHERNET 10/100baseT. A Figura 4.40 ilustra o ambiente operacional e os artefatos utilizados nos testes.

O tempo de resposta para programação dos sensores está em função da quantidade de artefatos que devem ser compilador e instalados na memória do microcontrolador. Quanto maior o número de funcionalidades (principalmente de tarefas), maior o espaço requerido em memória. Conseqüentemente, maior deve ser o tempo de resposta para efetuar a programação do agente selecionado. Pensando nisso, foram criados três diferentes cenários de programação para o nó sensor de ECG. O primeiro, incluía apenas o núcleo FreeRTOS, ocupando aproximadamente 5 KB de memória. O segundo, previa a instalação do FreeRTOS e alguns artefatos do MedOS, por exemplo, alguns *device drivers*. E, finalmente, o terceiro contendo o FreeRTOS, os artefatos do MedOS e três diferentes tarefas: ECG com única derivação e freqüência de amostragem igual a 100 Hz, ECG com três derivações e freqüência de amostragem igual a 500 Hz e ECG com seis derivações e

freqüência de amostragem igual a 1.000 Hz. Para cada tarefa foi incluído ainda um filtro digital FIR (*Finite Impulse Response*) para eliminar a variação da linha de base, a interferência de 60 Hz e o ruído branco. A implementação desse filtro foi disponibilizada em (Raju, 2005). Esse filtro acrescenta aproximadamente 900 bytes ao tamanho do código de cada tarefa. A Tabela 4.4 resume os principais resultados obtidos. A duração do ciclo de programação do sistema corresponde ao tempo de resposta médio arredondado (em segundos) obtido depois de 10 requisições enviadas pelo Httperf ao BWSNET *Proxy*.

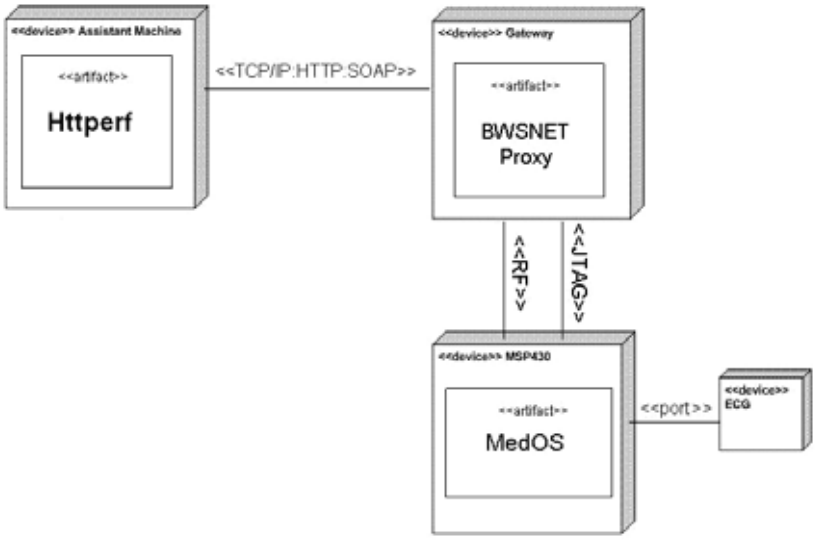


Figura 4.40 - Disposição dos dispositivos e dos artefatos para realização do teste de carga (Barbosa *et al*., 2006).

Tabela 4.4 – Tempo de resposta obtido para a programação do sistema pela Internet (Barbosa *et al*. 2006).

Tamanho dos artefatos (arredondado)	**Duração do ciclo de programação (arredondado)**	**Descrição do conteúdo**
5KB	4s	Apenas o FreeRTOSs
6KB	5s	FreeRTOS e MedOS
9KB	10s	FreeRTOS, MedOS e três tarefas: ECG1/100Hz, ECG3/500Hz ECG6/1000Hz

5 – CONCLUSÕES

5.1 – CONSIDERAÇÕES FINAIS

As redes de sensores do corpo humano devem ser projetadas para operar de maneira autônoma. Por outro lado, devem oferecer mecanismos que remetam o controle aos profissionais da saúde. Esses profissionais são os responsáveis legais pelas decisões acerca do monitoramento dos pacientes, mesmo que essas decisões tenham sido geradas pelo próprio sistema. Em última instância, a avaliação clínica é sempre soberana. A originalidade deste trabalho está em tratar de forma complementar estas hipóteses. Isso pode colaborar para a adoção de fato das redes de sensores em aplicações clínicas.

Neste trabalho, argumenta-se que as intervenções humanas podem contribuir para aumentar a eficiência do sistema, à medida que o próprio sistema se ajusta para atender aos interesses dos usuários (aplicações). Como exemplo, a Figura 3.15b (mostrada no Capítulo 3) apresenta um agente (programa) gerado com base nos requisitos de uma aplicação. A intervenção do usuário (durante a programação do sensor) obrigou o sistema a se ajustar para economizar energia e, assim, aumentar o tempo de funcionamento. De maneira análoga, outros parâmetros podem ser otimizados para aumentar a eficiência do sistema diante dos requisitos das aplicações. Por exemplo, as taxas utilizadas para transmissão dos sinais, a quantidade de perdas relativas aos dados transmitidas, a utilização da memória quando é necessário o armazenamento temporário dos dados capturados, dentre outros. Entretanto, todos os ajustes de parâmetros precisam ser efetuados com base no conhecimento especialista. Para isso, é necessário que a modelagem da arquitetura de sistema seja precedida pela modelagem do domínio das aplicações. Assim, avalia-se que este paradigma tenha ainda grande potencial para inovações.

A complexidade do tema tratado neste trabalho exigiu a criação de uma arquitetura de software de sistema com os seguintes atributos de projeto: *(i)* clara separação de interesses, sendo que a programação e a configuração de RSCH foram os aspectos destacados; *(ii)* modularidade como pressuposto para alcançar a qualidade, esta delimitada por atributos baseados em normas internacionais, e *(iii)* diferentes níveis de abstração que possibilitaram

a geração das inovações apresentadas nesta tese, tendo como base os interesses pré-estabelecidos.

A implementação da arquitetura resultou em sistemas originais e com grande potencial de utilidade. Por exemplo, uma ferramenta para programação da rede de sensores do corpo humano para uso dos profissionais da saúde; um simulador para avaliar os resultados de possíveis intervenções humanas na programação dos sensores; um compilador de nível intermediário que tem como objetivo gerar os programas usados nos sensores, de maneira a aumentar a transparência e a eficiência desses sistemas.

A adoção da tecnologia Java para o desenvolvimento da maioria dos protótipos da arquitetura de software antecede o lançamento dos *Spots*[28] da Sun Microsystems. Os *spots* são implementações de uma arquitetura de sensores sem fios com suporte nativo ao Java. A partir da disponibilização desses dispositivos (prevista para janeiro de 2008) uma única plataforma de desenvolvimento (Java) poderá ser utilizada na implementação de sistemas que compõem as redes de sensores. Além disso, se comparados com os sensores que implementam a arquitetura Motes (apresentada no Capítulo 2), os *spots* possuem maior capacidade computacional, vários sensores digitais embutidos e, principalmente, uma máquina virtual Java (Squawk VM) com maiores possibilidades em relação ao *framework* TinyOS, por exemplo, suporte à migração de objetos. Contudo, modificações na arquitetura SOAB para utilização de novas tecnologias, como os *spots*, devem ser facilitadas pela organização em camadas, pelos atributos de qualidade considerados e pelo uso prévio das tecnologias Java e XML.

A validação de uma arquitetura de software é realizada por meio de uma discussão aberta envolvendo os potenciais influenciadores dessa arquitetura. Na prática, tem como objetivo analisar quão apropriada é uma arquitetura de software proposta a fim de satisfazer um conjunto de requisitos não-funcionais dos sistemas (MENDES, 2002, p.62). Diante disso, os testes efetuados sobre os protótipos revelam aspectos de qualidade do sistema, considerados durante o projeto e especificação da SOAB e apresentados no Capítulo 3. Portanto, comprovam que as idéias propostas neste trabalho são factíveis, eficazes e, até

[28] http://www.sunspotworld.com/

mesmo, eficientes. Embora, até o momento, desconheça-se a existência de outros sistemas que implementam as mesmas funcionalidades para efeito de comparações.

Finalmente, este trabalho apresenta contribuições que permitem viabilizar clinicamente a utilização das redes de sensores do corpo humano, possibilitando a intervenção nos sensores e o reuso desses sistemas em diferentes aplicações de monitoramento da saúde.

5.2 – PROPOSTAS PARA TRABALHOS FUTUROS

5.2.1 – Avaliação da usabilidade do modelo de programação

Utilizabilidade (STAA, 2000, p.69) ou usabilidade (em inglês, *usability*) refere-se à capacidade do software em ser entendido, aprendido, usado e ser atrativo ao usuário, quando executado em certas condições (ISO, 2001). De acordo com (ISO, 2001), alguns aspectos da funcionalidade, da confiabilidade e da eficiência podem afetar a usabilidade. Mas, para efeito da normalização não são classificados como usabilidade.

Em geral, a avaliação da usabilidade busca quantificar quão bem as pessoas podem utilizar ou se beneficiar da utilização de um software. Comumente, o software a ser avaliado é representado por uma interface que pode ser gráfica, sonora, por linha de comandos ou outra qualquer. De acordo com Jakob Nielsen (NIELSEN, 1995), existem basicamente quatro tipos de métodos que podem ser aplicados para avaliação de uma interface de usuário:

- Métodos automáticos: a usabilidade é quantificada pela especificação da interface. Essa especificação é embutida num outro sistema (software) desenvolvido para esse propósito. Funciona como um jogo onde as ações do usuário são computadas (pontuadas) pelo sistema com total transparência;
- Métodos empíricos: a usabilidade é avaliada por meio de testes com usuários reais;
- Métodos formais: utilizam modelos matemáticos para calcular a usabilidade, em geral, por meio de equações;
- Métodos informais: esses utilizam regras baseadas no conhecimento e na experiência dos avaliadores para uma inspeção da interface de usuário. Um exemplo é a inspeção baseada em heurísticas. Esse método envolve o julgamento

da interface de usuário baseado em uma lista de princípios de usabilidade, chamada lista de heurísticas de usabilidade (NIELSEN, 2006).

Os métodos empíricos representam a melhor forma de avaliar as interfaces de usuários. Também, são os métodos mais utilizados. Entretanto, podem ser de alto custo, quando envolvem muitos usuários. Ainda, podem mascarar deficiências de usabilidade quando a seleção dos participantes dos testes é inadequada (NIELSEN, 1995).

A aplicação de um teste de usabilidade baseado num método empírico envolve a criação de um cenário ou de uma situação realística. Nela o usuário participante desenvolve tarefas usando o software. Enquanto isso, os avaliadores observam e tomam notas. Também, podem ser utilizados questionários, registros em vídeo e até instruções para utilização. Além disso, durante os testes podem ser utilizadas técnicas sofisticadas para obtenção de dados. Por exemplo, "pensar em voz alta" (em inglês, *think aloud*) e o rastreamento do movimento dos olhos (em inglês, *eye tracking*).

Em (CIMINO *et al*., 1997) é apresentada uma metodologia para avaliação da usabilidade de tecnologias direcionadas para a área da saúde. Essa metodologia envolve a aplicação de um método empírico que inclui testes, questionários e registros em vídeo. O desenvolvimento dessa metodologia foi fortemente influenciado por uma área de pesquisa chamada de análise de tarefas cognitivas (em inglês, *cognitive task analysis*). A análise de tarefas cognitivas concentra-se em caracterizar as tomadas de decisões e as habilidades de raciocínio dos participantes da mesma forma como eles desenvolvem suas atividades cotidianas.

O método desenvolvido por James Cimino (CIMINO *et al.*, 1997) considera os seguintes estágios:

1. Desenvolvimento de um plano de testes, contendo identificação e descrição de todos os objetivos do testes, ferramentas e recursos necessários;
2. Seleção (agrupamento) dos participantes baseada em critérios de representatividade. Esses critérios são utilizados para homogeneização dos grupos e futuras análises comparativas entre grupos. O número mínimo de elementos de cada grupo deve ser igual a oito elementos. Isso possibilita a aplicação de métodos estatísticos básicos, por exemplo, média, variância e outros;

3. Seleção das tarefas ou casos de testes;
4. Configuração do ambiente de testes;
5. Execução dos testes de usabilidade.
6. Análise dos dados gerados;
7. Geração de documento com propostas para modificações da interface.

5.2.1.1 – Proposta de uma metodologia para execução de testes de usabilidade

Do ponto de vista dos usuários é possível qualificar o modelo de programação proposto neste trabalho, pela aplicação de testes de usabilidade. Para isso, foram desenvolvidos testes de usabilidade com base num método empírico. Esse método é composto por um plano de teste e por alguns protótipos. Durante o desenvolvimento dos protótipos foi considerada a **lista das dez principais heurísticas** [29] que devem ser utilizadas para inspeção de uma interface de usuário. Essas heurísticas fazem parte de um método de avaliação informal (baseado em inspeções), desenvolvido por Jakob Nielsen (NIELSEN, 2006).

O plano de teste proposto neste trabalho tem como objetivo apresentar aos usuários a interface de programação do nó sensor de ECG, especificada na Seção 3.4 e apresentada na Seção 4.1.1. Pode ser descrito pelas seguintes etapas:

1. Inicialmente deve ser escolhido um perfil de usuários participantes. Recomenda-se a escolha de estudantes do curso de medicina, preferencialmente, alunos matriculados no internato[30]. Esse perfil de usuário tem as seguintes características:
    a. São jovens com idade inferior a 30 anos;
    b. Em geral, utilizam computador em suas atividades cotidianas e dispõem de computador e acesso a Internet em casa;
    c. São generalistas em relação ao conhecimento técnico, isto é, ainda não se especializaram numa determinada área. Entretanto, já cursaram conteúdos de diversas áreas da medicina, em contrapartida aos estudantes matriculados nos primeiros períodos.

[29] http://www.useit.com/papers/heuristic/
[30] O internato é o estágio obrigatório em serviços de saúde destinado a complementar e aprimorar os atos médicos e conhecimentos apreendidos nos períodos anteriores do curso de graduação.

2. Os testes devem executados à distância pela Internet, sem qualquer intervenção dos avaliadores.

Em relação às tarefas solicitadas:

- Para cada participante é fornecido o endereço eletrônico para acesso à ferramenta de programação.
    a. Deve ser solicitada a leitura dos manuais de instrução da ferramenta aos participantes. Neles constam as tarefas que devem ser executadas.
    b. Foram propostos três exercícios a cada participante. Esses exercícios requerem que o participante desenvolva três diferentes programações para o sensor de ECG. Essas programações são relativas a três diferentes casos clínicos. Os casos sugeridos são os seguintes:
        i. Paciente infartado em estado grave na UTI;
        ii. Paciente jovem com dor precordial atípica;
        iii. Paciente com palpitação associada à síncope.

3. Ao final de cada exercício é solicitado que o participante faça uso do simulador (apresentado na Seção 4.1.1). Esse procedimento tem como objetivo mostrar ao participante o desempenho da programação efetuada e, também, um resumo do que foi selecionado. Avalia-se que isso é importante para que o participante possa entender de fato o que esta sendo feito e, caso não concorde, refaça a programação.
4. Ao final da simulação da programação (para cada caso clínico) deve ser gerado um relatório. Os relatórios devem ser salvos (armazenados) em arquivos. Posteriormente, o conteúdo desses arquivos deve ser enviado para os avaliadores.
5. Ao final de todos procedimentos, é solicitado o preenchimento de um questionário de avaliação. Esse questionário contém 14 itens sugeridos e dispostos em três formulários (telas). No item final do questionário é solicitado que o participante anexe os arquivos gerados pelo simulador para envio juntamente com o questionário de avaliação.

5.2.1.2 – Protótipos desenvolvidos

Todos os protótipos desenvolvidos para os testes de usabilidade utilizam tecnologia Java e a API Applets. Isso possibilita que os participantes possam acessar os programas (sistemas) por meio do navegador de páginas Web (*web browser*). O sistema está

hospedado atualmente no site Myjavaserver. Pode ser acessado através endereço eletrônico *http://www.myjavaserver.com/~bwsnet*. O site Myjavaserver oferece hospedagem gratuita para sistemas que utilizam tecnologia Java. Além disso, oferece um ambiente de execução para que os programas (escritos em linguagem Java) possam fazer uso de bibliotecas especializadas.

Ao acessar o site, o participante tem acesso à tela principal do sistema. Nela há seis menus de opções, cada um com um ou mais itens. Para acesso aos manuais de instrução o participante deve acessar o item "Manual" do menu "Ajuda". Esse texto (manual de instruções) apresenta uma descrição da ferramenta e instruções básicas sobre como proceder para uso da mesma. A Figura 5.1 exibe a tela principal do sistema.

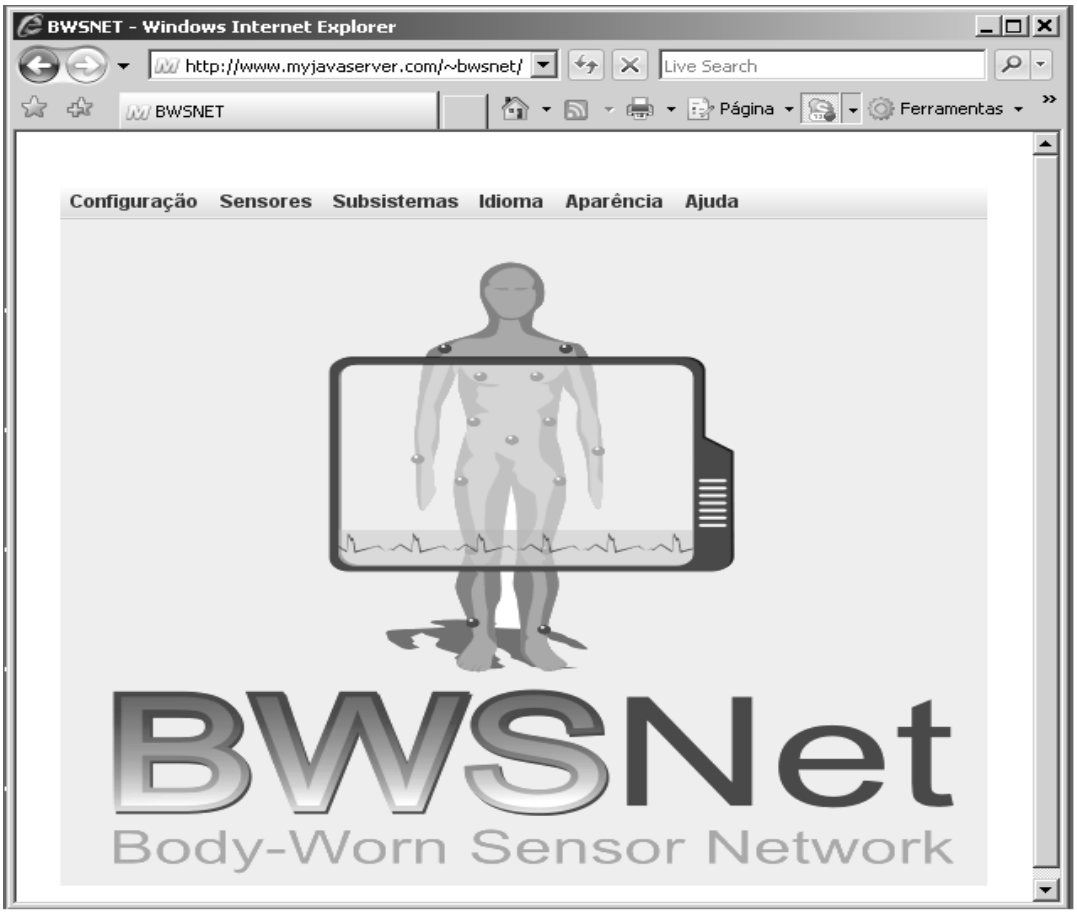


Figura 5.1 – Tela principal do sistema.

Ao final da leitura do manual de instruções, o participante é convidado a acessar a tela de programação do sensor de ECG para executar as tarefas descritas na Seção 5.2.1.1. Para isso, deve acessar o menu “Sensores” e, em seguida, o item “Nó sensor de ECG”.

A disposição e o formato dos objetos gráficos (*layout*) da tela de ECG foram modificados em relação à versão original apresentada na Figura 4.4. Isso foi feito com objetivo de melhorar o entendimento do usuário (participante). As diretrizes utilizadas para as modificações tiveram como base a *lista das dez principais heurísticas* de inspeção propostas por Jakob Nielsen (NIELSEN, 2006). A Figura 5.2 exibe a tela de ECG utilizada para os testes de usabilidade.

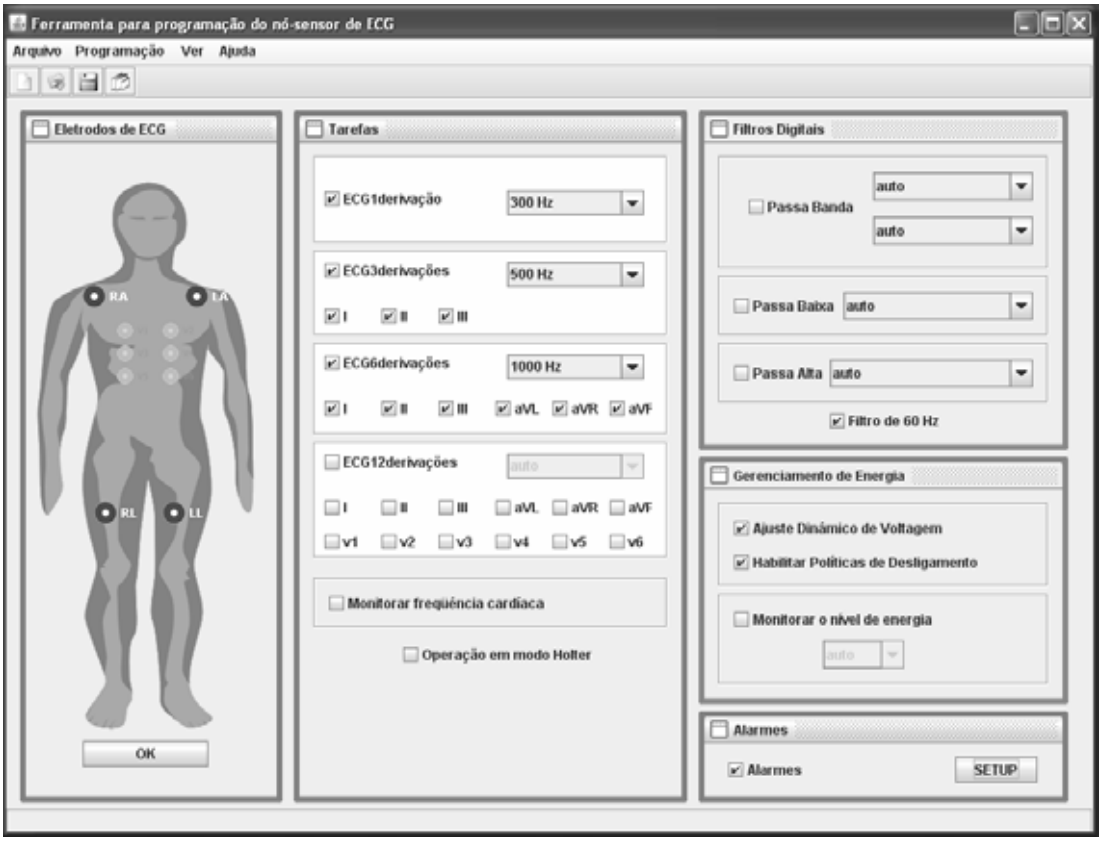


Figura 5.2 – Tela para programação do sensor de ECG.

O menu “Ajuda” que aparece na tela para programação do sensor de ECG (Figura 5.2) contém dois itens: o manual de instruções e o acesso ao questionário de avaliação do sistema. O manual de instruções contém informações sobre como operar a ferramenta e, ainda, a descrição dos exercícios que os participantes devem executar. O questionário de

avaliação solicita que o participante preencha os campos e, em seguida, faça a submissão pela Internet. As Figuras 5.3, 5.4 e 5.5 apresentam as telas do questionário de avaliação para ilustrar as perguntas que os participantes devem responder.

Ao ser preenchido, o questionário de avaliação é imediatamente enviado por e-mail para os avaliadores. Para o desenvolvimento do questionário de avaliação foi utilizada a API JavaMail 1.4[31]. Como exemplo, um trecho do relatório recebido pelos avaliadores é mostrado na Figura 5.6. A solução técnica adotada procurou minimizar os custos de manutenção desse serviço (testes de usabilidade). Avaliou-se que a quantidade de participantes seria pequena e, provavelmente, com poucos acessos concorrentes. Essa solução evita a necessidade de recursos de armazenamento mais sofisticados, como, por exemplo, bancos de dados e servidores dedicados.

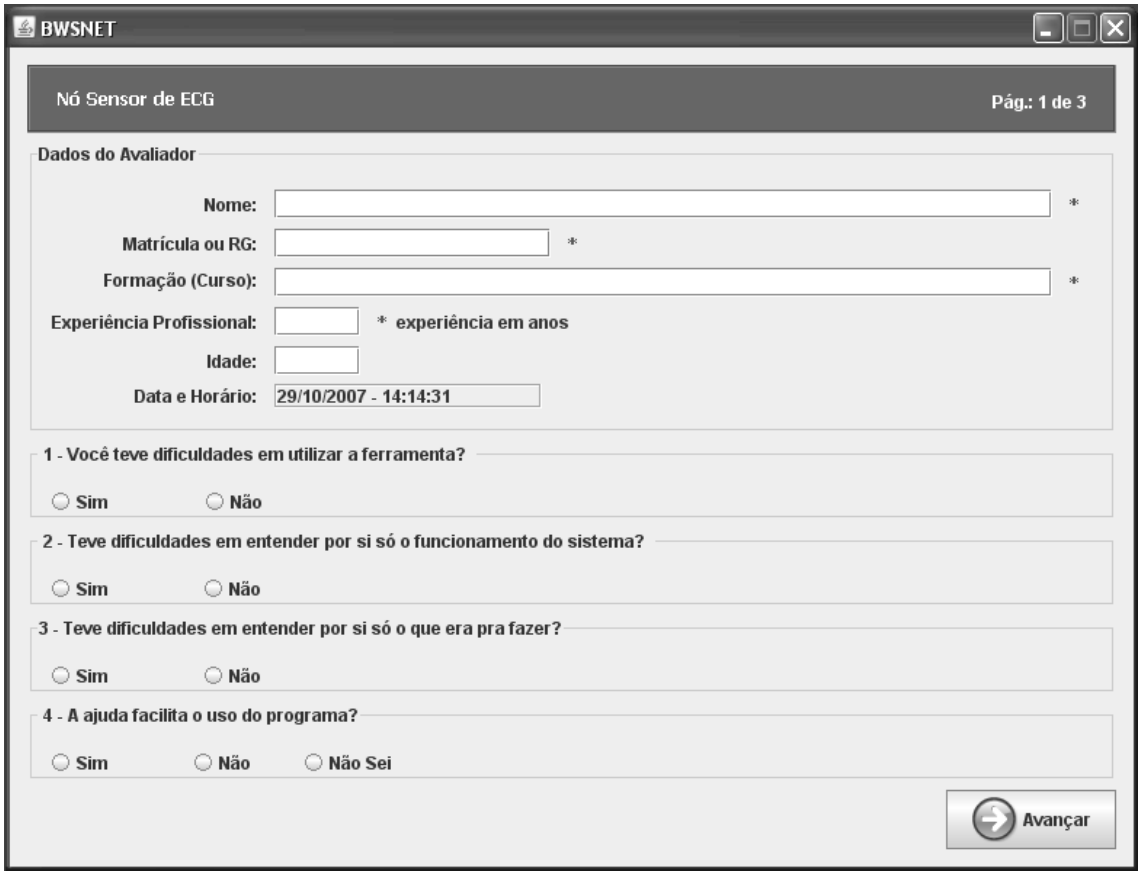


Figura 5.3 – Questionário de avaliação: primeira página.

[31] http://java.sun.com/products/javamail/

Figura 5.4 – Questionário de avaliação: segunda página.

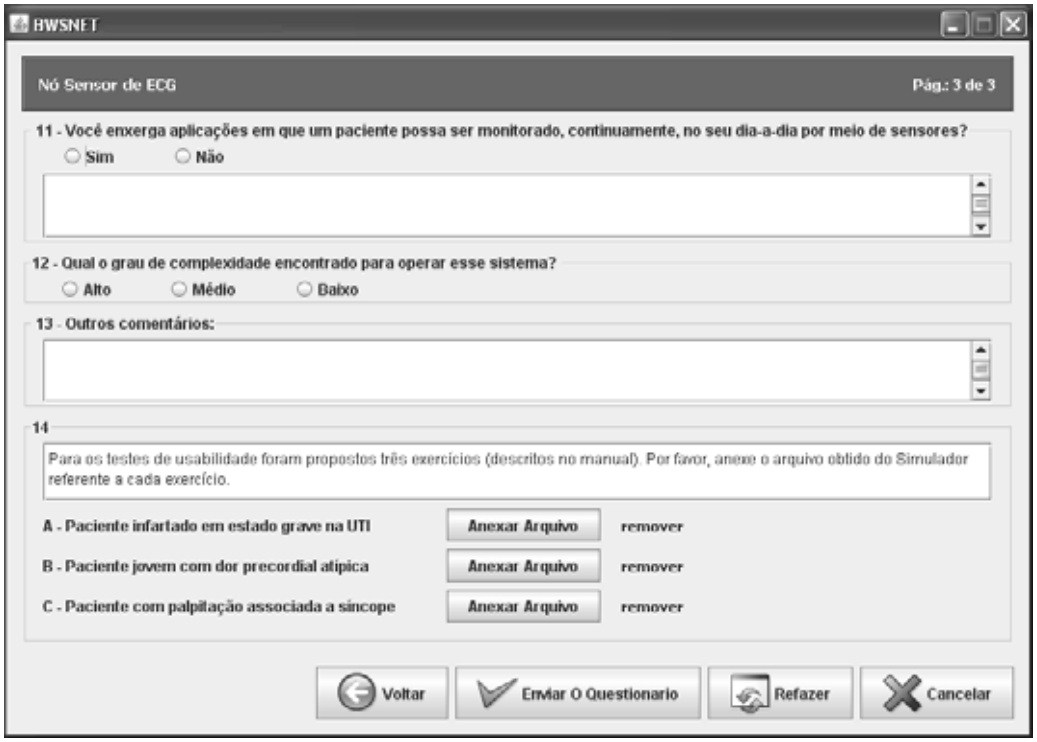
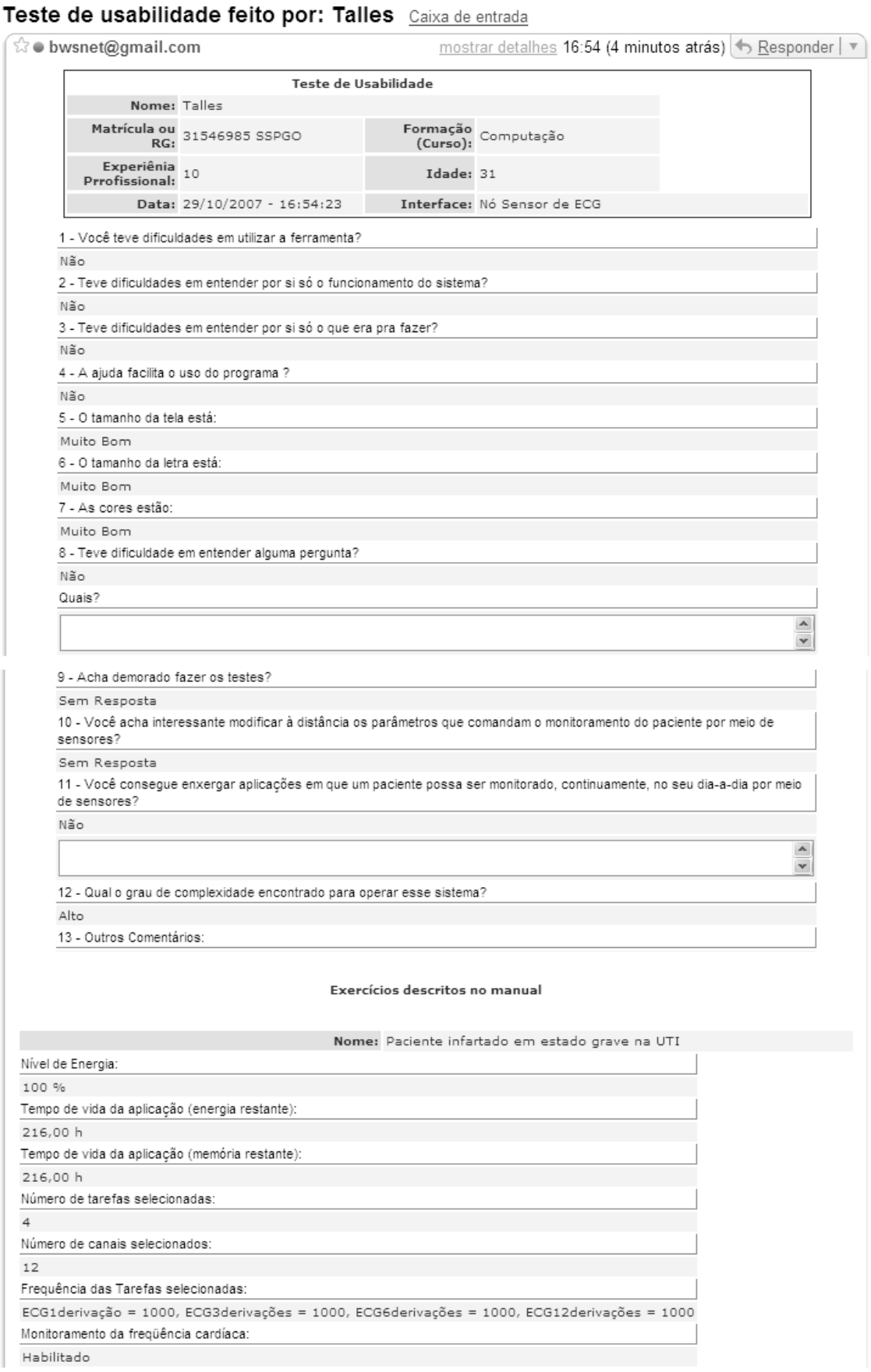


Figura 5.5 – Questionário de avaliação: terceira página.



Teste de usabilidade feito por: Talles Caixa de entrada

bwsnet@gmail.com mostrar detalhes 16:54 (4 minutos atrás) Responder

Teste de Usabilidade			
Nome:	Talles		
Matrícula ou RG:	31546985 SSPGO	**Formação (Curso):**	Computação
Experiência Prrofissional:	10	**Idade:**	31
Data:	29/10/2007 - 16:54:23	**Interface:**	Nó Sensor de ECG

1 - Você teve dificuldades em utilizar a ferramenta?	
Não	
2 - Teve dificuldades em entender por si só o funcionamento do sistema?	
Não	
3 - Teve dificuldades em entender por si só o que era pra fazer?	
Não	
4 - A ajuda facilita o uso do programa ?	
Não	
5 - O tamanho da tela está:	
Muito Bom	
6 - O tamanho da letra está:	
Muito Bom	
7 - As cores estão:	
Muito Bom	
8 - Teve dificuldade em entender alguma pergunta?	
Não	
Quais?	
9 - Acha demorado fazer os testes?	
Sem Resposta	
10 - Você acha interessante modificar à distância os parâmetros que comandam o monitoramento do paciente por meio de sensores?	
Sem Resposta	
11 - Você consegue enxergar aplicações em que um paciente possa ser monitorado, continuamente, no seu dia-a-dia por meio de sensores?	
Não	
12 - Qual o grau de complexidade encontrado para operar esse sistema?	
Alto	
13 - Outros Comentários:	

Exercícios descritos no manual

Nome:	Paciente infartado em estado grave na UTI
Nível de Energia:	
100 %	
Tempo de vida da aplicação (energia restante):	
216,00 h	
Tempo de vida da aplicação (memória restante):	
216,00 h	
Número de tarefas selecionadas:	
4	
Número de canais selecionados:	
12	
Frequência das Tarefas selecionadas:	
ECG1derivação = 1000, ECG3derivações = 1000, ECG6derivações = 1000, ECG12derivações = 1000	
Monitoramento da freqüência cardíaca:	
Habilitado	

Figura 5.6 – Trecho de um relatório recebido pelo avaliador.

5.2.2 – Avaliação da usabilidade do modelo de configuração

A avaliação da usabilidade do modelo de configuração apresentado neste trabalho pode fornecer informações acerca da utilidade e da facilidade em descrever aplicações com base em grafos de variáveis baseado em estados, grafos de QoS de sensores e valores de prioridade associados às tarefas executadas pelos sensores. Para isso, é necessária a elaboração de um plano de testes e a implementação de protótipos.

5.2.3 – Implementação do componente MiLAN

Os possíveis ajustes das configurações de rede desempenhados pelo MiLAN (apresentados na Seção 2.2.4.2 e na Seção 3.6.3) são importantes para automatização do modelo de configuração apresentado neste trabalho. A implementação do componente MiLAN, especificado com parte do componente Tradutor MedOS (Seção 3.6), deve possibilitar que os ajustes automáticos possam ser executados e avaliados.

5.2.4 – Outras perspectivas

Dentro dos diferentes temas que foram investigados neste trabalho, ainda, há espaço para os seguintes desdobramentos:

- Estudo e desenvolvimento de interfaces homem-máquina para RSCH;
- Estudo e desenvolvimento de compiladores inteligentes para as redes de sensores sem fios, considerando outros parâmetros, por exemplo, atrasos e perdas e, também, outras aplicações;
- Estudo e desenvolvimento de algoritmos para escalonamento de tarefas, considerando os requisitos das aplicações;
- Estudo e desenvolvimento de padrões de software para RSCH. Por exemplo, dos estudos realizados durante este trabalho constatou-se que existe uma lacuna acerca de padrões para interoperabilidade entre as RSCH e os demais sistemas de informação utilizados em Saúde.

REFERÊNCIAS BIBLIOGRÁFICAS

AHAMED S.; VYAS A.; M. ZULKERNINE. “Towards developing sensor networks monitoring as a middleware service”. In: THE INTERNATIONAL WORKSHOP ON AD HOC AND SENSOR NETWORKS - INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING (ICPP '04). Montreal (Canadá), agosto 2004. *Proceedings of...* Montreal (Canadá), 2004.

ANDERSON E. T. “The Case for Application-Specific Operating Systems”. In: WORKSHOP ON WORKSTATION OPERATING SYSTEMS (WWOS), 3., 1992. Disponível em: <http://www.cs.washington.edu/homes/tom/pubs/app-spec.html>. Acesso em: 7 set. 2006.

ANLIKER U.; WARD J.A.; LUKOWICZ P.; TROSTER G.; DOLVECK F.; BAER M.; KEITA F.; SCHENKER E.B.; CATARSI F.; COLUCCINI L.; BELARDINELLI A.; SHKLARSKI D.; ALON M.; HIRT E.; SCHMID R.; VUSKOVIC M.; *AMON: A Wearable Multiparameter Medical Monitoring and Alert System*. *IEEE transactions on information technology in biomedicine,* vol. 8, no. 4, dezembro 2004.

APACHE - Xerces Java Parser 1.4.4, 2006. Disponível em: <http://xerces.apache.org/xerces-j/>. Acessado em: 7 set. 2006.

ASADA H. H.; SHALTIS P.; REISNER A.; SOKWOO R.; HUTCHINSON R.C. *Mobile Monitoring with Wearable photoplethysmographic biosensors*. *IEEE Engineering in Medicine and Biology Magazine*, vol. 22, maio-junho 2003.

ATMEL – ATmega128. Especificação de produtos. Disponível em: <http://www.atmel.com/dyn/products/product_card.asp?family_id=607&family_name=AVR+8%2DBit+RISC+&part_id=2018>. Acessado em: 7 set. 2006.

AVANCHA S.; JOSHI A.; FININ T. *Enhanced Service Discovery in Bluetooth*. *IEEE Computers*, vol. 35, no. 6, Junho 2002.

BAJAZAT R.; Thinlet. Disponível em: <http://thinlet.sourceforge.net/home.html> . Acessado em 7 set. 2006.

BALDUS D., KLABUNDE K., MÜSCH G. “Reliable Set-Up of Medical Body-Sensor Networks”. In: EUROPEAN WORKSHOP ON WIRELESS SENSOR NETWORKS (EWSN '04), 1., Berlim (Alemanha), 2004. *Proceedings of...* Berlim (Alemanha): Springer, 2004.

BARBOSA, T. M. G. DE A.; SENE JR, I. G.; CASTRO, L. S. S.; BRANISSO P. J. H.; FIGUEIREDO E. C.; CARVALHO, H. S.; DA ROCHA, A. F.; NASCIMENTO, F. A. O. Sistema móvel para monitoração da saúde: algoritmo para captura inteligente de sintomas. In: CONGRESSO BRASILEIRO DE INFORMÁTICA EM SAÚDE (CBIS), 9., 2004, Ribeirão Preto – SP. *Anais...* Ribeirão Preto – SP:USP, 2004.

BARBOSA, T. M. G. DE A.; SENE JR., I. G.; CARVALHO, H. S.; DA ROCHA, A. F.; NASCIMENTO F. A. O. Arquitetura de software para redes de sensores sem fios: a proeminência do middleware. In: SEMINÁRIO INTEGRADO DE SOFTWARE E HARDWARE (SEMISH), 32., 2005, São Leopoldo – RS. *Anais do XXV Congresso da Sociedade Brasileira de Computação (CSBC)*, São Leopoldo – RS:UNISINOS, 2005.

BARBOSA T. M. G. DE A.; SENE JR I. G.; CARVALHO H. S.; DA ROCHA A. F.; NASCIMENTO F. A. O.; CAMAPUM J. F. “Application-oriented programming model for sensor networks embedded in the human body”. In: *Annual International Conference IEEE Engineering in Medicine and Biology Society (EMBC)*, 28., 2006, Nova Iorque (Estados Unidos). *Proceedings of…* Nova Iorque (Estados Unidos):IEEE-EMBS, 2006.

BARBOSA T. M. G. DE A., SENE JR I. G., CARVALHO H. S., DA ROCHA A. F. E NASCIMENTO F. A. O. Programação de uma rede de sensores para o corpo humano a partir de uma interface gráfica. In: CONGRESSO BRASILEIRO DE INFORMÁTICA EM SAÚDE (CBIS), 10., 2006, Florianópolis – SC. *Anais...,* Florianópolis – SC:SBIS, 2006b.

BARBOSA, T. M. G. A., ROCHA, ADSON FERREIRA DA, SAMPAIO, HERVALDO CARVALHO, SENE JR, IWENS G ; NASCIMENTO E FRANCISCO A O DO. Compilador Inteligente para *Body Sensor Networks*. In: VII Workshop de Informática Médica, evento do SBQS 2007 - VI Simpósio Brasileiro de Qualidade de Software, 2007, Porto de Galinhas-PE. Anais do VII Workshop de Informática Médica, 2007.

BARBOSA, T. M. G. DE A., SENE JR, IWENS GERVÁSIO, SAMPAIO, HERVALDO CARVALHO ; ROCHA, ADSON FERREIRA DA ; NASCIMENTO, FRANCISO ASSIS DE O ; CARVALHO A. L. J . "A New Model for Programming Software in Body Sensor Networks". In: *Annual International Conference of the IEEE EMBS*, 29., 2007, Lyon (França). *Proceedings of* ... Piscataway NJ: IEEE Engineering in Medicine and Biology Society, 2007b.

BARRY R. *FreeRTOS*. Disponível em: <http://www.freertos.org/>. Acessado em: 7 set. 2006.

BENINI L.; BOGLIOLO A.; DE MICHELI G. *A survey of design techniques for system-level dynamic power management. IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 8 junho 2000. ISSN: 1063-8210.

BERKELEY, U. C. TinyOS. Disponível em: <http://www.tinyos.net>. Acessado em: 7 set. 2006.

BISHOP, J. *Thermistor Temperature Transducer to ADC Application*. *Analog Applications Journal*, novembro 2000. Disponível em: <http://focus.ti.com/lit/an/slyt156/slyt156.pdf#search=%22Thermistor%20Temperature%20Transducer%20to%20ADC%20Application%22>. Acessado em: 7 set. 2006.

BLUMENTHAL J.; HANDY M.; GOLATOWSKI F.; HAASE M.; TIMMERMANN D. "Wireless sensor networks - new challenges in software engineering". In: IEEE INTERNATIONAL CONFERENCE ON EMERGING TECHNOLOGIES AND FACTORY AUTOMATION (ETFA'03), 9., 2003, Lisboa (Portugal). *Proceedings of* ... Lisboa (Portugal), 2003.

BONATO P. *Wearable sensors systems and their Impact on biomedical engineering*. *IEEE Engineering in Medicine and Biology Magazine,* maio-junho 2003.

BOUCH. A. *A user's perspective of the network QoS and charging.* 2001. Tese (doutorado em Ciência da Computação) – *Department of Computer Science*, Universidade *College London*, Londres.

CARVALHO H. S.; RIBEIRO-NETO B., COELHO JR C. “Evidence based cardiovascular information retrieval. In: WORLD CONGRESS OF CARDIOLOGY, 14., Austrália, 2002. Resumo publicado em *the Journal of the American College of Cardiology supplement*, 2002.

CARVALHO H.; HEINZELMAN W.; MURPHY A.; COELHO C. “Network-based distributed systems middleware. In: INTERNATIONAL WORKSHOP ON MIDDLEWARE FOR PERVASIVE AND AD-HOC COMPUTING, 1., Junho 2003, Rio de Janeiro. *Proceedings of...* Rio de Janeiro, 2003.

CARVALHO H. S.; ZUQUIM A.;O LOUREIRO A. A.; COELHO. JR. C.; VIEIRA M.; VIEIRA L. F; VIEIRA A.; FERNANDES A.; SILVA JR. D.; DA MATA J.; NACIF J. “Efficient power management in real-time embedded systems”. In: IEEE INTERNATIONAL CONFERENCE ON EMERGING TECHNOLOGIES AND FACTORY AUTOMATION (ETFA'03), 9., 2003, Lisboa (Portugal). *Proceedings of* ... Lisboa (Portugal), 2003b.

CARVALHO H.; PERILLO M.; HEINZELMAN W; MURPHY A. *Middleware to Support Sensor Network Applications. IEEE Network Magazine Special Issue*, Janeiro 2004.

CARVALHO, H. S. *Data fusion implementation in sensor networks applied to health monitoring.* 2005. 158 f. Tese (doutorado em Ciência da Computação) - Departamento de Ciência Computação, Universidade Federal de Minas Gerais, Belo Horizonte.

CASTRO, L. S. S.; BARBOSA, T. M. G. DE A.; SENE JR, I. G. L; BRANISSO P. J. H.; FIGUEIREDO E. C.; CARVALHO, H. S.; DA ROCHA, A. F.; NASCIMENTO, F. A. O. HandMed – um sistema móvel integrado para captura automática de sintomas. In: CONGRESSO BRASILEIRO DE INFORMÁTICA EM SAÚDE (CBIS), 9., 2004, Ribeirão Preto – SP. *Anais...* Ribeirão Preto – SP:USP, 2004.

CERULLI M. O *método oscilométrico de medição da pressão arterial. Revista Hipertensão,* vol. 3, no. 3, 2000.

CHANDRAKASAN P. A.; WENTZLOFF D. D.; CALHOUN H. B.; MIN R.; WANG A.; ICKES N. “Design considerations for next generation wireless power-aware microsensor nodes”. In: INTERNATIONAL CONFERENCE ON VLSI DESIGN (VLSI DESIGN 2004), 17., Mumbai (Índia),janeiro 2004. *Proceedings of*... Mumbai (Índia): IEEE Computer Society, 2004. ISBN 0-7695-2072-3.

CHIPCOM - CC2420. Especificação de produtos. Disponível em: <http://www.chipcon.com/index.cfm?kat_id=10>. Acessado em: 7 set. 2006.

CHONG C.; KUMAR S. *Sensor networks: evolution, opportunities and challenges. Proceedings of the IEEE*, vol. 91, no. 8, agosto 2003.

CHUA S. C.; HIN M. S. G. *Digital Blood Pressure Meter*. Freescale Semicondutor Application Notes, 2005. Disponível em: < www.freescale.com/files/sensors/doc/app_note/AN1571.pdf>. Acessado em 17 de março de 2007.

CIMINO J.; KUSHNIRUK A.; PATEL V. “Usability testing in medical informatics: a cognitive approaches to evaluation of information systems and user interfaces”. In: ANNUAL FALL SYMPOSIUM AMERICAN MEDICAL INFORMATICS ASSOCIATION, 1997, Nashville-TE (Estados Unidos). *Proceedings of*... Nashville-TE (Estados Unidos):AMIA Press, 1997.

CORMEM T. H.; LEISERSON C. E.; RIVEST R. L. *Algorithms*. 1ed. MIT Press, 1997. 1028 p. ISBN: 0-262-53091-0.

COULSON G. *What is a reflective Midddleware?*. *IEEE Distributed Systems on-line,* 2000. Disponível em: <http://dsonline.computer.org/middleware/RMarticle1.htm> . Acessado em 17 de novembro de 2004.

COULOURIS G; DOLLIMORE J.; KINDBERG T. *Sistemas distribuídos: conceitos e aplicações*. 4 ed. Bookman, 2007. 792 p. ISBN: 978-85-60031-49-8.

CROSSBOW - *Crossbow Smarter Sensor in Silicon*. Especificação de produtos. Disponível em: <http://www.xbow.com>. Acessado em: 7 set. 2006.

CULLER D. AND LEVIS P. "Maté: a tiny virtual machine for sensor networks". In: INTERNATIONAL CONFERENCE ON ARCHITECTURAL SUPPORT FOR PROGRAMMING LANGUAGES AND OPERATING SYSTEMS, 10., 2002, San Jose-CA (Estados Unidos). *Proceedings of* … San Jose-CA (Estados Unidos), 2002.

CULLER D., HANDZISKI L., POLASTRE J., HAUER J., SHARP C., WOLISZ A. "Flexible hardware abstraction for wireless sensor networks". In: EUROPEAN WORKSHOP ON WIRELESS SENSOR NETWORKS (EWSN '05), 2., Istanbul (Turquia), 2005. *Proceedings of*... Istanbul (Turquia): Springer, 2005.

DELICATO C. F. *Middleware* baseado em serviços para redes de sensores sem fio. 2005. 180 f. Tese (doutorado em Engenharia Elétrica) - COPPE/PEE, Universidade Federal do Rio de Janeiro, Rio de Janeiro.

DUNKELS A.; ALONSO J.; VOIGT T. "Making TCP/IP viable for wireless sensor networks". In: EUROPEAN WORKSHOP ON WIRELESS SENSOR NETWORKS (EWSN '04), 1., Berlim (Alemanha), 2004 *Proceedings of*... Berlim (Alemanha), 2004.

FARSCHCHI, S.; NUYUJUKIAN, H. P.; PESTEREV, A.; MODY I.; JUDY W. J. A. *TinyOS-Enabled MICA2-Based wireless neural interface*. *IEEE Transactions on Biomedical Engineering*, vol. 53, no. 7, jul. 2006.

FOK C.; ROMAN G.; LU C. "Mobile agent middleware for sensor networks: an application case study". In: INTERNATIONAL CONFERENCE ON INFORMATION PROCESSING IN SENSOR NETWORKS (IPSN'05), 2005, Los Angeles-CA (Estados Unidos). *Proceedings of*... Los Angeles-CA (Estados Unidos):ACM Press, 2005.

FRÖHLICH A. A. *Application-Oriented Operating Systems*, Sankt Augustin:GMD - Forschungszentrum Informationstechnik, 2001, 200 p., ISBN 3-88457-400-0.

FRÖHLICH A.; WANNER L.; HOELLER JR. A.; POLPETA F. "SUPORTE DE SISTEMA OPERACIONAL PARA TRATAMENTO DE HETEROGENEIDADE EM REDES DE SENSORES SEM FIOS". In: Brazilian Workshop on Operating System, 2., 2005, São Leopoldo – RS. *Anais do XXV Congresso da Sociedade Brasileira de Computação (CSBC)*, São Leopoldo – RS:UNISINOS, 2005.

GAY D; LEVIS P; VON BEHREN R.; WELSH M; BREWER E; CULLER D. "The nesC Language: a holistic approach to network embedded systems". In: ACM SIGPLAN 2003 CONFERENCE ON PROGRAMMING LANGUAGE DESIGN AND IMPLEMENTATION, 2003, San Diego-CA (Estados Unidos). *Proceedings of…* San Diego-CA (Estados Unidos), 2003.

GERSTING J. L. Fundamentos matemáticos para a ciência da computação. 5ed. LTC, 2004. 616 p. ISBN: 8521614225.

GRAHAM S.; SIMEONOV S.; BOUBEZ T.; DANIELS G.; DAVIS D.; NAKAMURA Y.; NEYAMA R. *Building Web Services with Java: Making Sense of XML, SOAP, WSDL and UDDI* . 1ed. SAMS Publishing, 2002. 600 p. ISBN-10: 0-672-32181-5.

GREEN, D. Trail: The Reflection AP. Disponível em: <http://java.sun.com/docs/books/tutorial/reflect/>. Acessado em: 7 set. 2006.

GUYTON C. A.; HALL E. J. *Tratado de fisiologia médica.* 11 ed. Elsevier, 2006. 1264 p. ISBN: 8535216413.

GUTIÉRREZ E. M., FRANÇA R. D., BARBOSA T. M. G. DE A., CARVALHO H. S. E DA ROCHA A. F. Desenvolvimento de uma rede de sensores sem fios para o monitoramento biomédico. In: CONGRESSO BRASILEIRO DE INFORMÁTICA EM SAÚDE (CBIS), 10., 2006, Florianópolis – SC. *Anais...,* Florianópolis – SC:SBIS, 2006.

HAN R.; NEUFELD M, DAI H.; "ELF: An Efficient Log-Structured Flash File System for Wireless Micro Sensor Nodes". In: ACM CONFERENCE ON EMBEDDED NETWORKED SENSOR SYSTEMS (SENSYS 2004), 2., 2004, Baltimore (Estados Unidos). *Proceedings of...* Baltimore (Estados Unidos):ACM Press, 2004.

HAN R.; BHATTI S.; CARLSON J.; DAI H.; DENG J.; ROSE J.; SHETH A; SHUCKER B; GRUENWALD C; TORGERSON A. *MANTIS OS: an embedded multithreaded operating system for wireless micro sensor platforms. ACM/Kluwer Mobile Networks & Applications (MONET), Special Issue on Wireless Sensor Networks*, vol. 10, no. 4, agosto 2005.

HILL J. L. *System Architecture for Wireless Sensor Networks*. 2003. 196f. Tese (doutorado em Ciência da Computação) – *Graduate Division*, Universidade da Califórnia, Berkeley.

HUI J. E CULLER D. "The dynamic behavior of a data dissemination protocol for network programming at scale". In: ACM CONFERENCE ON EMBEDDED NETWORKED SENSOR SYSTEMS (SENSYS 2004), 2., 2004, Baltimore (Estados Unidos). *Proceedings of...* Baltimore (Estados Unidos):ACM Press, 2004.

HURLER B.; HOF H.; ZITTERBART M. "A general architecture for wireless sensor networks: first steps". In: INTERNATIONAL WORKSHOP ON SMART APPLIANCES AND WEARABLE COMPUTING, 4., Tóquio (Japão), março 2004. *Proceedings of...* Tóquio (Japão), 2004.

ICL - IMPERIAL COLLEGE LONDON – Ubiquitous monitoring environment for wearable and implantable sensors. Disponível em: <http://www.doc.ic.ac.uk/vip/ubimon/partners/index.html> . Acessado em: 7 set. 2006.

IEEE 802.15 WPAN™ Task Group 4 (TG4). Disponível em: <http://www.ieee802.org/15/pub/TG4.html> . Acessado em: 7 set. 2006.

IEEE Std 1061-1992. *IEEE standard for a software quality metrics methodology*. Nova Iorque (Estados Unidos):IEEE Computer Society Press, 1993. 96 p., ISBN 1-55937-277-X.

IETF - Service Location Protocol (SLP), 2001. Disponível em: <http://www.ietf.org/html.charters/OLD/svrloc-charter.html>. Acessado em: 7 set. 2006.

ISO/IEC 9126-1:2001. *Software engineering – Product quality – Part 1: quality model*. Suíça:ISO Press, 2001. 25 p.

JOSÉ, ALEXANDRE B, BARBOSA, T. M. G. A., CARVALHO, HERVALDO SAMPAIO, ROCHA, ADSON FERREIRA DA, NASCIMENTO, FRANCISCO DE ASSIS OLIVEIRA. “A Framework for Automated Evidence Gathering with Mobile Systems Using Bayesian Networks”. In: *Annual International Conference of the IEEE EMBS*, 29., 2007, Lyon (França). *Proceedings of* ... Piscataway NJ : IEEE Engineering in Medicine and Biology Society, 2007.

JOVANOV E.; LORDS A.; RASKOVIC D.; COX; ADHAMI R.; ANDRASIK F. *Stress Monitoring Using a Distributed Wireless Intelligent Sensor System*. *IEEE Engineering in Medicine and Biology Magazine,* maio-junho 2003.

JOVANOV E.; MILENKOVIC A.; BASHAM S.; CLARK D.; KELLEY D. “Reconfigurable intelligent sensors for health monitoring: a case study of pulse oximeter sensor”. In: ANNUAL INTERNATIONAL CONFERENCE OF THE IEEE ENGINEERING IN MEDICINE AND BIOLOGY Society. São Francisco (Estados Unidos), 26., setembro 2004. *Proceedings of* ... São Francisco (Estados Unidos):IEEE-EMBS, 2004.

JOVANOV E. “WHMS - Wearable Health Monitoring System”. Disponível em: <http://www.ece.uah.edu/~jovanov/whrms/>. Acessado em: 7 set. 2006.

JOVANOV E.; OTTO C; MILENKOVIC A; SANDERS C. "System Architecture of a Wireless Body Area Sensor Network for Ubiquitous Health Monitoring," *Journal of Mobile Multimedia*, vol. 1, no. 4, 2006b.

KARLOF C; SASTRY N; WAGNER D. “TinySec: a link layer security architecture for wireless sensor networks”. In: ACM CONFERENCE ON EMBEDDED NETWORKED SENSOR SYSTEMS (SENSYS 2004), 2., 2004, Baltimore (Estados Unidos). *Proceedings of*... Baltimore (Estados Unidos):ACM Press, 2004.

KASTEN O.; RÖMER K: “Beyond Event Handlers: Programming Wireless Sensors with Attributed State Machines”. In: INTERNATIONAL CONFERENCE ON INFORMATION PROCESSING IN SENSOR NETWORKS (IPSN’05), 2005, Los Angeles-CA (Estados Unidos). *Proceedings of*... Los Angeles-CA (Estados Unidos):ACM Press, 2005.

KICZALES G. *Beyond the black box: Open implementation*. *IEEE Software*, vol. 13, Janeiro 1996.

KICZALES G.; DEVANBU P.; BALZER B.; BATORY D.; LAUNCHBURY J.; PARNAS D.; TARR P. "Modularity in the new millenium: a panel summary". In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING (ICSE'03), 25., Portland (Estados Unidos), maio 2003. *Proceedings of...* Portland (Estados Unidos), 2003.

KON F. Visão geral de alguns métodos ágeis XP, Scrum e Crystal. Disponível em: < http://agilcoop.incubadora.fapesp.br/portal/agilvideo>. Acessado em: 7 set. 2007.

KOSHY J.; PANDEY R. "VM* synthesizing scalable runtime environments for sensor networks". In: ACM CONFERENCE ON EMBEDDED NETWORKED SENSOR SYSTEMS (SENSYS 2005), 3., 2005, San Diego-CA (Estados Unidos). *Proceedings of...* San Diego-CA (Estados Unidos):ACM Press, 2005.

LEVIS P.; LEE N.; WELSH M.; CULLER D. "TOSSIM: accurate and scalable simulation of entire TinyOS applications." In: ACM CONFERENCE ON EMBEDDED NETWORKED SENSOR SYSTEMS (SENSYS 2003)., 1., 2003, Los Angeles-CA (Estados Unidos). *Proceedings of ...*Los Angeles-CA (Estados Unidos), 2003.

LINS A.; NAKAMURA E.; ROCHA L.; LOUREIRO A.; COELHO JR C. "Semi-automatic generation of monitoring applications for wireless networks". In: IEEE INTERNATIONAL CONFERENCE ON EMERGING TECHNOLOGIES AND FACTORY AUTOMATION (ETFA'03), 9., 2003, Lisboa (Portugal). *Proceedings of ...* Lisboa (Portugal), 2003.

LINZ T., KALLMAYER C., ASCHENBRENNER R., REICHL H. "Fully integrated EKG shirt based on embroidered electrical interconnections with conductive yarn and miniaturized flexible electronics". In: INTERNATIONAL WORKSHOP ON WEARABLE AND IMPLANTED BODY SENSOR NETWORKS (BSN'06), 3., Boston (Estados Unidos), 2006. *Proceedings of…* Boston (Estados Unidos): IEEE Computer Society, 2006. ISBN 0-7695-2547-4/06.

LYMBERIS A.;DITTMAR A. *Advanced Wearable Health Systems and Applications - Research and Development Efforts in the European Union. IEEE Engineering in Medicine and Biology Magazine, Vol 26, Issue 3, May-June 2007.*

LORINCZ K.; MALAN D.; FULFORD-JONES T.; NAWOJ A.; CLAVEL V.; SHNAYDER V.; MAINLAND G.; MOULTON S.; WELSH M. *Sensor networks for emergency response: challenges and opportunities*. *IEEE Pervasive Computing,* IEEE CS and IEEE ComSoc, outubro-dezembro 2004.

LU W., WALKE B.; SHEN X. *Guest Editorial: 4G mobile comunications: torward open wireless architecture*. *IEEE Wireless Communications*, abril 2004.

MADDEN S.; FRANKLIN M. J.; HELLERSTEIN J. M.; HONG W. "TAG: a tiny aggregation service for ad-hoc sensor networks". ANNUAL SYMPOSIUM ON OPERATING SYSTEMS DESIGN AND IMPLEMENTATION (OSDI), 5., 2002, Boston-MA (Estados Unidos). *Proceedings of ...* Boston-MA (Estados Unidos), 2002.

MAES P. "Concepts and experiments in computational reflection". In: CONFERENCE ON OBJECT-ORIENTED PROGRAMMING SYSTEMS, LANGUAGES, AND APPLICATIONS (OOPSLA'87), 1987, Orlando (Estados Unidos). *Proceedings of...* Orlando (Estados Unidos), 1987.

MENDES, A. Arquitetura de software. 1 ed. Rio de Janeiro, Brasil: Campus, 2002. ISBN: 85-352-1013-X.

MICROCHIP – PIC18F2550. Especificação de produtos. Disponível em: < http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1335&dDocName=en010280>. Acessado em: 7 set. 2007.

MONTEMAGNO D. C. *Nanomachines: A roadmap for realizing the vision*. *Jornal of Nanoparticle Research,* vol. 3, 1–3, 2001.

MOSBERGER D.; JIN T. Httperf: *A Tool for Measuring Web Server Performance***.** *Performance Evaluation Review*, vol. 26, no. 3, dezembro 1998, Disponível em: <http://www.hpl.hp.co.uk/research/linux/httperf/docs.php>. Acessado em: 7 set. 2006.

MSPGCC – The GCC toolchain for the Texas Instruments MSP430 MCUs. Disponível em: < http://mspgcc.sourceforge.net/>. Acessado em: 7 set. 2007.

NIELSEN J. "Usability inspection methods" In: CONFERENCE ON HUMAN FACTORS IN COMPUTING SYSTEMS, 1995, Denver-CO (Estados Unidos). *Proceedings of*... Denver-CO (Estados Unidos):ACM Press, 1995.

OLIMEX - MSP430-P149 DEVELOPMENT BOARD - Especificação de produtos. Disponível em: < http://www.olimex.com/dev/index.html>. Acessado em: 7 set. 2006.

ORTIS R. S.; CARVALHO H. S.; DA ROCHA A. F.; COELHO JR. C. J. N.; NASCIMENTO F. A. O. Monitorização de sinais biomédicos em assistentes pessoais digitais. In: CONGRESSO BRASILEIRO DE INFORMÁTICA EM SAÚDE (CBIS), 9., 2004, Ribeirão Preto – SP. *Anais...* Ribeirão Preto – SP:USP, 2004.

PARADISO J. A; STARNER T. *Energy scavenging for mobile and wireless electronics*. *IEEE Pervasive Computing,* IEEE CS and IEEE ComSoc, janeiro-março 2005.

PARADISO J. A. **"**Systems for human-powered mobile computing". Annual conference on Design automation, 43rd, São Francisco (Estados Unidos), 2006. *Proceedings of* ... ACM Press, 2006.

PATHAK A.; PRASANNA V. K. "Issues in design a compilation framework for macroprogrammed networked sensor systems". In: INTERNATIONAL CONFERENCE ON INTEGRATED INTERNET AD HOC AND SENSOR NETWORKS, 6., Nice (França), 2006. *Proceedings of* ... Nice (França):ACM Press, 2006.

PEREIRA F. *Microcontroladores MSP430: teoria e prática*. 1 ed. SãoPaulo: Érica, 2005. ISBN 85-365-0067-0.

POWERCAST - Powercaster™ and Powerharvester™ modules. Especificação de produtos. Disponível em: < http://www.powercastco.com/index.php?page=products>. Acessado em: 7 set. 2007.

RAJU M. *Heart Rate and EKG Monitor using the MSP430FG439*. Texas Instruments Technical Report SLAA280, outubro, 2005. Disponível em: < http://focus.ti.com/lit/an/slaa280/slaa280.pdf#search=%22EKG%20%2B%20MSP430%20%2B%20RAJU%22>. Acessado em: 7 set. 2006.

RAVULA S.; KIM J. E.; STOERMER C. "Position paper: quality attributes in wireless sensor networks". In: IEEE WORKSHOP ON SOFTWARE TECHNOLOGIES FOR FUTURE EMBEDDED AND UBIQUITOUS SYSTEMS, 3., Seattle (Estados Unidos), 2005.

ROCHA A.; RABELO Jr. A.; SOUSA A.; XIMENES A.; LOBO N.; WERTHER J.; OLIVEIRA K.; ALIRIO M.; FREITAS M.; WERNECK V. "An expert system for diagnosis of acute myocardial infarction". In: SYMPOSIUM ON APPLIED COMPUTING, 1995, Nashville-TE (Estados Unidos). *Proceedings of...* Nashville-TE (Estados Unidos):ACM Press, 1995.

RÖMER K.; MATTERN F. *The design space of wireless sensor networks***.** *IEEE Wireless Communications*, Vol. 11 No. 6, p. 54-61, dez. 2004.

RUIZ L. B.; LOUREIRO A. A.; CORREIA L. H.; VIEIRA L. F., MACEDO D.; NAKAMURA E.; FIGUEIREDO C.; VIEIRA M. A.; BECHELANE H.; NOGUEIRA M.; SILVA JR. D. FERNADES A. Arquitetura de redes de sensores sem fio. In: Simpósio Brasileiro de Redes de Computadores (SBRC), 22., 2004, Gramado-RS. Anais... Gramado-RS:SBC, 2004.

SCHWIEBERT L.; GUPTA S.; WEINMANN J. "Research challenges in wireless networks of biomedical sensors". International Conference on Mobile Computing and Networking, 7th, 2001, Roma (Itália). *Proceedings of...*ACM, 2001.

SENE JR I. G.; BARBOSA T. M. G. DE A.; CARVALHO H. S.; DA ROCHA A. F.; NASCIMENTO F. A. O. Fusão de dados em uma rede de sensores sem fio do corpo

humano. In: CONGRESO ARGENTINO DE BIOINGENIERÍA (SABI), 15., 2005**,** Ciudad de Paraná (Argentina). *Anais***...** Ciudad de Paraná (Argentina): Universidad Nacional de Entre Ríos**,** 2005.

SENE JR I. G.; BARBOSA T. M. G. DE A.; CARVALHO H. S.; DA ROCHA A. F.; NASCIMENTO F. A. O. Avaliação de uma rede de sensores sem fios para obtenção da temperatura média do corpo humano. In: CONGRESSO BRASILEIRO DE ENGENHARIA BIOMÉDICA (CBEB), 20., 2006, São Pedro – SP. *Anais...* São Pedro – SP:USP, 2006.

SENE JR I. G.; BARBOSA T. M. G. DE A.; CARVALHO H. S.; DA ROCHA A. F.; NASCIMENTO F. A. O. Monitoração da temperatura corporal baseada em uma rede de sensores sem fios. In: CONGRESSO BRASILEIRO DE INFORMÁTICA EM SAÚDE (CBIS), 10., 2006, Florianópolis – SC. *Anais...,* Florianópolis – SC:SBIS, 2006b.

SILBERSCHATZ A; GALVIN P.; GAGNE G. *Sistemas operacionais: conceitos e aplicações*. 1 ed. Rio de Janeiro: Campus, 2001. ISBN 85-352-0719-8.

SINHA, Amit.; CHANDRAKASAN, Anantha. “Dynamic power management in sensor networks”. In: ILYAS M. e MAHGOUB I. *Handbook of Sensor Networks:Compact Wireless and Wired Sensing Systems*. 1. ed. CRC Press, 2005.

SHOHAN M.; KÓSA G. *Propulsion Method for Swimming Microrobots*. *IEEE Transactions on Robotics* 23(1), 137-150, fevereiro, 2007.

SOAP - Version 1.2 Part 0: Primer, 2003. Disponível em: <http://www.w3.org/TR/2003/REC-soap12-part0-20030624/>. Acessado em: 7 set. 2006.

SOONG R.K.; NEVES H.P.; SCHMIDT J.; MONTEMAGNO C.D. *Engineering Hybrid Nanoscience Devices Powered By Biomolecular Motors*. *Biomedical Microdevices* 3(1), 69–71, 2001.

SP – SPARK FUN ELETRONICS Bluetooth Modem – BlueSMiRF. Especificação de produtos. Disponível em:

<http://www.sparkfun.com/commerce/product_info.php?products_id=582>. Acessado em: 7 set. 2006.

STAA A. *Programação modular: desenvolvendo programas complexos de forma organizada e segura*. 1 ed. Campus, 2000. 690 p. ISBN: 85-352-0608-6.

STARK I. "Thermal energy harvesting with thermo life". In: INTERNATIONAL WORKSHOP ON WEARABLE AND IMPLANTED BODY SENSOR NETWORKS (BSN'06), 3., Boston (Estados Unidos), 2006. *Proceedings of…* Boston (Estados Unidos): IEEE Computer Society, 2006. ISBN 0-7695-2547-4/06.

SUN - The Source for Java Developers. Especificação de Produtos. Disponível em:<http://java.sun.com/>. Acessado em: 7 set. 2006.

SUNG M.; PENTLAND A. "*MIThril LiveNet: Health and Lifestyle Networking*". In: WORKSHOP ON APPLICATIONS OF MOBILE EMBEDDED SYSTEMS (WAMES'04), 2004, Boston (Estados Unidos). *Procedings of* ... Boston (Estados Unidos), 2004.

TI – TEXAS INSTRUMENTS MSP430F149. Especificação de produtos. Disponível em: < http://focus.ti.com/docs/prod/folders/print/msp430f149.html>. Acessado em: 7 set. 2006.
TI - Texas Instruments. - Features of the MSP430 Bootstrap Loader (Rev. C). Especificação de Produtos. Disponível em: <http://focus.ti.com/lit/an/slaa089d/slaa089d.pdf#search=%22Features%20of%20the%20MSP430%20Bootstrap%20Loader%22 >. Acessado em: 7 set. 2006b.

TOBS - THE OFFICIAL BLUETOOTH WIRELESS INFO SITE. Disponível em: <http://www.bluetooth.com/bluetooth/>. Acessado em: 7 set. 2006.

VIEIRA M. A. M.; DA SILVA JR. D. C.; COELHO JR. C. J. N.; DA MATA J. M. "Survey on wireless sensor network devices". In: IEEE INTERNATIONAL CONFERENCE ON EMERGING TECHNOLOGIES AND FACTORY AUTOMATION (ETFA'03), 9., 2003, Lisboa (Portugal). *Proceedings of* ... Lisboa (Portugal), 2003.

VENEZIANO. H. W. Estudo do comportamento eletromiográfico de superfície em atividades subaquáticas. 2006. 151 f. Tese (doutorado em Engenharia Elétrica) - Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília.

W3C. *SOAP Version 1.2 Part 1: Messaging Framework.* Disponível em: <http://www.w3.org/TR/2003/REC-soap12-part1-20030624/>. Jun. 2003. Acessado em: 7 set. 2006.

W3C. *Web Services Architecture*. Disponível em: <http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/> . Fev. 2004. Acessado em: 7 set. 2006.

W3C. *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*. Disponível em: < http://www.w3.org/TR/wsdl20/> . Mar. 2006. Acessado em: 7 set. 2006.

WALD L. *Some terms of reference in data fusion. IEEE Transactions on Geoscience and Remote Sensing*, vol.37, no.3, maio 1999.

WELSH M.; LORINCZ K.; MALAN D.; FULFORD-JONES T.; NAWOJ A.; CLAVEL V.; SHNAYDER V.; MAINLAND G.; MOULTON S. *Sensor Networks for Emergency Response: Challenges and Opportunities*. *IEEE Pervasive Computing*, out-/dez 2004.

WELSH M.; MAINLAND G. **"**Programming Sensor Networks Using Abstract Regions". In: USENIX/ACM SYMPOSIUM ON NETWORKED SYSTEMS DESIGN AND IMPLEMENTATION (NSDI '04), 1., 2004, San Francisco-CA (Estados Unidos). *Proceedings of* ... San Francisco-CA (Estados Unidos):USENIX Press, 2004.

WELSH M.; GAO T.; GREENSPAN D.; JUANG R.; ALM A. "Vital signs monitoring and patient tracking over a wireless network". In: ANNUAL INTERNATIONAL CONFERENCE IEEE ENGINEERING IN MEDICINE AND BIOLOGY SOCIETY (EMBC), 27., 2005, Shangai (China). *Proceedings of…* Shangai (China):IEEE-EMBS, 2005.

WELSH M. "CodeBlue: Wireless Sensor Networks for Medical Care". Disponível em: <http://www.eecs.harvard.edu/~mdw/proj/codeblue/>. Acessado em: 7 set. 2006.

WELSH M; BAIRD S.; DAWSON-HAGGERTY S.; MYUNG D.; GAYNOR M.; MOULTON S. “Communicating Data from Wireless Sensor Networks using the HL7v3 Standard”. In: INTERNATIONAL WORKSHOP ON WEARABLE AND IMPLANTED BODY SENSOR NETWORKS (BSN’06), 3., Boston (Estados Unidos), 2006b. *Proceedings of…* Boston (Estados Unidos): IEEE Computer Society, 2006b. ISBN 0-7695-2547-4/06.

WHITEHOUSE K; JIANG F; WOO A; KARLOF C; CULLER D. *Sensor field localization: a deployment and empirical analysis*. UC Berkeley Technical Report UCB//CSD-04-1349, 9 abril, 2004. Disponível em: <http://www.cs.berkeley.edu/~culler/papers/>. Acessado em: 7 set. 2006.

WSA - Web Services Architecture, 2004. Disponível em: < http://www.w3.org/TR/ws-arch/>. Acessado em: 7 set. 2006.

WSDL - Web Services Description Language (WSDL) Version 2.0 Part 0: Primer, 2006. Disponível em: < http://www.w3.org/TR/wsdl20-primer/>. Acessado em: 7 set. 2006.

YANG, G. “Body Sensor Networks”. Disponível em <http://ubimon.doc.ic.ac.uk/bsn/index.php?m=206> . Acessado em: 7 set. 2006.

YANG, G. *Body Sensor Networks*. 1 ed. London, England: Springer-Verlag, 2006b. ISBN-10: 1-84628-272-1

ZA - ZIGBEE ALLIANCE. Disponível em: <http://www.zigbee.org/en/index.asp>. Acessado em: 7 set. 2006.

APÊNDICES

APÊNDICE A – PRODUÇÃO CIENTÍFICA DO PERÍODO DE DOUTORAMENTO

A1 – Prêmio

1. Melhor trabalho na categoria Trabalho em Andamento pelo artigo intitulado Compilador Inteligente para *Body Sensor Networks*. O prêmio foi concedido pela Sociedade Brasileira de Qualidade de Software durante o Workshop de Informática Médica (WIM) em 2007.

A2 – Periódico

1. **BARBOSA T. M. G. DE A.**, SENE JR I. G., CARVALHO H. S., DA ROCHA A. F. E NASCIMENTO F. A. O. Uma metodologia para programação de redes de sensores para monitoração do corpo humano. Artigo aceito para publicação na Revista Brasileira de Engenharia Biomédica. Submetido em outubro de 2006 e aceito em 3 de dezembro de 2007.

A3 – Capítulos de livros

1. **BARBOSA T. M. G. DE A.**, SENE JR I. G., CARVALHO H. S., DA ROCHA A. F. E NASCIMENTO F. A. O. "Programming Body Sensor Networks". Capítulo de livro aceito para publicação em *ENCYCLOPAEDIA OF HEALTHCARE INFORMATION SYSTEMS*, IDEA Group Inc. (em fase de edição com previsão para março de 2008).

2. **BARBOSA T. M. G. DE A.**, SENE JR I. G., CARVALHO H. S., DA ROCHA A. F. E NASCIMENTO F. A. O. As Redes de Sensores e o Monitoramento da Saúde Humana. Capítulo de livro aceito para publicação pela editora UNIVERSA da Universidade Católica de Brasília (em fase de edição com previsão para abril de 2008).

A4 – Congressos Internacionais

1. **BARBOSA, T. M. G. DE A.**, SENE JR, IWENS GERVÁSIO, SAMPAIO, HERVALDO CARVALHO ; ROCHA, ADSON FERREIRA DA ; NASCIMENTO, FRANCISO ASSIS DE O ; CARVALHO A. L. J . "A New Model for Programming Software in Body Sensor Networks". In: *Annual International Conference of the IEEE EMBS*, 29., 2007, Lyon (França). *Proceedings of* ... Piscataway NJ: IEEE Engineering in Medicine and Biology Society, 2007.

2. JOSÉ, ALEXANDRE B, **BARBOSA, T. M. G. A.,** CARVALHO, HERVALDO SAMPAIO, ROCHA, ADSON FERREIRA DA, NASCIMENTO, FRANCISCO DE ASSIS OLIVEIRA. "A Framework for Automated Evidence Gathering with Mobile Systems Using Bayesian Networks". In: *Annual International Conference of the IEEE EMBS*, 29., 2007, Lyon (França). *Proceedings of* ... Piscataway NJ : IEEE Engineering in Medicine and Biology Society, 2007.

3. **BARBOSA T. M. G. DE A.**; SENE JR I. G.; CARVALHO H. S.; DA ROCHA A. F.; NASCIMENTO F. A. O.; CAMAPUM J. F. "Application-oriented programming model for sensor networks embedded in the human body". In: *Annual International Conference IEEE Engineering in Medicine and Biology Society (EMBC)*, 28., 2006, Nova Iorque (Estados Unidos). *Proceedings of…* Nova Iorque (Estados Unidos):IEEE-EMBS, 2006.

4. JOSÉ, A. B.; DOS REIS, M. C; CAMAPUM J. F; VASCONCELOS, D. F.; **BARBOSA T. M. G. DE A**.; CARVALHO H. S.; DA ROCHA A. F.; "Classification and retrieval of medical images in an integrated healthcare environment". In: *Annual International Conference IEEE Engineering in Medicine and Biology Society (EMBC)*, 28., 2006, Nova Iorque (Estados Unidos). *Proceedings of…* Nova Iorque (Estados Unidos):IEEE-EMBS, 2006.

A5 – Congressos Nacionais (Brasil)

1. **BARBOSA, T. M. G. A.,** ROCHA, ADSON FERREIRA DA, SAMPAIO, HERVALDO CARVALHO, SENE JR, IWENS G ; NASCIMENTO E FRANCISCO A O DO. Compilador Inteligente para *Body Sensor Networks*. In: VII Workshop de Informática Médica, evento do SBQS 2007 - VI Simpósio Brasileiro de Qualidade de Software, 2007, Porto de Galinhas-PE. Anais do VII Workshop de Informática Médica, 2007.

2. SENE JR, IWENS G, **BARBOSA, T. M. G. A., ROCHA**, ADSON FERREIRA DA ; SAMPAIO, HERVALDO CARVALHO E NASCIMENTO, FRANCISCO A O DO. Uma Linguagem de Fusão de Dados para o Corpo Humano. In: VII Workshop de Informática Médica, evento do VI Simpósio Brasileiro da Qualidade de Software, 2007, Porto de Galinhas - PE. VII Workshop de Informática Médica, evento do VI Simpósio Brasileiro da Qualidade de Software, 2007.

3. **BARBOSA T. M. G. DE A.,** SENE JR I. G., CARVALHO H. S., DA ROCHA A. F. E NASCIMENTO F. A. O. Programação de uma rede de sensores para o corpo humano por meio de uma interface gráfica. In: CONGRESSO BRASILEIRO DE INFORMÁTICA EM SAÚDE (CBIS), 10., 2006, Florianópolis – SC. *Anais...,* Florianópolis – SC:SBIS, 2006.

4. GUTIÉRREZ E. M., FRANÇA R. D., **BARBOSA T. M. G. DE A.**, CARVALHO H. S. E DA ROCHA A. F. Desenvolvimento de uma rede de sensores sem fios para o monitoramento biomédico. In: CONGRESSO BRASILEIRO DE INFORMÁTICA EM SAÚDE (CBIS), 10., 2006, Florianópolis – SC. *Anais...,* Florianópolis – SC:SBIS, 2006.

5. SENE JR I. G.; **BARBOSA T. M. G. DE A**.; CARVALHO H. S.; DA ROCHA A. F.; NASCIMENTO F. A. O. Avaliação de uma rede de sensores sem fios para obtenção da temperatura média do corpo humano. In: CONGRESSO BRASILEIRO DE ENGENHARIA BIOMÉDICA (CBEB), 20., 2006, São Pedro – SP. *Anais...* São Pedro – SP:USP, 2006.

6. SENE JR I. G.; **BARBOSA T. M. G. DE A.**; CARVALHO H. S.; DA ROCHA A. F.; NASCIMENTO F. A. O. Monitoração da temperatura corporal baseada em uma rede de sensores sem fios. In: CONGRESSO BRASILEIRO DE INFORMÁTICA EM SAÚDE (CBIS), 10., 2006, Florianópolis – SC. *Anais...,* Florianópolis – SC:SBIS, 2006.

7. **BARBOSA, T. M. G. DE A.**; SENE JR., I. G.; CARVALHO, H. S.; DA ROCHA, A. F.; NASCIMENTO F. A. O. Arquitetura de software para redes de sensores sem fios: a proeminência do middleware. In: SEMINÁRIO INTEGRADO DE SOFTWARE E HARDWARE (SEMISH), 32., 2005, São Leopoldo – RS. *Anais do XXV Congresso da Sociedade Brasileira de Computação (CSBC)*, São Leopoldo – RS:UNISINOS, 2005.

8. JOSÉ, A. B.; **BARBOSA, T. M. G. DE A.**; CASTRO, L. S. S; SENE JR, I. G.; CARVALHO, H. S.; DA ROCHA, A. F; NASCIMENTO, F. A. O. Implementação da revisão sistemática de sintomas em sistemas móveis utilizando redes bayesianas. In: WORKSHOP DE INFORMÁTICA MÉDICA (WIM), 5., 2005, Porto Alegre – RS. *Anais...* Porto Alegre – RS:PUC, 2005.

9. CASTRO, L. S. S.; **BARBOSA, T. M. G. DE A**.; SENE JR, I. G. L; BRANISSO P. J. H.; FIGUEIREDO E. C.; CARVALHO, H. S.; DA ROCHA, A. F.; NASCIMENTO, F. A. O. HandMed – um sistema móvel integrado para captura automática de sintomas. In: CONGRESSO BRASILEIRO DE INFORMÁTICA EM SAÚDE (CBIS), 9., 2004, Ribeirão Preto – SP. *Anais...* Ribeirão Preto – SP:USP, 2004.

10. **BARBOSA, T. M. G. DE A.**; SENE JR, I. G.; CASTRO, L. S. S.; BRANISSO P. J. H.; FIGUEIREDO E. C.; CARVALHO, H. S.; DA ROCHA, A. F.; NASCIMENTO, F. A. O. Sistema móvel para monitoração da saúde: algoritmo para captura inteligente de sintomas. In: CONGRESSO BRASILEIRO DE INFORMÁTICA EM SAÚDE (CBIS), 9., 2004, Ribeirão Preto – SP. *Anais...* Ribeirão Preto – SP:USP, 2004.

A6– Congressos Nacionais (Argentina)

1. SENE JR I. G.; **BARBOSA T. M. G. DE A.**; CARVALHO H. S.; DA ROCHA A. F.; NASCIMENTO F. A. O. Fusão de dados em uma rede de sensores sem fio do corpo humano. In: CONGRESO ARGENTINO DE BIOINGENIERÍA (SABI), 15., 2005, Ciudad de Paraná (Argentina). *Anais*... Ciudad de Paraná (Argentina): Universidad Nacional de Entre Ríos, 2005.

2. CASTRO, L. S. S; BRANISSO P. J. H.; FIGUEIREDO E. C.; CARVALHO H. S.; **BARBOSA T. M. G. DE A.**; SENE JR I. G.; ROCHA A. F.; NASCIMENTO F. A. O. Automatização na captura de sintomas – handmed. In: CONGRESO ARGENTINO DE BIOINGENIERÍA (SABI), 15., 2005, Ciudad de Paraná (Argentina). *Anais*... Ciudad de Paraná (Argentina): Universidad Nacional de Entre Ríos, 2005.