GERALDO ANTÔNIO FERREIRA

APLICAÇÕES MIDP EM APARELHOS MÓVEIS CELULARES E MONITORAMENTO REMOTO DE BIO-SINAIS: CONSIDERAÇÕES E DESENVOLVIMENTO DE UMA SOLUÇÃO

Belo Horizonte

12 de Agosto de 2005

GERALDO ANTÔNIO FERREIRA

APLICAÇÕES MIDP EM APARELHOS MÓVEIS CELULARES E MONITORAMENTO REMOTO DE BIO-SINAIS: CONSIDERAÇÕES E DESENVOLVIMENTO DE UMA SOLUÇÃO

Dissertação apresentada ao Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação

Belo Horizonte

12 de Agosto de 2005

Agradecimentos

Agradeço à Universidade Federal de Minas Gerais, instituição onde cursei a graduação em Engenharia e o mestrado em Computação. Vindo de uma família sem recursos, eu não conseguiria de outra forma.

Ao Claudionor, meu orientador, por acreditar em nosso projeto.

Ao Alla, André Luiz Lins de Aquino, por me ajudar na revisão.

Ao pessoal da secretaria do DCC, particularmente à Renata e Sheila.

Por suportar minha ausência constante nos últimos meses, agradeço à Patrícia e ao pessoal de casa: minha mãe, irmão Gilbert, irmãs Gilmara e Gisele. E também aos cunhados e sobrinhas.

Resumo

A telemedicina tem experimentado grandes avanços nos últimos anos. Inovações como a transmissão por fibra-óptica e a telefonia móvel celular têm contribuído para isto. Outra fonte de melhoramentos tem vindo da microeletrônica. Dispositivos cada vez menores, dentre eles os sensores, os sistemas embutidos e os aparelhos celulares tem possibilitado ferramentas que proporcionam precisão, mobilidade e conforto durante procedimentos de avaliação médica.

Considerando tudo isto, nosso trabalho propõe uma solução de monitoramento médico à distância baseada no emprego de um aparelho celular como interface de rede na aquisição dos sinais. No aparelho celular desenvolvemos uma aplicação Java-MIDP que desempenha funções de interface de usuário, recebimento dos sinais, armazenamento dos mesmos e, finalmente, envio para um servidor no hospital. O envio dos dados é feito por uma conexão HTTP, acionada pela aplicação MIDP e suportada pela rede de dados celular. Para *debugging* do código fonte Java, simulação e geração do pacote executável, utilizamos as ferramentas *J2ME Wireless Toolkit*, da Sun, e *Nokia Developer's Suite*, da Nokia. Os resultados foram bastante satisfatórios nas simulações e implementação real, ocorrida no aparelho Nokia modelo 6600.

Assim, vislumbramos que o emprego de aparelhos celulares com aplicações Java em atividades de avaliação médica tende a ser um campo fértil nos próximos anos, para desenvolvedores, fabricantes e operadoras de telefonia celular.

Abstract

The telemedicine has improved a lot in the last few years. Innovations like fiber-optic and the cellular network has given important contributions to that. Another source of improvements has been obtained from microelectronics. Smaller devices, as for instance sensors, embedded systems and cell phones have provided the means for the development of tools that give accuracy, mobility and comfort during medical evaluation proceedings.

With this in mind, our work suggest a remote monitoring solution based on the employment of a mobile phone as network interface in the acquisition of signals. To get there, we developed an Java-MIDP application in the mobile phone that supply functions of user interface, receiving of signals, storage of them and, finaly, the sending to a server in the hospital. The sending of data is made by a HTTP connection, triggered by the MIDP application and supported by data cellular network. For debugging of Java source code, simulation and generation of the executable file, we used the tools *J2ME Wireless Toolkit,*of Sun, e *Nokia Developer's Suite*, of Nokia. The results were pretty satisfactory in the emulations and real execution, done in mobile phone Nokia 6600.

Therefore we have an expectation that the employment of mobile phone with Java applications in the medical evaluation activities might be a growing field in next years, for developers, makers and cellular operators.

Sumário

Capítulo 1 – Introdução 1
1.1 Motivação e Objetivos 2
1.2 Requisitos da Solução 3
1.3 Trabalhos Relacionados 4
1.3.1 The Ambulance Project 4
1.3.2 MOMEDA 5
1.3.3 DITIS 6
1.4 Estrutura do Trabalho 7
Capítulo 2 – Contextualização 8
2.1 Telemedicina 9
2.2 Rede de Telefonia Móvel Celular 10
2.2.1 Histórico 10
2.2.2 Redes de Geração 2 e 2.5 11
2.2.3 Redes de Geração 3 12
2.2.4 Rede Celular GSM-GPRS : Apresentação 14
2.2.5 Rede Celular GSM 14
2.2.6 Rede Celular GPRS 17
2.3 Programação MIDP - MIDlets 20
2.3.1 CLDC 20
2.3.2 MIDP 21
2.3.3 MIDlets 27
2.4 Symbian OS 29
2.4.1 Arquitetura do Symbian OS 31
Capítulo 3 – Monitoramento Remoto de Bio-Sinais 32
3.1 Apresentação 32
3.2 Módulo de Coleta 34
3.2.1 Módulo ECG 35
3.2.2 Eletrodos 37
3.2.3 ECG ASIC 38
3.2.4 ADC 38
3.2.5 uC 8051 38

3.3 Aparelho Celular MIDP 40
3.3.1 Aparelho Nokia 6600 40
3.3.2 MIDlet ECG 41
3.3.3 Método *recebe* 44
3.3.4 Método *mostra_dados* 47
3.3.5 Método *envia* 49
3.3.6 Método *recebe_envia* 51
3.4 Servidor Destino 54
3.4.1 Servidor Destino – Aplicativo PHP 54
3.4.2 Servidor Destino – Aplicativo Java 55
3.5 Infra-Estrutura de Rede 56
Capítulo 4 – Resultados 59
4.1 Simulação no Sun J2ME Wireless Toolkit 59
4.1.1 Network Monitor 60
4.2 Simulação no Nokia Developer's Suíte 61
4.3 Execução Real no Aparelho Nokia 6600 62
Capítulo 5 – Conclusões e Trabalhos Futuros 64
5.1 Dificuldades Encontradas 64
5.2 Conclusões 66
5.3 Trabalhos Futuros 67
Capítulo 6 – Referências Bibliográficas 69
Capítulo 7 – Apêndices 71
Apêndice A – Código .java do MIDlet ECG 71
Apêndice B – Especificações do Aparelho Nokia 6600 84
Apêndice C – Código .php do Servidor Destino 86
Apêndice D – Diagrama do Bloco ECG ASIC 87
Apêndice E – Diagrama do Bloco ADC 88
Apêndice F – Diagrama do Bloco uC 8051 89

Lista de Abreviações

Abreviação	Significado
1XRTT	CDMA Radio Transmission Technology (1 x 1,25 MHz)
ANATEL	Agência Nacional de Telecomunicações
API	Application Programming Interface
Auc	Authentication Center
BSC	Base Station Controller
BTS	Base Transceiver Station
CDMA	Code Division Multiple Access
CDMA EV-DO	CDMA Evolution Data Only ou Evolution Data Optimized
CLDC	Connected Limited Device Configuration
ECG	Eletrocardiograma
EIR	Equipment Identity Register
Gbps	Giga bits/segundo
GGSN	Gateway GPRS Support Node
GPRS	General Packet Radio Service
GSM	Global System for Mobile Communications
EDGE	Enhanced Data for GSM Evolution ou Global Evolution
HLR	Home Location Register
HTTP	Hypertext Transfer Protocol
I/O	Input/Output
IMEI	International Mobile Equipment Identity
IMSI	International Mobile Subscriber Identity
IP	Internet Protocol
ISDN	Integrated Services Digital Network
J2EE	Java 2 Enterprise Edition
J2ME	Java 2 Micro Edition
J2SE	Java 2 Standard Edition
JAD	Java Application Descriptor
JAR	Java Archive
JSP	Java Server Pages
Kbps	Kilo bits/segundo
KVM	KiloJava Virtual Machine
Mbps	Mega bits/segundo
MID	Mobile Information Device
MIDP	Mobile Information Device Profile
MMS	Multimedia Messaging Service
MSC	Mobile Switching Center
mW	miliWatt
PCU	Packet Control Unit
PDA	Personal Digital Assistant
PHP	Hypertext PreProcessor
RMS	Record Management System
SGSN	Serving GPRS Support Node
SIM	Subscriber Identity Module
SS7	Signalling System Number 7
TDMA	Time Division Multiple Access
UMTS	Universal Mobile Telecommunication Service
VLR	Visitor Location Register
VoIP	Voice Over IP
WAP	Wireless Application Protocol
W-CDMA	Wide-Band CDMA

Lista de Figuras

2-1 Clusters de Células 10

2-2 Evolução dos Padrões de Telefonia Móvel Celular 13

2-3 Arquitetura GSM 15

2-4 Arquitetura GPRS 17

2-5 Arquitetura MIDP 23

2-6 Classes de Interface do Usuário MIDP 25

2-7 Arquitetura Symbian OS v7.0s 31

3-1 Monitoramento Remoto de Bio-Sinais 32

3-2 Eletrodos e o Módulo de Eletrocardiograma 35

3-3 Vistas Frontal e Traseira do Módulo ECG 36

3-4 Arquitetura do Módulo ECG 37

3-5 Execução do MIDlet ECG no Aparelho Nokia 6600 41

3-6 Estrutura do MIDlet ECG 43

3-7 Método recebe 44

3-8 Método mostra_dados 47

3-9 Método envia 49

3-10 Método recebe_envia 53

3-11 Servidor Destino das Medições de Eletrocardiograma 54

3-12 Gráfico do Eletrocardiograma Recebido no Servidor Destino 55

4-1 Simulação no Sun J2ME Wireless Toolkit 60

4-2 Simulação no Network Monitor 61

4-3 Simulação no Nokia Developer's Suíte 62

Capítulo 1 – Introdução

A comunicação móvel sofreu um grande avanço nas últimas décadas, particularmente a telefonia móvel celular, popularizando o uso do aparelho celular na maioria dos países. Paralelamente, um novo campo da medicina nasceu e vem se desenvolvendo rapidamente à sombra das telecomunicações: a telemedicina. Diferentes projetos pelo mundo têm utilizado as telecomunicações em suas soluções de auxílio à medicina.

Mais recentemente, o desenvolvimento da linguagem Java para dispositivos móveis, J2ME/MIDP, tem possibilitado a inclusão de aplicativos nos aparelhos móveis celulares, suportando diferentes funcionalidades. Por sua vez, aparelhos móveis baseados no sistema operacional *Symbian* permitem o emprego de uma consolidada biblioteca de APIs na programação MIDP. Tais aplicativos, desenvolvidos a partir de um aparelho com *Symbian*/MIDP podem ser empregados no suporte à telemedicina.

Nossa proposta conjuga estes elementos: aplicação MIDP desenvolvida em um aparelho celular *Symbian*, suportando uma atividade médica de monitoramento à distância.

1.1 Motivação e Objetivos

Aplicações de monitoramento médico à distância podem sem desenvolvidas utilizando-se diferentes tecnologias em diferentes cenários. Analisemos o seguinte exemplo fictício: um médico precisa acompanhar diariamente, a partir do hospital, o eletrocardiograma de um paciente que está em processo de recuperação pós-operatória, em sua própria casa. Considerando que o paciente está em repouso em sua cama, uma solução simples seria o desenvolvimento de um módulo que coletasse as medidas de eletrocardiograma e as enviasse para o hospital, através de *modems* ou através da rede internet. Esta solução utilizaria cabos e o paciente não teria mobilidade, ele estaria confinado à cama.

Uma boa melhoria deste cenário seria o desenvolvimento de um outro módulo que funcionasse agora em uma rede local sem-fio. Com este módulo o paciente poderia locomover-se dentro da casa, desde que haja cobertura de sinal por parte da rede local sem-fio, não interrompendo o monitoramento. Tal módulo seria mais sofisticado uma vez que, além da função de coleta do sinal de eletrocardiograma, ele teria que suportar a função de *host* da rede local sem-fio. Isto é, o módulo precisaria ter a funcionalidade de rádio da rede local, seja ela 802.11 ou *bluetooth*.

Agora considere uma última situação para o mesmo paciente. Suponha que ele precise sair de casa, sua situação clínica complicou-se e ele será levado de ambulância para o hospital. Justamente neste momento, quando em teoria ele mais precisa reportar seus sinais vitais, o monitoramento será perdido, uma vez que a ambulância sairá da área

de cobertura da rede local sem-fio, instalada na casa do paciente.

Nossa proposta de monitoramento remoto visa justamente cobrir tal situação. E a ferramenta central usada para atingir este objetivo é um aparelho celular. Através dele conseguimos transmitir a taxas satisfatórias as medições obtidas pelo módulo de coleta. E, o que não era possível com a rede local sem-fio, cobrimos uma área geográfica muito mais ampla, benefício da rede celular.

1.2 Requisitos da Solução

Nosso trabalho é sustentado por requisitos mínimos de projeto que determinaram a escolha de tecnologias específicas. Apresentamos a seguir alguns destes requisitos, acompanhados de suas respectivas tecnologias.

- Mobilidade do paciente.
    - Conexão *Bluetooth* e Rede de Telefonia Móvel Celular.
- Cobertura satisfatória para transmissões a partir de uma ambulância.
    - Rede de Telefonia Móvel Celular.
- Emprego de conexões HTTP em detrimento de conexões discadas.
    - Aplicação Java/MIDP em um Aparelho Celular.
- Plataformas suficientemente robustas para suportar aplicações médicas.
    - Aparelho Celular com *Symbian OS*.
- Portabilidade das aplicações em diferentes plataformas/fabricantes.
    - Programação Java/MIDP em um Aparelho Celular.

1.3 Trabalhos Relacionados

Como ainda é incipiente o desenvolvimento de aplicações MIDP para aparelhos celulares, não encontramos na literatura nenhuma que fosse relacionada especificamente ao monitoramento remoto de sinais vitais, ou mesmo à telemedicina em geral. Percebemos que a maioria dos aplicativos propostos fica restrita a jogos e troca de mensagens, seja de texto ou multimídia. Por outro lado, existe na literatura uma satisfatória quantidade de trabalhos que empregam as redes sem-fio com o intuito de suportar aplicações médicas, como pode ser comprovado em [1], [2], [3] e [4]. Sendo assim, apresentamos nas seções 1.3.1, 1.3.2 e 1.3.3 os trabalhos que mais forneceram subsídios de comparação para nossa proposta de monitoramento remoto.

1.3.1 The Ambulance Project

A oferta de pronto-atendimento médico especializado pode melhorar bastante os serviços de saúde de regiões de carência de pessoal capacitado, como áreas rurais ou longínquas. O suprimento de telemedicina de emergência e monitoramento doméstico de pacientes é a principal área de interesse de *The Ambulance Emergency 112 Project* [5], projeto europeu (Grécia, Itália, Suécia, Chipre e Espanha) patrocinado parcialmente pelo *European Commission/DGXIII Telematics Application Programme*. O projeto consta do desenvolvimento de um dispositivo de telemedicina portátil de emergência, que suporta a transmissão de bio-sinais críticos em tempo real, assim como imagens do paciente,

usando-se o enlace GSM. Este dispositivo pode ser usado por para-médicos ou pessoal instruído para tal. O sistema consta de 2 diferentes módulos: (1) a unidade móvel, que é localizado na ambulância, perto do paciente; e (2) a unidade de suporte remoto, que é localizada no hospital, e usada pelos médicos especializados para fornecer as instruções. O sistema permite tele-diagnóstico e suporte especializado a longa distância.

1.3.2 MOMEDA

Transmitir de maneira adequada e contínua as informações sobre os pacientes durante procedimentos médicos avançados (cirurgias, por exemplo) é uma tarefa importante na medicina moderna. Além disso, pacientes enquanto hospitalizados necessitam prosseguir com suas rotinas, ou pelo menos precisam ter acesso aos meios de comunicação modernos, como fax, e-mail e internet. Por outro lado, médicos em trânsito num hospital, ou fora dele, necessitam ter informações completas e atualizadas sobre os registros de seus pacientes, possibilitando a aplicação de procedimentos médicos mais adequados. Estas são as principais questões abordadas pelo projeto MOMEDA, *Mobile Medical Data* [6], também patrocinado parcialmente pelo *European Commission/DGXIII Telematics Application Programme*. O sistema consiste de dois módulos: (1) o módulo de informação do paciente e (2) o módulo de informação do médico. O objetivo principal do módulo de informação do paciente é possibilitar o acesso a informações específicas sobre o problema médico do paciente, procedimentos médicos planejados, e qual o estilo de vida o paciente deveria seguir durante a hospitalização, e após ela. O principal

objetivo do módulo de informação do médico é permitir ao médico ou outro profissional da saúde a consulta a registros médicos eletrônicos, por intermédio de um aparelho de comunicação pessoal *Nokia Communicator 9110* conectado à rede GSM. O usuário deste módulo consegue acessar o servidor do hospital, recebendo registros eletrônicos e imagens médicas, como a reprodução de uma radiografia, por exemplo. O projeto Momeda foi testado com sucesso em três países europeus: Finlândia, Itália e Grécia.

1.3.3 DITIS

Doenças crônicas e complexas, como o câncer, requerem o emprego de protocolos de tratamento especializados, administrados e monitorados por uma equipe coordenada de profissionais. Tratamentos baseados na própria residência do paciente de doença crônica, realizado por uma equipe de profissionais, é freqüentemente uma necessidade, devido ao estendido período da doença. Por outro lado, o tratamento do doente crônico realizado no hospital é mais curto, restringindo-se a intervenções pontuais. Como não é possível para a equipe médica estar fisicamente com o paciente durante todo o tempo do tratamento, o principal benefício deste projeto é contornar este problema, através de uma rede para colaboração médica chamada DITIS, *Collaborative Virtual Medical Team for Home Heathcare of Cancer Patients* [7]. DITIS é um sistema que suporta equipes médicas virtuais que trabalham no atendimento domiciliar de pacientes de câncer em Chipre. Ele suporta a criação, gerenciamento, e coordenação de equipes médicas virtuais, para o tratamento contínuo de pacientes em suas próprias residências. Se necessário DITIS também pode coordenar visitas das equipes médicas a

centros especializados de tratamento. O projeto e desenvolvimento do sistema são baseados na conectividade entre Internet e a rede GSM/WAP. O projeto DITIS é composto por um sistema de servidores que suportam o armazenamento dos registros médicos e a comunicação com os membros da rede. Os membros da equipe médica virtual, e também os pacientes, podem se comunicar através de diferentes meios, como computadores pessoais conectados à Internet, PDAs conectados à rede local sem-fio, ou aparelhos celulares com conexão GSM.

1.4 Estrutura do Trabalho

O texto está dividido em sete capítulos. O capítulo um, Introdução, apresenta uma visão geral de nosso trabalho, descrevendo a motivação, objetivos, requisitos e trabalhos relacionados. O capítulo dois, Contextualização, detalha as tecnologias que utilizamos, entre elas a telemedicina, a rede celular, o MIDP e o *Symbian OS*. O capítulo três, Monitoramento Remoto de Bio-Sinais, descreve a implementação de nossa solução. O capítulo quatro, Resultados, mostra os resultados alcançados por nossa solução. O capítulo cinco, Conclusões e Trabalhos Futuros, sugere os melhoramentos que podem ser aplicados ao nosso trabalho. O capítulo seis, Referências Bibliográficas, lista a bibliografia empregada. Finalmente, no capítulo sete, estão incluídos os apêndices.

Capítulo 2 – Contextualização

Neste capítulo iremos discorrer sobre os principais elementos presentes em nossa solução de monitoramento remoto. Primeiramente, na seção 2.1, apresentaremos a telemedicina e seus principais ramos. Em seguida, na seção 2.2 abordaremos a rede celular, seus conceitos básicos, seus principais padrões comerciais e evolução dos mesmos. Um detalhamento maior é fornecido à rede GSM-GPRS, empregada por nossa solução. Uma visão geral da programação Java para dispositivos móveis é mostrada na seção 2.3, com especial atenção aos MIDlets, aplicativos MIDP. Por fim, na seção 2.4, forneceremos uma rápida apresentação do sistema operacional *Symbian*.

2.1 Telemedicina

A telemedicina pode ser definida como o fornecimento de cuidados médicos e o compartilhamento do conhecimento médico à distância, empregando-se as telecomunicações. Nos últimos anos temos observado uma grande expansão da telemedicina. A modernização das redes de telefonia e a proliferação das redes de dados e redes sem-fio são fatores que tem contribuído para isto. Um outro fomento tem vindo da tecnologia de transmissão por fibra óptica, que suporta indiretamente as outras redes e atualmente proporciona velocidades da ordem de dezenas de Gbps, conforme [17]. Talvez um grande marco desta evolução tenha sido a conjugação da robótica na telemedicina, possibilitando o incrível feito de uma cirurgia à distância.

De acordo com [1], [2], [3] e [4], aplicações de telemedicina têm expandido diversas especialidades médicas, entre elas:

- cardiologia em telecardiologia
- radiologia em teleradiologia
- dermatologia em teledermatologia
- psiquiatria em telepsiquiatria

Tais aplicações têm possibilitado o pronto-atendimento médico em áreas carentes de pessoal especializado, como áreas rurais, ambulâncias, trens, barcos e aviões. Por fim, aplicações de telemedicina podem também possibilitar o monitoramento doméstico de pessoas idosas ou em pós-operatório.

2.2 Rede de Telefonia Móvel Celular

2.2.1 Histórico

A primeira chamada telefônica celular foi realizada 1973 e é creditada a Martin Cooper, pesquisador norte-americano da Motorola. O aparelho móvel celular pesava 1 Kg e custou quatro mil dólares americanos. A partir daí veio o desenvolvimento dos primeiros sistemas de telefonia móvel celular, geração 1G, que foram implementados nos Estados Unidos, Europa e Japão, já no início da década de 1980. Eram baseados na tecnologia de rádios analógicos FM e já empregavam a técnica de divisão da área geográfica em *clusters* de células, permitindo uma otimização do espectro de freqüências disponível [8]. A figura 2-1 mostra uma configuração de 4 *clusters,* sendo que cada um é composto por 7 células. A quota *d* ilustra a chamada "distância de reuso", encontrada entre células de mesma freqüência.

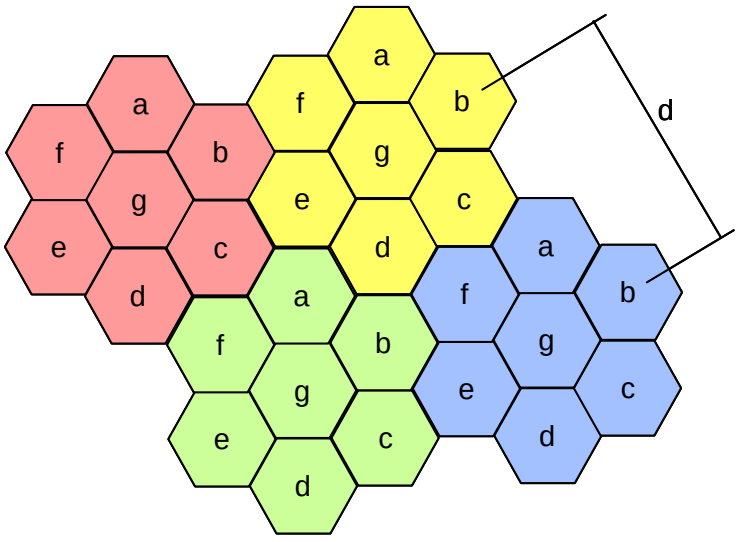


Figura 2-1: Clusters de Células

Diferentemente dos sistemas de rádio empregados até então (taxistas, caminhoneiros, rádio-amador, etc), o sistema de telefonia móvel celular possibilitou a interligação de seus usuários ao sistema de telefonia fixa. Isto permitia chamadas de telefone móvel para telefone fixo e vice-versa, o que foi um grande avanço. As pessoas teriam agora mobilidade completa e seriam encontradas a partir de um telefone fixo, instalado em qualquer parte do mundo. Uma definição mais completa de telefonia móvel celular é fornecida pela ANATEL em [12]: “Serviço móvel celular é o serviço de telecomunicações móvel terrestre, aberto à correspondência pública, que utiliza sistema de radiocomunicações com técnica celular, interconectado à rede pública de telecomunicações, e acessado por meio de terminais portáteis, transportáveis ou veiculares, de uso individual”.

2.2.2 Redes de Geração 2 e 2.5

No início da década de 1990 a tecnologia de codificação digital para os rádios impulsionou os padrões de geração 2 na telefonia móvel celular. Os padrões de geração 2 eram: o TDMA Norte-Americano (IS-136), o GSM Europeu, o PDC Japonês , e posteriormente o CDMA Norte-Americano (IS-95). Os mesmos possibilitaram um incremento do número de usuários na rede, além de uma considerável diminuição no tamanho dos aparelhos e baterias, entre outros melhoramentos. Apesar disto, estes padrões digitais não traziam o benefício de uma taxa de transmissão de dados satisfatória. Conforme mostrado na figura 2-2, o limite de velocidade para transmissão de dados nos

padrões de geração 2 era de 14,4 Kbps, valor bastante baixo para muitas aplicações de rede.

No início da década de 2000 as primeiras redes de telefonia celular de geração 2.5 começaram a ser implementadas. Os padrões desta geração foram o CDMA 1xRTT, o GSM GPRS e o GSM EDGE. Estes padrões descendiam diretamente do CDMA IS-95 e GSM respectivamente, simplificando bastante os *upgrades* nas operadoras. Com a geração 2.5 melhores taxas para transmissão de dados foram alcançadas. Na prática, dependendo das condições da rede, médias de 50 Kbps são alcançadas pelo GSM GPRS, contra 80Kbps do CDMA 1xRTT. Já o GSM EDGE pode alcançar 384 Kbps teóricos, e é utilizado em áreas específicas de cobertura, chamadas *Spots*. Estas velocidades de transmissão proporcionaram um grande avanço no desenvolvimento de aplicativos para os aparelhos celulares. Aplicações como *minibrowser*, *e-mail* e MMS puderam ser incorporadas aos aparelhos móveis.

2.2.3 Redes de Geração 3

A Geração 3 da telefonia móvel celular tem a intenção de cobrir serviços de dados de alta capacidade, tais como:

- Acesso à Internet a taxas acima de 1 Mbps
- Voz sobre IP, VoIP
- Transmissão ao vivo de música
- Sessões interativas de acesso à rede

Os principais padrões da Geração 3 são atualmente o W-CDMA/UMTS e o CDMA EV-DO, ambos propondo uma taxa de transmissão teórica de 2,4 Mbps . A figura 2-2 mostra a evolução da telefonia móvel celular, indicando em azul a taxa de transmissão teórica alcançada por cada padrão. Valores práticos são sensivelmente menores e dependem das condições da rede. A mesma figura também relaciona a geração a qual cada padrão pertence.

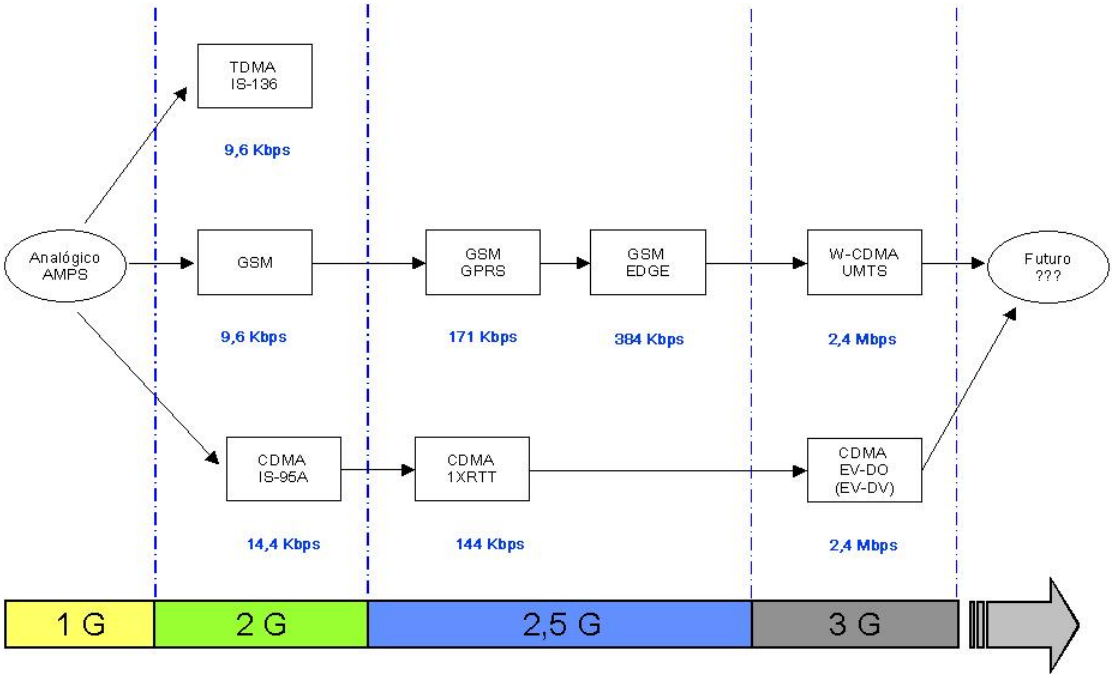


Figura 2-2: Evolução dos Padrões de Telefonia Móvel Celular

2.2.4 Rede Celular GSM-GPRS : Apresentação

Detalharemos nas seções 2.2.5 e 2.2.6 os padrões GSM e GPRS, uma vez que nossa implementação faz uso destas tecnologias. A escolha não se baseou em critérios comerciais ou mesmo técnicos, e sim na disponibilidade deste padrão de Geração 2,5 na cidade de Belo Horizonte. O padrão CDMA 1xRTT não foi considerado, uma vez que não há oferta do mesmo por parte das operadoras da cidade de Belo Horizonte.

2.2.5 Rede Celular GSM

GSM foi o primeiro padrão digital da telefonia celular a operar comercialmente. Implementado na Europa, teve o objetivo de unificar os padrões emergentes do final da década de 1980. O GSM foi desenvolvido para suportar boa qualidade de voz, baixo custo, compatibilidade com o ISDN e *roaming* por toda a Europa. No entanto, o GSM cresceu espantosamente e hoje é utilizado por mais de 200 países em todo o mundo, de acordo com [8] e [18].

A figura 2-3 mostra a arquitetura simplificada de uma rede com o padrão GSM. Nela podemos perceber uma rede composta por 2 células, cada qual com sua própria estação BTS e aparelhos móveis. Por sua vez, as BTSs são conectadas à estação controladora BSC através de enlaces de protocolo aberto Abis. Finalmente temos a central MSC interligando a BSC à rede externa e às plataformas de serviços.

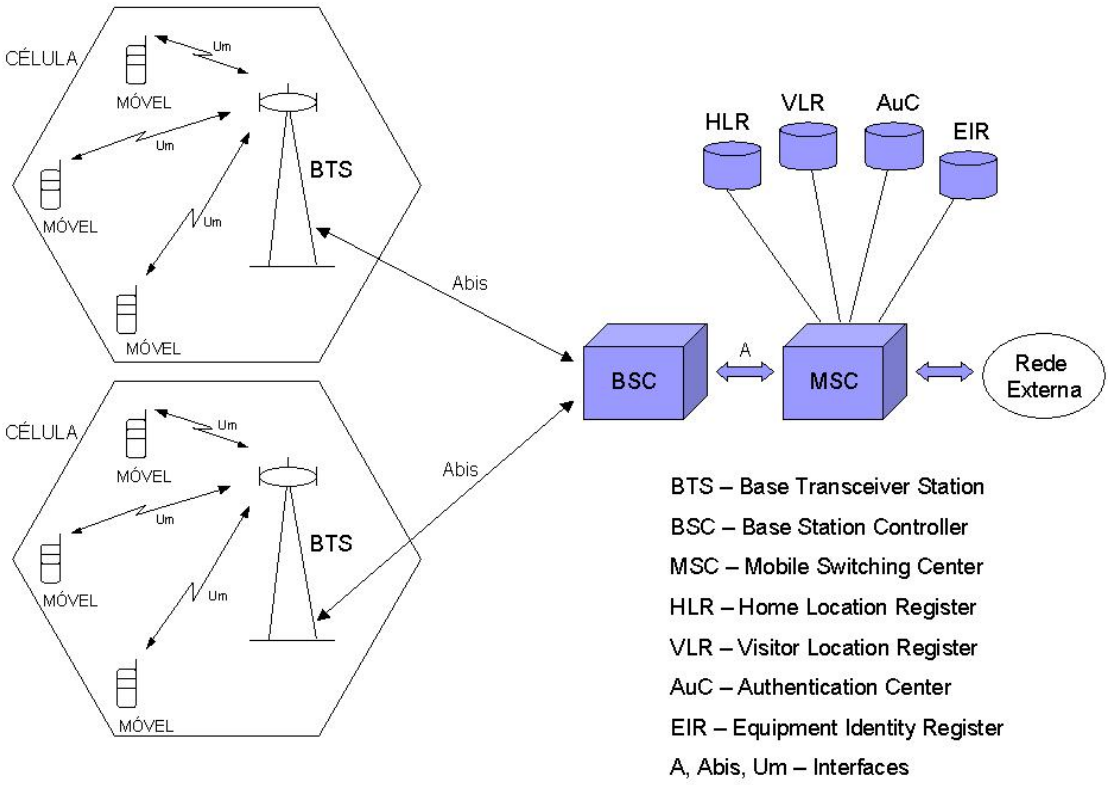


Figura 2-3: Arquitetura GSM

A seguir apresentaremos uma breve descrição sistêmica dos principais componentes da rede celular GSM ilustrados na figura 2-3.

- **Móvel:** aparelho utilizado pelo assinante. O mesmo contém o cartão SIM. Sem o cartão SIM o aparelho móvel não está associado a um usuário e não pode fazer nem receber chamadas. O cartão SIM armazena entre outras informações um número de 15 dígitos que identifica unicamente um assinante, denominado IMSI. Já o aparelho móvel é identificado pelo IMEI, atribuído pelo fabricante.
- **BTS**: elemento da rede GSM responsável pela comunicação com os aparelhos móveis dos assinantes. Contém os rádios e toda a infra-estrutura de torre e antenas

necessária à cobertura de radio-freqüência da célula GSM.

- **BSC**: elemento responsável pelo controle das BTSs de uma determinada área geográfica, assumindo funções de manutenção e gerência das mesmas. A BSC também tem a função de interface com a central de comutação MSC.
- **MSC**: é a central responsável pelas funções de comutação e sinalização dos aparelhos móveis localizadas em uma área geográfica designada como a área da MSC. A diferença principal entre uma MSC e uma central de comutação fixa é que a MSC tem que levar em consideração a mobilidade dos assinantes locais ou visitantes.
- **HLR**: é a base de dados que contém as informações sobre os assinantes de um sistema celular.
- **VLR**: é a base de dados que contém as informações sobre os assinantes em visita (*roaming*) a um sistema celular.
- **AuC**: é responsável pela autenticação dos assinantes no sistema. Ele armazena uma chave de identidade para cada assinante do HLR, possibilitando a autenticação do IMSI do assinante.
- **EIR**: é a base de dados que armazena os IMEIs dos aparelhos móveis de um sistema GSM.

2.2.6 Rede Celular GPRS

O GPRS oferece aos assinantes do GSM o acesso a aplicações de comunicações de dados como e-mail, redes corporativas e Internet. O serviço GPRS usa a rede GSM existente, incluindo novos equipamentos de rede de comutação de pacotes. Este serviço é uma evolução das redes GSM para comunicação de dados. A figura 2-4 ilustra uma rede com o padrão GPRS. Nela podemos perceber que a base GSM foi mantida, expandindo-se alguns elementos que proporcionam a comunicação por pacotes. Tais elementos serão detalhados ao final desta seção 2.2.6.

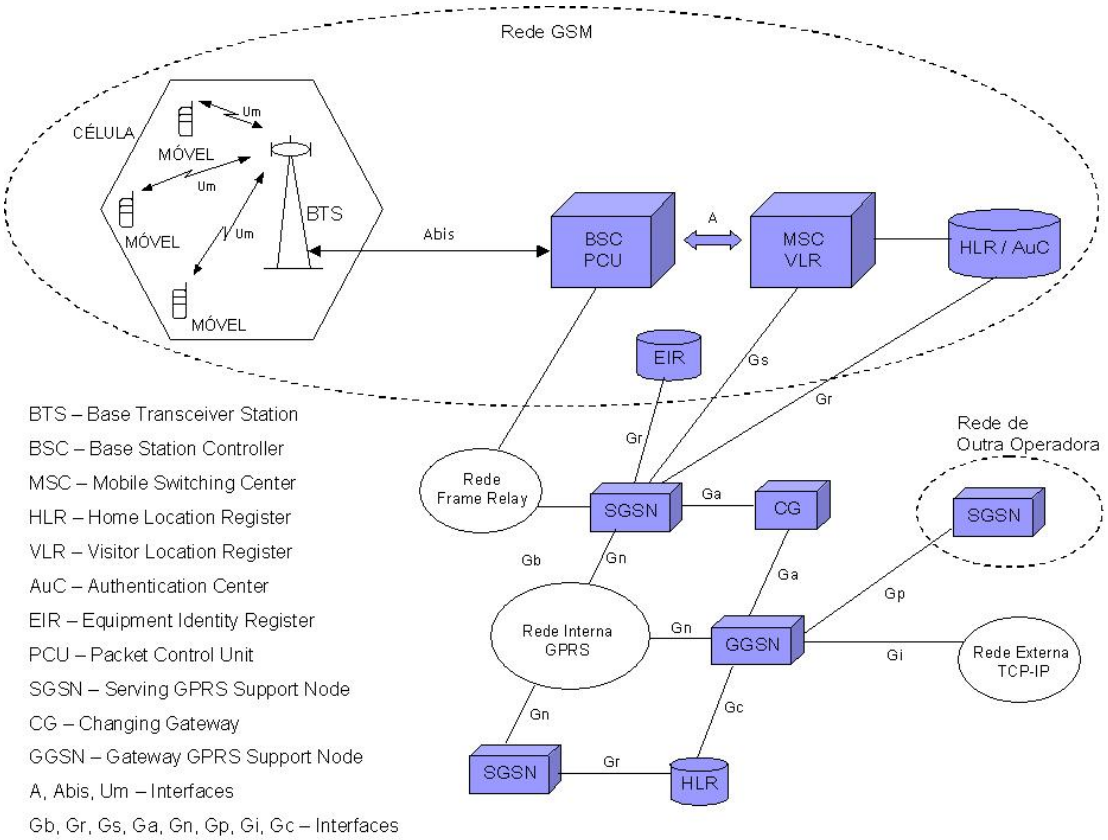


Figura 2-4: Arquitetura GPRS

As redes GSM existentes usam a tecnologia de comutação de circuitos para transferir informações, voz ou dados, entre usuários. Entretanto, o GPRS usa a comutação de pacotes, o que significa que não há circuitos dedicados atribuídos aos telefones móveis GPRS. Um canal físico é estabelecido dinamicamente, somente enquanto os dados estiverem sendo transferidos. Assim que os dados tiverem sido enviados, o recurso de um *timeslot* na interface aérea pode ser re-alocado a outros usuários, para tornar mais eficiente o uso da rede.

Quando os dados comutados por pacotes deixam a rede GPRS/GSM, eles são transferidos para as redes Internet ou X.25. Assim, o GPRS inclui novos procedimentos de transmissão e sinalização, assim como novos protocolos para a inter-operação com o mundo IP e outras redes de pacotes padrão.

Para operar com taxas de dados altas, o GPRS emprega novos esquemas de codificação de erro e múltiplos *timeslots* nas interfaces aéreas. Teoricamente, podemos ter uma taxa de dados máxima de 171,2 Kbps, usando oito *timeslots*. Entretanto, na prática, este valor cai substancialmente devido ao emprego de somente 4 *timeslots* e também devido às limitações da interface aérea. Médias de 50 Kbps são encontradas em campo.

Como a comunicação por comutação de pacotes permite a atribuição não contínua de recursos a um usuário, a tarifação GPRS não é baseada no tempo da conexão, e sim na utilização de recursos. Isto é, na quantidade de *bytes* enviados ou recebidos.

A seguir apresentaremos uma breve descrição sistêmica dos principais componentes da rede celular GPRS ilustrados na figura 2-4.

- **GGSN**: coloca um gateway entre a rede GPRS e a rede pública de pacotes (PDN) ou outras redes GPRS. O GGSN fornece funções de gerência de autenticação e localização, conecta-se ao HLR por meio da interface Gc e conta o número de pacotes transmitidos, para tarifar corretamente o assinante.
- **SGSN**: controla a conexão entre a rede e o aparelho móvel. O SGSN fornece a gerência da sessão e funções de gerência de mobilidade GPRS, como *handovers* e *paging*, sendo conectado ao HLR pela interface Gr e à MSC/VLR pela interface Gs. Além disso, também conta o número de pacotes roteados.
- **PCU**: funções de conversão dos dados em pacotes em um formato que possa ser transferido pela interface aérea, a gerência dos recursos de rádio e a implementação das medições de qualidade de serviço (QoS).
- **Interfaces:** os enlaces de sinalização entre os nós GPRS e os blocos GSM são interfaces SS7. A sinalização entre os nós GPRS é definida pelas especificações GPRS. Entre as novas interfaces físicas estão a interface Gb, que conecta o SGSN à PCU; a interface Gn, que conecta o GGSN e o SGSN; e as interfaces Gc, Gr e Gs, que transportam protocolos baseados no SS7.

2.3 Programação MIDP - MIDlets

2.3.1 CLDC

A CLDC, *Connected Limited Device Configuration*, cuja especificação está descrita em [10], tem como objetivo definir uma plataforma Java padrão, mínima para pequenos dispositivos com as seguintes características:

- 160kB a 512kB de memória disponível para a plataforma Java
- Processador de 16 a 32 bits
- Baixo consumo de energia, freqüentemente usando energia de baterias
- Conectividade a algum tipo de rede, em geral sem-fio.

Aparelhos celulares, *pagers*, PDAs, aparelhos domésticos e terminais de vendas são alguns dos dispositivos que podem suportar essa especificação. A CLDC é uma tecnologia núcleo que será usada como base para um ou mais perfis. Um perfil J2ME define uma plataforma Java mais definida e focada para um mercado, categoria de dispositivo, ou indústria em particular. Na seção 2.3.2 iremos descrever o perfil MIDP, *Mobile Information Device Profile,* utilizado por nossa solução de monitoramento remoto.

Bibliotecas CLDC

O objetivo geral no desenvolvimento das bibliotecas CLDC é o de prover o conjunto mínimo necessário para um prático desenvolvimento de aplicações e definição

de perfis para uma variedade de pequenos dispositivos. Como tais dispositivos apresentam memória restrita e características variadas, é virtualmente impossível se ter um conjunto de bibliotecas que agradaria todos.

Na definição das bibliotecas, foi usada a especificação Java original como base, dando-se bastante ênfase na conectividade. A maioria das bibliotecas CLDC são um subconjunto das edições Java maiores, J2SE e J2EE, para garantir compatibilidade e portabilidade de aplicações. Embora compatibilidade seja um objetivo muito desejado, as bibliotecas J2SE e J2EE possuem fortes dependências internas que dificultam a formação de subconjuntos destas bibliotecas. Por esta razão, algumas bibliotecas foram modificadas, especialmente na área de rede e I/O. As bibliotecas CLDC podem ser divididas em duas categorias:

1) As que são subconjunto das bibliotecas J2SE padrões
2) As que são específicas do CLDC (mas que podem ser mapeadas para J2SE)

Classes pertencentes à primeira categoria estão localizadas nos pacotes java.lang.*, java.util.* e java.io.*. Estas classes são derivadas do J2SE. Classes pertencentes à segunda categoria estão localizadas no pacote java.microedition.*.

2.3.2 MIDP

O MIDP, *Mobile Information Device Profile*, cuja especificação está descrita em [9], é a definição de uma arquitetura e APIs associadas necessárias para prover um ambiente de desenvolvimento aberto para os MIDs, *Mobile Information Devices*. O

MIDP foi feito para rodar em cima do CLDC. Para tanto, um MID deve possuir as seguintes características mínimas de hardware, além daquelas que são requeridas pelo CLDC:

Display:

- Tamanho da tela: 96x54;
- Profundidade: 1 bit;
- Formato do *pixel* (proporção de aspecto): 1:1;

Input:

- "*One handed keyboard*" ou
- "*Two handed keyboard*" ou
- *Touch Screen*;

Memória:

- 128Kbytes para os componentes MIDP;
- 8Kbytes para dados das aplicações;
- 32Kbytes para o JAVA *runtime*;

Rede:

- Duplex, sem fio, possivelmente intermitente e com largura de banda limitada.

Como os MIDs possuem uma grande quantidade de potencialidades, o MIDPEG, grupo que elaborou o MIDP, limitou o conjunto de APIs necessárias para apenas àquelas necessárias para alcançar uma grande portabilidade. São as seguintes:

- Aplicação;

- Interface do Usuário;
- Armazenamento Persistente;
- Rede;
- Temporizadores (*timers*);

A figura 2-5, obtida da especificação [9], mostra como o MIDP se encaixa em um aparelho MID.

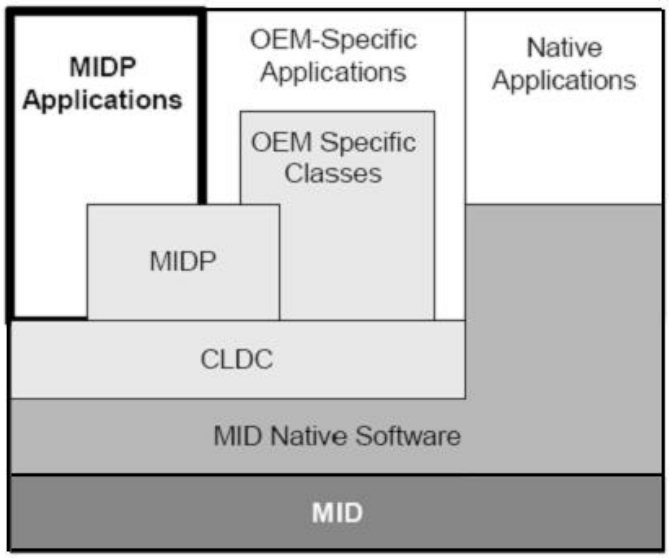


Figura 2-5: Arquitetura MIDP

No nível mais abaixo está o hardware, e logo acima dele o seu sistema. Após, vem o CLDC, que é onde está a Máquina Virtual K, KVM. A KVM irá permitir que APIs Java de alto nível sejam construídas.

As APIs MIDP rodam em cima do CLDC, como foi explicado anteriormente e

pode ser visto na figura. As classes OEM são classes não definidas pelo MIDP e podem ser utilizadas para funcionalidades específicas para determinado aparelho, o que significa que elas podem ou não ser portáveis para outros MIDs/aparelhos.

Quanto aos aplicativos, um MIDlet é aquele que usa APIs definidas pelo MIDP e CLDC, e são portáveis entre vários aparelhos. Uma aplicação OEM são aquelas que não fazem parte da especificação. Já as nativas são as que estão implementadas diretamente no sistema e não são escritas em Java.

A seguir detalharemos um pouco mais as APIs de interface do usuário, armazenamento persistente, rede e temporizadores. As aplicações MIDP, MIDlets, serão detalhadas em separado na seção 2.3.3

Interface do Usuário

A interface do usuário do MIDP, *Displayble*, é formada por uma API de alto-nível e outra de baixo-nível. A API de baixo-nível é baseada no uso da classe abstrata *Canvas.* Já a API de alto-nível é formada pelas classes *Alert, Form, List e TextBox,* que são extensões da classe abstrata *Screen.*

As classes da API de alto-nível fornecem abstrações e componentes que são altamente portáteis em diferentes MIDs, tais como: tipo de fonte, navegação, movimento de tela e alerta de erro. A implementação das classes de alto-nível em um certo dispositivo é “auto-ajustável” ao hardware e à interface de usuário nativa, não necessitando de configuração específica.

A classe *Canvas* da API de baixo-nível permite aplicações com um controle mais

direto da interface de usuário. Ela permite um maior controle do que é desenhado no *display* e recebido do teclado. Diferentemente da classe *Screen*, é de responsabilidade do programador da aplicação certificar-se da portabilidade destas funções através de MIDs de diferentes características, tais como: tamanho do *display*, se o *diplay* é colorido ou preto-e-branco e tipo de teclado. A figura 2-6 mostra a relação entre as classes de interface de usuário do MIDP.

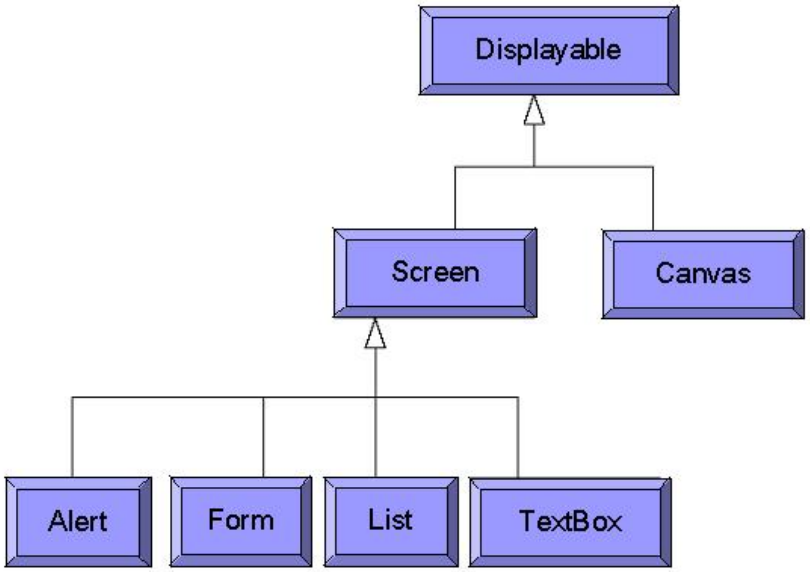


Figura 2-6: Classes de Interface do Usuário MIDP

Armazenamento Persistente

O MIDP disponibiliza um mecanismo para que os MIDlets possam guardar dados e lê-los mais adiante. É o chamado RMS, *Record Management System*. Conforme ilustrado em [23], o RMS é formado basicamente por duas entidades: *Record Store* e

Record.

- *Record Store*

É uma coleção de *records* que permanece mesmo durante múltiplas chamadas do MIDlet. A plataforma é responsável por manter a integridade desses *records*, mesmo após *reboots* ou trocas de baterias. O nome do *record store* não pode ter mais de 32 caracteres. Um record store é representado pela classe javax.microedition.rms.RecordStore

- *Records*

São vetores de *bytes*. Utilizados para armazenagem de diferentes tipos de dados. Eles são unicamente identificados pelo seu *recordId*, um valor inteiro. Vários *records* podem ser alocados num mesmo *record store*.

Rede

O MIDP herda a conectividade do CLDC e suporta um subconjunto do HTTP, que pode ser implementado com protocolos IP (TCP/IP) e não-IP (WAP e i-mode).

Como há uma grande variedade de redes sem-fio, fica por responsabilidade do aparelho e da própria rede sem-fio providenciar o serviço de aplicação. Pode ser necessário um *gateway* para ligar a rede sem-fio à internet. A aplicação cliente não precisa saber se há alguma rede não-IP sendo usada ou qualquer outra característica de outras redes. No entanto, essa possibilidade existe, o que faria o cliente e o servidor tirar vantagem deste conhecimento otimizando suas transmissões.

A interface HttpConnection possui funcionalidades que permitem a realização de funções específicas do HTTP. Qualquer aparelho que implemente MIDP deve suportar o

HTTP 1.1, requisições HEAD, GET, POST e *forms*.

Temporizadores

Aplicações que necessitem agendar tarefas para um tempo futuro pode utilizar as classes *Timer* e *TimerTask*, que incluem funções para uma execução ou várias execuções em determinados intervalos. Elas estão em java.util.Timer e java.util.TimerTask.

2.3.3 MIDlets

O MIDP define um modelo de aplicação que permite que os recursos limitados dos MIDs sejam compartilhados por várias aplicações, os MIDlets. Este compartilhamento é viável mesmo com os limitados recursos e *framework* de segurança do MID pois eles são obrigados a compartilhar classes e estão sujeitos a um conjunto de políticas e controles que permitem isso.

Os elementos de um MIDlet *suite*, cujo conjunto forma o software de gerenciamento da aplicação, e dos quais se espera que implementem as funções necessárias para instalar, selecionar, rodar e remover MIDlets, são:

- **Ambiente de Execução:** é compartilhado por todos os MIDlets que estão na mesma MIDlet suite, e qualquer MIDlet pode interagir com outro que esteja no mesmo pacote.
- **Empacotamento do MIDlet Suite**: um ou mais MIDlets podem ser empacotados

num único arquivo JAR, que contém as classes compartilhadas e os arquivos de recursos utilizados pelos MIDlets, além de um arquivo manifesto descrevendo seu conteúdo. Existem vários atributos pré-definidos que permitem identificação de um MIDlet, como nome, versão, tamanho de dados, descrição, etc.

- **Descritor de Aplicação**: é utilizado para gerenciar o MIDlet e é usado pela próprio MIDlet para atributos de configuração específica. O descritor permite que seja verificado que o MIDlet é adequada ao aparelho antes de se carregar todo o arquivo JAR do MIDlet suite. Ele também permite que parâmetros sejam passados para os MIDlets sem modificar os arquivos JAR.
- **Ciclo de Vida da Aplicação:** um MIDlet não deve possuir um método *public void static main* (). O software de gerenciamento de aplicação deve suprir a classe inicial necessária pelo CLDC para iniciar o MIDlet. Quando um MIDlet é instalado, ele é mantido no aparelho e fica pronto para uso. Quando é rodado, uma instância é criada através de seu construtor público sem argumentos, e seus métodos são chamados para passar pelos estados do MIDlet. Quando ele é destruído, os recursos utilizados por ela podem ser recuperados, incluindo os objetos criados e suas classes.

O software de gerenciamento de aplicação disponibilidade um ambiente no qual o MIDlet é instalado, iniciado, parado e desinstalado. Também é responsável por manusear os erros que podem ocorrer durante alguma destas etapas.

2.4 Symbian OS

Nesta seção iremos contextualizar o Symbian OS, uma vez que nosso projeto faz uso de um aparelho Nokia 6600, baseado neste sistema operacional. O emprego do Symbian OS possibilita o emprego de várias APIs específicas da programação MIDP. De acordo com [14] o aparelho Nokia 6600 faz uso do Symbian OS versão 7.0s. Por este motivo, consideraremos esta versão no decorrer de toda a seção 2.4.

Symbian OS é um sistema operacional aberto, adotado como padrão pelos grandes fabricantes de aparelhos móveis celulares. Ele foi projetado para atender aos requerimentos específicos dos telefones móveis de geração 2G, 2.5 G e 3G, especialmente no que tange aos requerimentos de comunicação de dados. O Symbian OS inclui um *kernel* multi-tarefas robusto, suporte integrado para telefonia, protocolos de comunicação, gerenciamento de dados, suporte avançado gráfico, interface de usuário de baixo-nível e uma variedade de ferramentas de aplicação [15]. O Symbian OS é caracterizado por:

- **Telefonia Móvel Integrada –** Symbian OS integra o poder da computação com a telefonia móvel, possibilitando serviços de dados ao mercado de massa.
- **Ambiente Aberto para Aplicações –** Symbian OS possibilita ao telefone móvel tornar-se uma plataforma para desenvolvimento de aplicações e serviços (programas e conteúdos), desenvolvidos em diferentes linguagens e formatos.
- **Padrões Abertos e Interoperabilidade –** Através de uma implementação flexível e modular, Symbian OS possibilita um conjunto de APIs e tecnologias

que são compartilhadas por todos os telefones móveis Symbian OS.

- **Multi-Tarefa –** Symbian OS é baseado em uma arquitetura micro *kernel* e implementa o processamento multi-tarefa e *threading*. Serviços do sistema, tais como telefonia, *middleware* de rede e ferramentas de aplicação, podem todos rodar nos seus próprios processos.
- **Inteiramente Orientado a Objeto e Componente –** O sistema operacional foi projetado desde o início levando-se em consideração seu direcionamento para os aparelhos móveis, usando técnicas avançadas de OO (programação orientada a objeto), proporcionando uma arquitetura baseada em componentes flexíveis.
- **Interface de Usuário Flexível –** Symbian OS possibilita o desenvolvimento de interfaces de usuário gráficas flexíveis, oferecendo inovação e novos serviços aos fabricantes, operadoras e usuários. O uso do mesmo sistema operacional "núcleo" em diferentes projetos também facilita a incorporação de aplicações desenvolvidas por terceiros.
- **Robustez –** Symbian OS guarda o acesso instantâneo ao dado do usuário. Isto assegura a integridade do dado, mesmo na presença de comunicação não confiável e perda de recursos como memórias e energia.

2.4.1 Arquitetura do Symbian OS

A figura 2-7, obtida de [15], mostra uma visão geral da arquitetura do Symbian OS versão 7.0s.

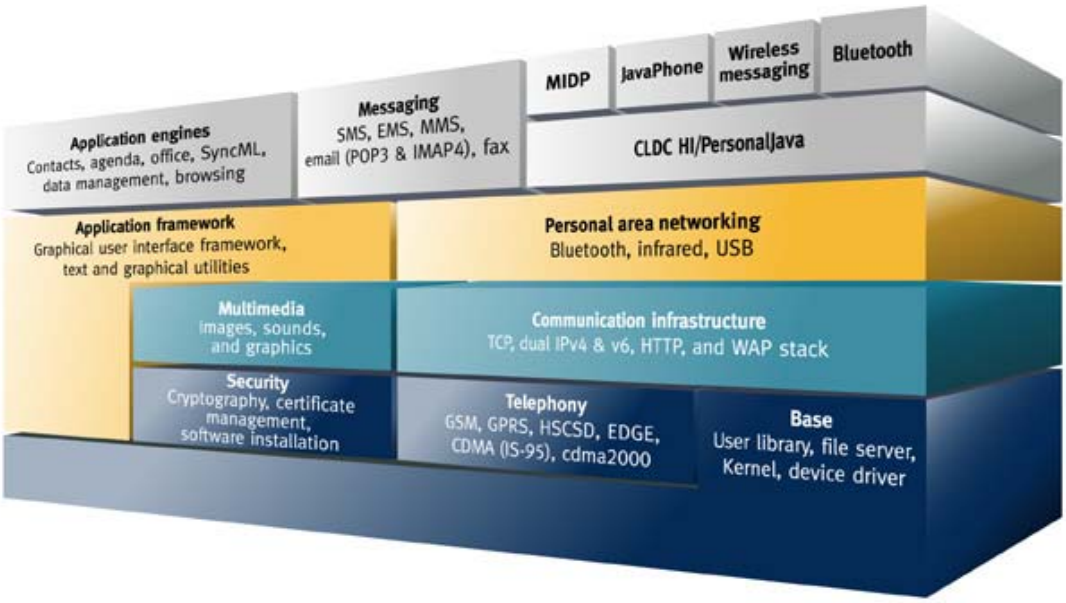


Figura 2-7: Arquitetura Symbian OS v7.0s

Capítulo 3 – Monitoramento Remoto de Bio-Sinais

3.1 Apresentação

A arquitetura que propomos é mostrada na figura 3-1. Ela é composta basicamente pelo módulo de coleta de bio-sinais, pelo aparelho celular compatível com MIDP e pelo servidor destino, além da estrutura de rede necessária para interligação. Detalharemos cada um destes macro-elementos nas seções 3.2 , 3.3 , 3.4 e 3.5.

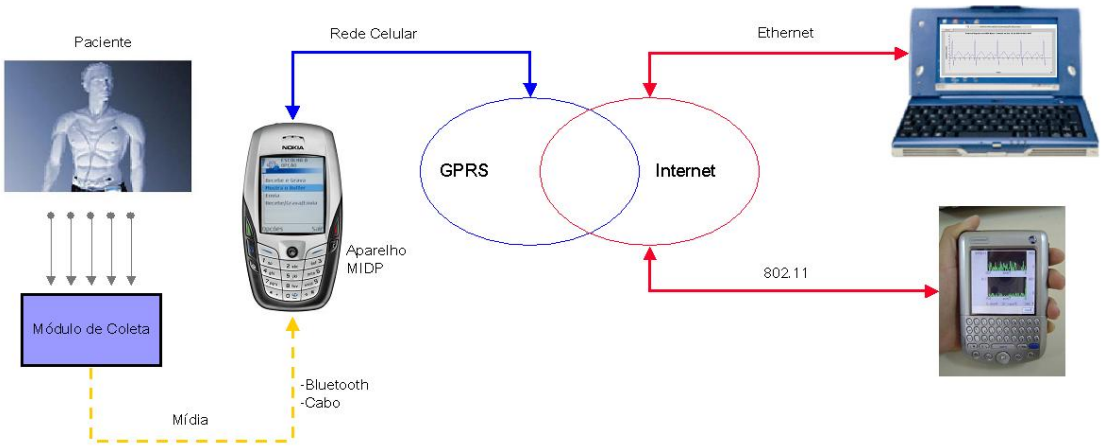


Figura 3-1: Monitoramento Remoto de Bio-Sinais

Esta arquitetura pode ser aplicada a diferentes tipos de bio-sinais, entre eles:

- Eletrocardiograma
- Temperatura
- Pressão sanguínea
- Saturação de oxigênio

- Taxa respiratória

Neste cenário, o fluxo do sinal coletado é o seguinte: a partir de eletrodos fixados no corpo do paciente, os sinais analógicos são enviados ao módulo de coleta onde sofrem amplificação, integração e digitalização. Os dados são então convertidos para o formato serial e enviados de forma contínua ao aparelho celular, para armazenamento e posterior retransmissão. A conexão entre o módulo de coleta e o aparelho celular MIDP pode ser feita através de um cabo serial ou através de uma conexão *bluetooth*. A partir daí, dependendo da lógica programada no aparelho, os dados são então enviados via uma conexão GPRS a um servidor remoto, instalado no hospital. Uma segunda opção é a disponibilização dos dados no PDA do médico, para pronta análise, através de uma rede local sem fio 802.11.

A vantagem do emprego de uma conexão *bluetooth* entre o módulo de coleta e o aparelho MIDP está na mobilidade proporcionada pela mesma. Segundo [22] o alcance proporcionado por uma conexão *bluetooth* é de 10m para rádios de 1mW e até 100m para rádios de 100mW. A mobilidade entre o módulo de coleta e o aparelho MIDP pode ser importante para casos de pronto atendimento médico. Um exemplo seria o emprego numa ambulância de pronto socorro, onde o paciente poderia ser monitorado na maca, externa à ambulância, somente pelo módulo de coleta. Por sua vez, o módulo de coleta transmitiria os dados medidos via *Bluetooth* para o celular MIDP alojado na ambulância. Os dados então seriam retransmitidas para o hospital, via conexões HTTP POST.

3.2 Módulo de Coleta

O módulo de coleta é responsável pela aquisição do sinal medido, sua manipulação e posterior exteriorização para o celular MIDP. Conforme mencionado na seção 3.1, estamos propondo uma solução independente do tipo de bio-sinal coletado. Isto é, definindo-se os parâmetros de conversação entre o módulo de coleta e o aparelho celular MIDP, poder-se-á ter diferentes tipos medições, sejam elas de eletrocardiograma, de temperatura, de pressão sanguínea, de saturação de oxigênio ou de taxa respiratória. Fundamentalmente, como parâmetro de conversação básico, teríamos que os dados enviados pelo módulo de coleta estejam no formato serial, seja a mídia um cabo RS-232 ou uma conexão sem-fio *bluetooth*. Isto possibilita ao aparelho celular MIDP abrir uma conexão para a porta RS-232 ou *bluetooth* e receber de forma serial os bytes relativos a cada amostra da medição, armazenado-os no *buffer.* Em seguida, também de forma serial *byte* a *byte*, os dados do *buffer* são retransmitidos via uma conexão HTTP POST para o servidor destino, para posterior análise. Vale lembrar que a conexão HTTP POST é estabelecida tendo como base a conexão GPRS estabelecida entre o aparelho celular MIDP e a Rede Celular GSM/GPRS.

Com o intuito de demonstrar as características funcionais e práticas de um módulo de coleta real, apresentaremos nas seções 3.2.1 , 3.2.2 , 3.2.3 , 3.2.4 e 3.2.5 o Módulo ECG. O Módulo ECG, descrito em [28], é um módulo de coleta de eletrocardiograma humano desenvolvido pelo aluno de mestrado Fábio Lúcio C. Jr, do Departamento de Ciência da Computação da UFMG. Ele atende as características para interface com o aparelho celular MIDP descritas no parágrafo anterior, enviando de forma serial as

amostras de eletrocardiograma. Como mídia, para exteriorização dos dados, o Módulo ECG faz uso de uma porta RS-232.

3.2.1 Módulo ECG

O Módulo ECG é constituído por elementos que permitem a coleta de sinais de eletrocardiograma, sua manipulação e processamento, sua armazenagem e, finalmente, seu envio para uma rede ou unidade exterior. A figura 3-2 mostra o hardware do módulo ECG, bem como sua conexão com os eletrodos.

Figura 3-2: Eletrodos e o Módulo de Eletrocardiograma

Uma característica importante do Módulo ECG é que ele é portátil, possibilitando aplicações de monitoramento sem-fio. Na figura 3-3 podemos ver os detalhes da placa do Módulo ECG: seus componentes eletrônicos, circuitos integrados, conectores, porta-pilhas e memória *Compact Flash*.

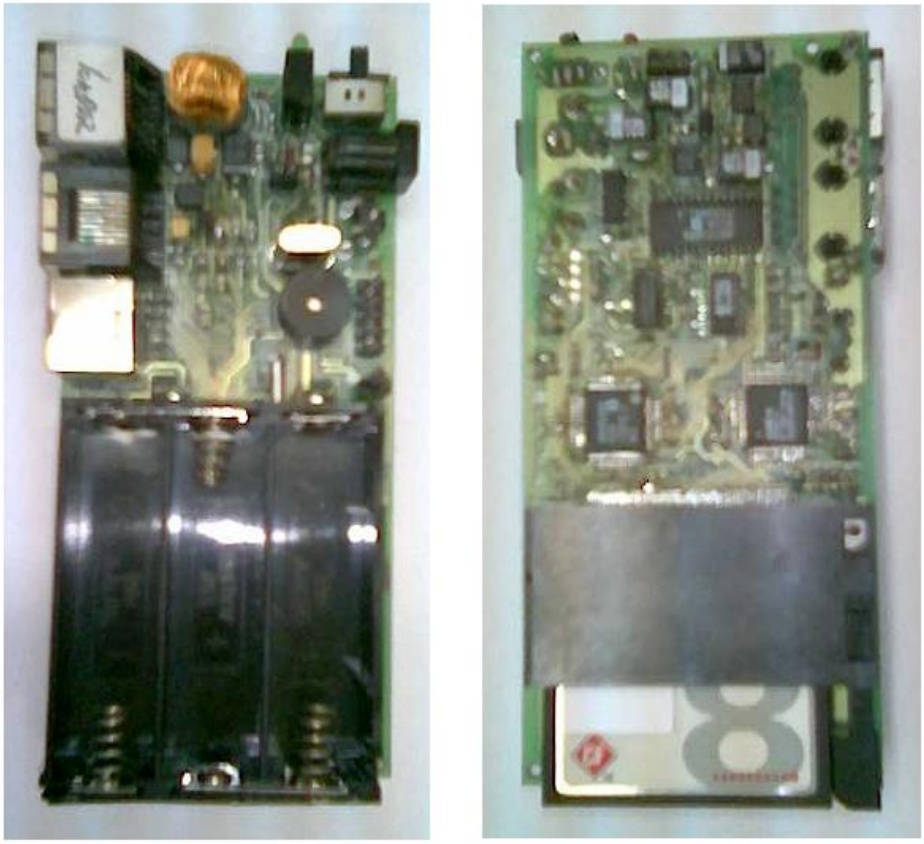


Figura 3-3: Vistas Frontal e Traseira do Módulo ECG

A figura 3-4 mostra de maneira simplificada a arquitetura do Módulo ECG. Nela podemos perceber o emprego de três blocos principais que, fisicamente, correspondem a três *chips*: o ECG ASIC, o ADC e o micro-controlador uC 8051. Nas seções 3.2.3 , 3.2.4 e 3.2.5 faremos uma breve apresentação destes blocos.

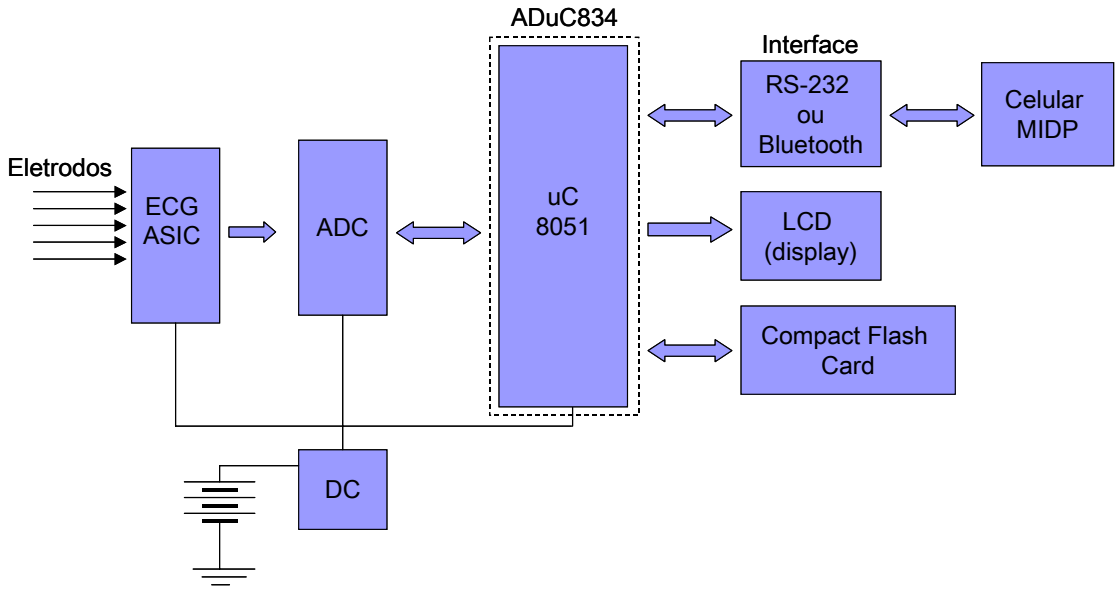


Figura 3-4: Arquitetura do Módulo ECG

3.2.2 Eletrodos

Sinais de eletrocardiograma podem ser medidos de diferentes modos, variando-se a quantidade de eletrodos. O Módulo ECG está configurado para receber sinais de 5 eletrodos, espalhados em diferentes pontos do tronco humano, conforme se segue:

- Eletrodos RA, vermelho, fixado próximo ao braço direito
- Eletrodo LA, amarelo, fixado próximo ao braço esquerdo
- Eletrodo RL, preto, fixado próximo à perna direita
- Eletrodo LL, verde, fixado próximo à perna esquerda
- Eletrodo C, azul, fixado na região central do peito

3.2.3 ECG ASIC

O bloco ECG ASIC é responsável pela amplificação e integração dos sinais dos eletrodos. Na placa do Módulo ECG, o bloco ECG ASIC é constituído por um simples *chip* comercial da Welch Allyn® , que integra diversos circuitos eletrônicos. O diagrama do bloco ECG ASIC pode ser visto no apêndice A.

3.2.4 ADC

O bloco ADC tem como função transformar o sinal analógico proveniente do ASIC ECG em amostras digitais. A precisão de cada amostra, isto é, a quantidade de bits necessários para representar os níveis de tensão do sinal de eletrocardiograma, pode ser configurada de acordo com as necessidades da aplicação. No módulo ECG a precisão utilizada tem sido de 24 bits para a amostra, disponibilizados em 3 interações de 8 bits. Na placa do módulo ECG, o bloco ADC é composto pelo chip comercial AD7718 da Analog Devices ®. O diagrama do bloco ADC pode ser visto no apêndice E.

3.2.5 uC 8051

O bloco uC 8051, assim como outras funcionalidades adicionais, está contido no chip comercial ADuC834 da Analog Devices ®. O uC 8051 tem papel central no módulo ECG e é responsável pelo controle dos demais dispositivos. Em sua memória de código

ficam armazenadas todas as funções responsáveis pela interação com os demais blocos. Para a programação do 8051 é utilizada a ferramenta Keil da Keil SoftwareTM. As funções são programadas em linguagem C e o código é então convertido em HEXADECIMAL para ser carregado no 8051. O diagrama do bloco uC 8051 e as funções desenvolvidas para sua programação podem ser vistos no apêndice F.

3.3 Aparelho Celular MIDP

Nas seções 3.3.1 e 3.3.2 discorreremos respectivamente sobre o modelo de aparelho celular utilizado como plataforma de desenvolvimento e também sobre a aplicação MIDP desenvolvida, o MIDlet ECG.

3.3.1 Aparelho Nokia 6600

O modelo de aparelho celular que escolhemos como plataforma para nossa solução foi o Nokia 6600. O mesmo apresenta alguns requisitos essenciais que nossa solução de monitoramento remoto demandava, entre eles podemos citar:

- Sistema Operacional Symbian (Plataforma Série 60)
- Suporte para aplicativos Java (J2ME, MIDP 2.0)
- Grande memória (interna de 7MB + cartão de 32MB)
- Interfaces Infravermelho e *Bluetooth* (para *loading* do aplicativo)
- Suporte a conexões GPRS

O conjunto completo das características e especificações técnicas do aparelho Nokia 6600, obtido em [16], está listado no apêndice B.

A figura 3-5 mostra o aparelho Nokia 6600 que utilizamos como plataforma para nossa aplicação MIDP, o MIDlet ECG. Nela são mostradas duas telas reais do MIDlet ECG, quando em funcionamento. A primeira tela mostra o menu inicial de opções: “Recebe e Grava”, “Mostra o Buffer”, “Envia” e “Recebe/Grava/Envia”. Já na segunda

tela é apresentado o resultado da opção “Mostra o Buffer”. Podemos notar que a tela nos mostra o conteúdo dos três *records* que sempre acompanham cada medição:

1) TAMANHO DA MEDIÇÃO, em bytes
2) TIMESTAMP, data e horário da medição
3) DADOS, conteúdo das amostras

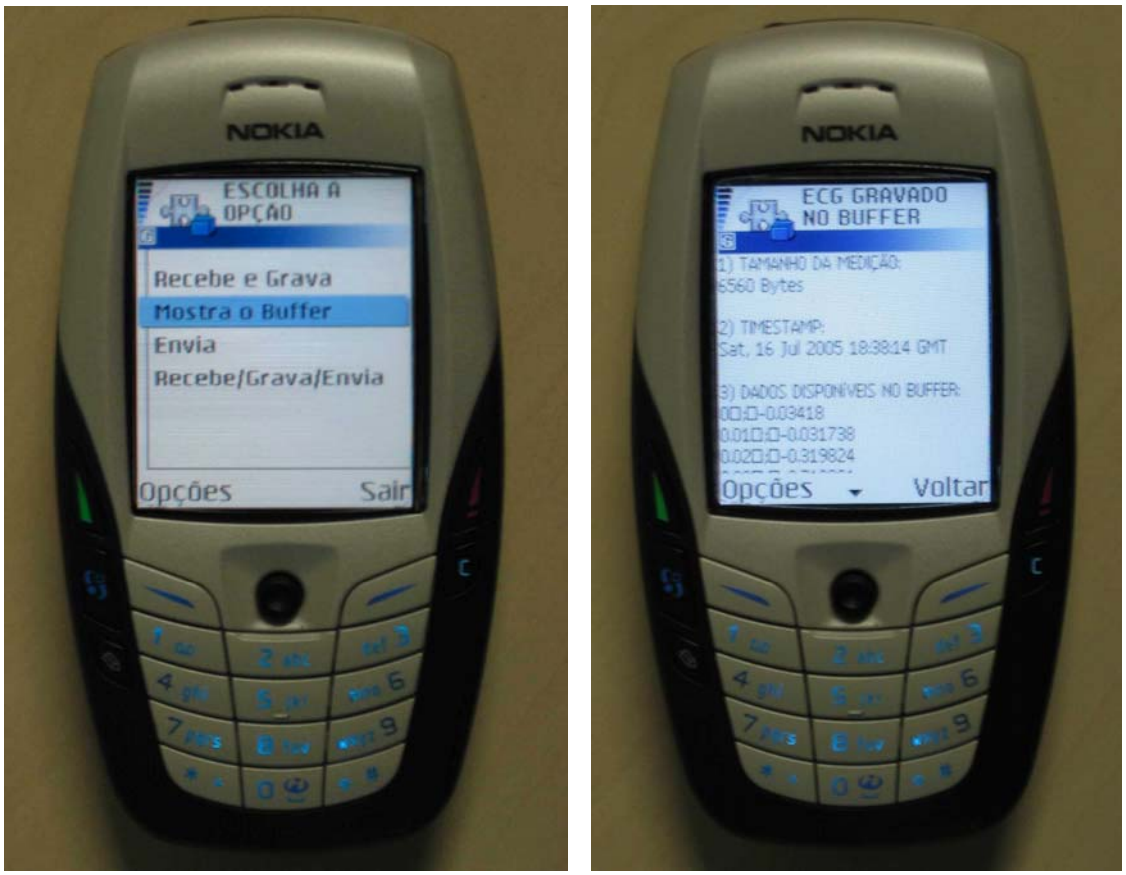


Figura 3-5: Execução do MIDlet ECG no Aparelho Nokia 6600

3.3.2 MIDlet ECG

A aplicação que desenvolvemos está totalmente confinada em um único MIDlet,

chamado ECG. O MIDlet ECG, quando solicitado, pode realizar as seguintes ações principais, disponíveis no menu inicial:

- **Recebe e Grava** – recebe as amostras da medição e grava o conteúdo na memória persistente do MIDlet.. Além disso, grava o *timestamp,* e também o tamanho da medição em *bytes.*
- **Mostra o *Buffer*** – acessa a memória persistente do MIDlet e mostra na tela do aparelho celular o conteúdo da última medição gravada, *timestamp,* e também o tamanho da medição em *bytes.*
- **Envia** – acessa a memória persistente do MIDlet e envia para o servidor destino o conteúdo da última medição gravada, *timestamp,* e também o tamanho da medição em *bytes.*
- **Recebe/Grava/Envia** – recebe as amostras da medição e grava o conteúdo, o *timestamp,* e também o tamanho da medição em *bytes* na memória persistente do MIDlet. Em seguida, ele acessa a memória persistente do MIDlet e envia estas três informações (conteúdo dos 3 *records*) para o servidor destino.

Conforme mostrado na figura 3-6, cada ação do menu inicial aciona um dos quatro métodos principais seguintes:

- recebe()
- mostra_dados()
- envia()
- recebe_envia()

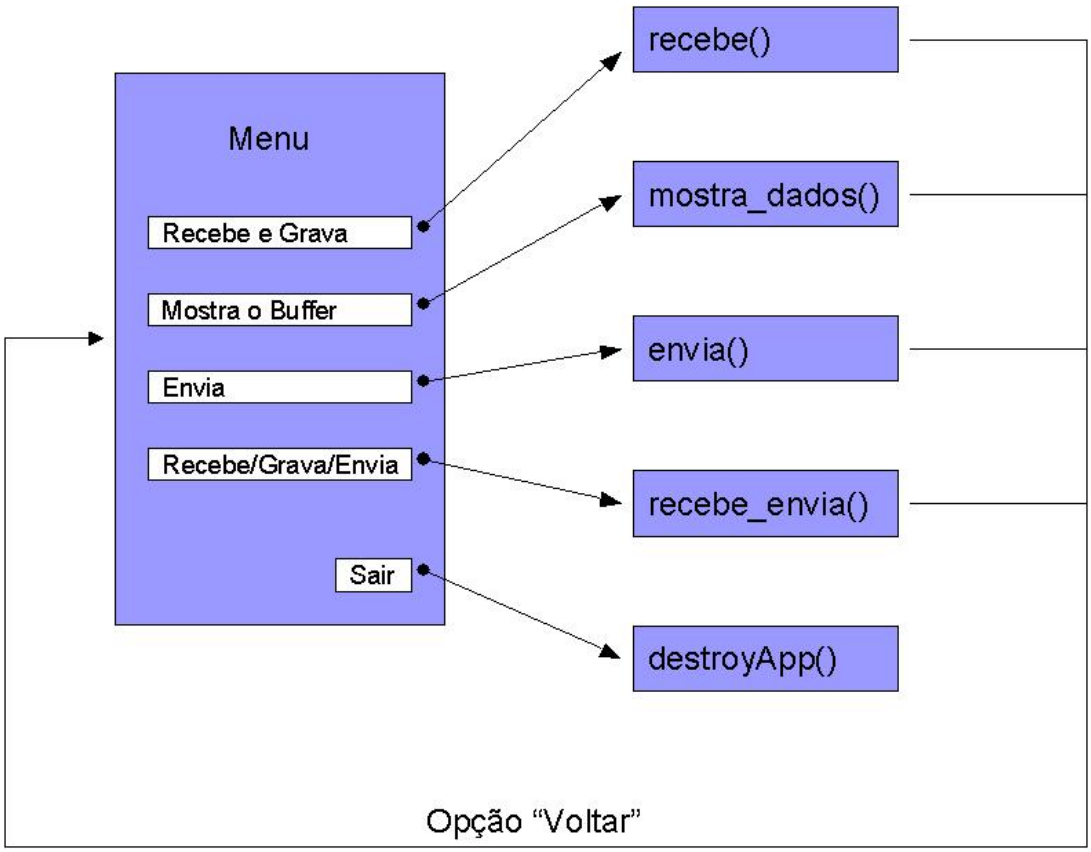


Figura 3-6: Estrutura do MIDlet ECG

Após a execução de uma das quatro ações do menu inicial, e conseqüente execução de um dos quatro métodos principais, o usuário pode voltar ao menu inicial através da opção "Voltar", presente na tela de resultado da execução. De volta ao menu inicial, o usuário pode novamente escolher uma das quatro ações e reinicializar a execução.

Para finalizar a completa execução do MIDlet ECG, o usuário deve escolher a opção "Sair", presente no canto inferior direito do menu inicial. Tal opção dispara o método destroyApp(), fechando o MIDlet ECG. A seguir, nas seções 3.3.3, 3.3.4, 3.3.5 e 3.3.6, detalharemos os quatro métodos principais do MIDlet ECG.

3.3.3 Método *recebe*

O método principal *recebe* implementa a opção “Recebe e Grava” do menu inicial. Ele recebe as amostras da medição e grava o conteúdo na memória persistente do MIDlet. Além disso, grava o *timestamp*, e também o tamanho da medição em bytes. A figura 3-7 ilustra o diagrama de execução do método principal *recebe*, com seus quatro métodos componentes.

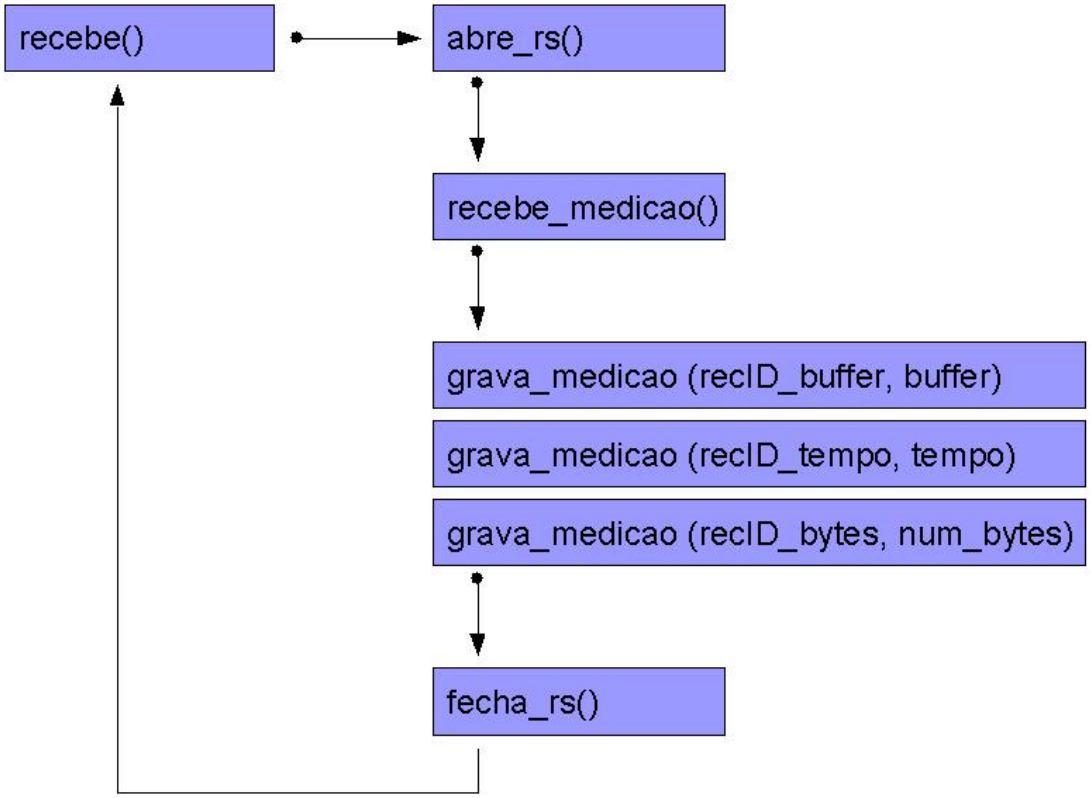


Figura 3-7: Método recebe

A seguir apresentaremos uma breve explicação destes quatro componentes.

1) **abre_rs()** : realiza a abertura do *record store* rs, unidade de armazenamento persistente. O *record store* rs contém os três *records* usados por nossa aplicação. Em poucas palavras, armazenamento persistente é aquele que se conserva após o fechamento do MIDlet ou mesmo após o desligamento do aparelho celular. O código fonte do método abre_rs() está descrito no apêndice A.

2) **recebe_medição()** : abre uma conexão para recebimento do conteúdo da medição. Na atual versão do MIDlet ECG a conexão implementada é uma conexão HTTP com um servidor *web*, onde está armazenado um arquivo contendo os dados de uma medição real de eletrocardiograma. A URL onde está localizado o arquivo com os dados de uma medição real é o seguinte: http://www2.dcc.ufmg.br/~geraldof/medicao.txt. Como trabalho futuro, o método recebe_medição() pode ser modificado afim de receber os dados provenientes de um módulo de coleta. Neste caso, poder-se-á utilizar um aparelho celular que suporte cabo serial RS-232 ou que suporte uma conexão sem-fio *Bluetooth*, desde que o módulo de coleta suporte uma destas duas interfaces. O código fonte do método recebe_medicao() está descrito no apêndice A.

3) **grava_medicao()** : realiza a gravação persistente dos três *records* alocados no *record store* rs. São eles:

    a. *record* de ID=1, para armazenamento do conteúdo da medição.

    b. *record* de ID=2, para armazenamento do *timestamp* da coleta.

    c. *record* de ID=3, para armazenamento do número de *bytes* da medição.

No método principal *recebe* o método grava_medição() é chamado seqüencialmente 3 vezes, gravando os três *records* acima. O código fonte do método grava_medicao() está descrito no apêndice A.

4) **fecha_rs()** : realiza o fechamento do *record store* rs. O *record store* rs armazena de maneira persistente os *records* utilizados no MIDlet ECG. O fechamento de um *record store* sempre acompanha a abertura do mesmo, ocorrendo entre eles as operações de leitura, escrita e demais. O código fonte do método fecha_rs() está descrito no apêndice A.

3.3.4 Método *mostra_dados*

O método principal *mostra_dados* acessa a memória persistente do MIDlet e lê a última medição gravada, a data e horário, *timestamp*, e também o tamanho da medição em *bytes*. Estes dados devem ter sido gravados previamente nos três *records* descritos na seção 3.3.3. Em seguida, ele mostra na tela do aparelho celular o resultado destas três leituras. A figura 3-8 ilustra o diagrama de execução do método principal *mostra_dados*, com seus quatro métodos componentes.

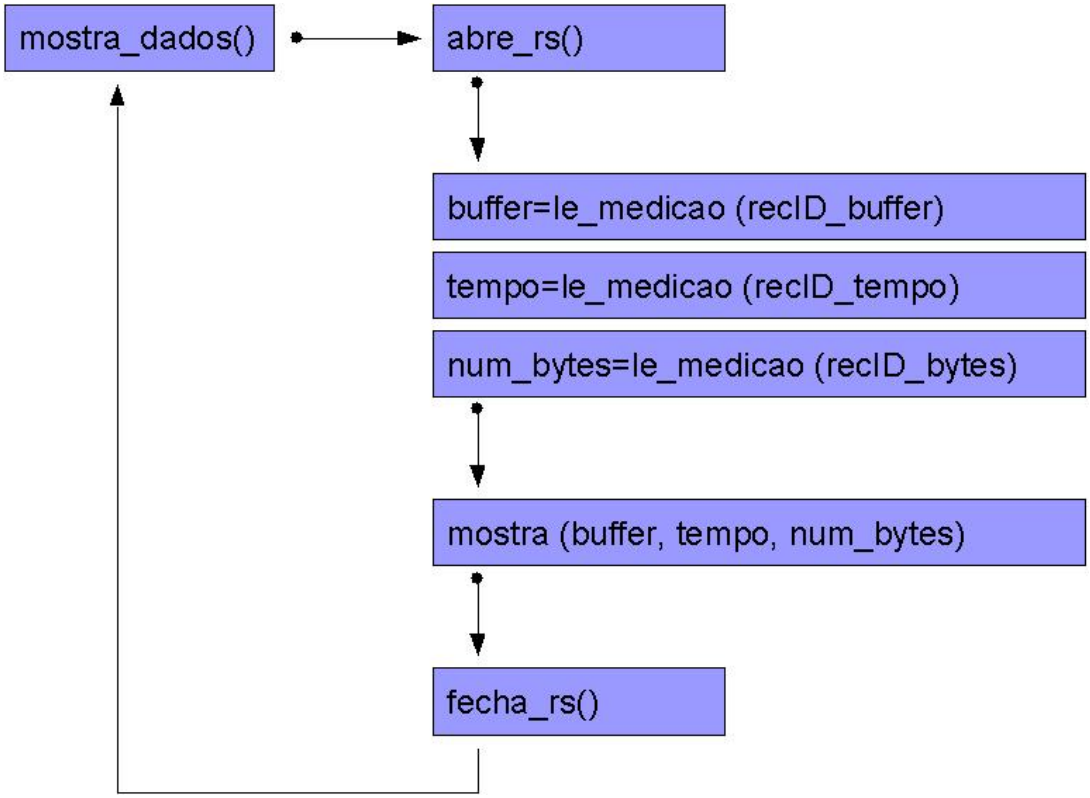


Figura 3-8: Método mostra_dados

A seguir apresentaremos uma breve explicação destes quatro métodos componentes.

1) **abre_rs()** : este método, já utilizado pelo método principal *recebe* e descrito na seção 3.3.3, realiza a abertura do *record store* rs.

2) **le_medicao()** : o método le_medicao() realiza a leitura dos três *records* alocados no *record store* rs, retornando uma variável *String* para cada. Afim de não retornar um valor nulo, *null,* os *records* precisam ter sido gravados previamente pelo menos uma vez. Os três *records* onde é aplicada a leitura são os seguintes:

    a. *record* de ID=1, onde está armazenado o conteúdo da medição.

    b. *record* de ID=2, onde está armazenado o *timestamp* da coleta.

    c. *record* de ID=3, onde está armazenado o número de *bytes* da medição.

No método principal *mostra_dados* o método le_medição() é chamado seqüencialmente 3 vezes, lendo os três *records* acima. O código fonte do método le_medicao() está descrito no apêndice A.

3) **mostra()** : o método mostra() apresenta na tela do aparelho celular o resultado da leitura dos três *records* utilizados no MIDlet ECG. São eles:

- TAMANHO DA MEDIÇÃO, em bytes
- TIMESTAMP, data e horário da medição
- DADOS, conteúdo das amostras

A figura 3-5 da seção 3.3.1 ilustra o resultado real da execução do método mostra(), quando em funcionamento no aparelho Nokia 6600. O código fonte do método mostra() está descrito no apêndice A.

4) **fecha_rs()** : este método, já utilizado pelo método principal *recebe* e descrito na seção 3.3.3, realiza o fechamento do *record store* rs.

3.3.5 Método *envia*

Este método acessa a memória persistente do MIDlet, lê o conteúdo da última medição gravada, a data e horário, *timestamp*, e também o tamanho da medição em *bytes*, enviando-os para o servidor destino via uma conexão HTTP. A figura 3-9 ilustra o diagrama de execução do método principal *envia*, com seus quatro métodos componentes.

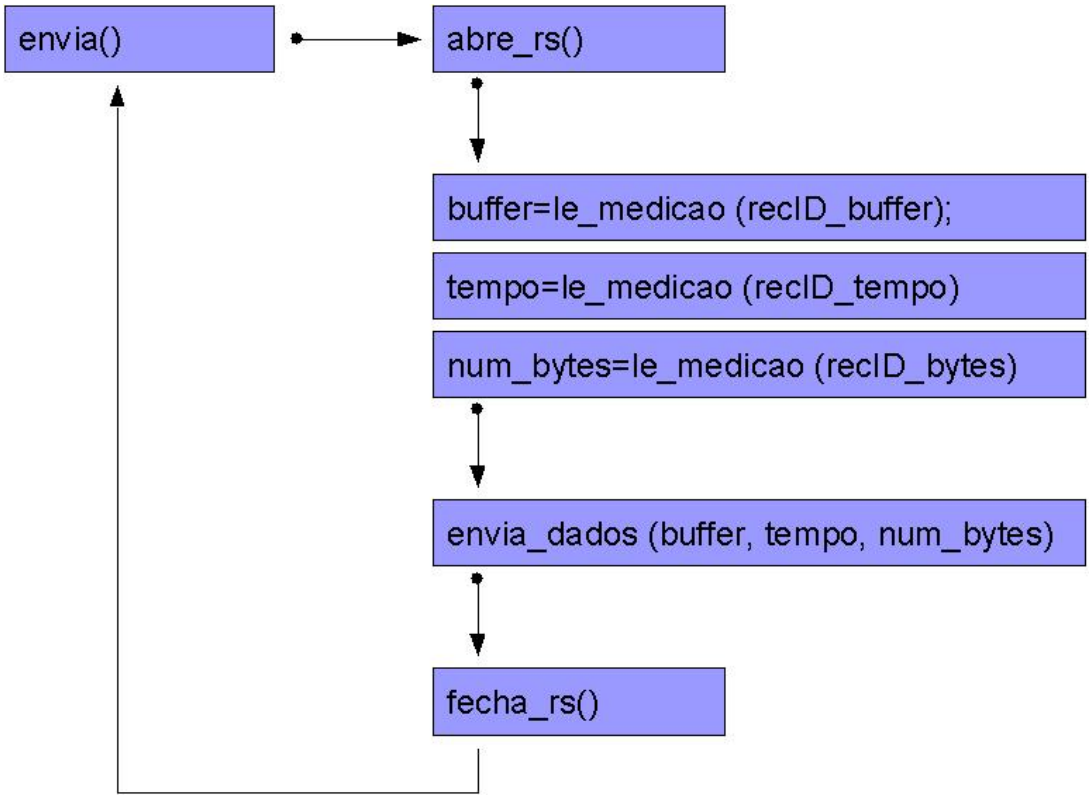


Figura 3-9: Método envia

A seguir apresentaremos uma breve explicação destes quatro métodos componentes.

1) **abre_rs()** : este método, já utilizado pelo método principal *recebe* e descrito na seção 3.3.3, realiza a abertura do *record store* rs.

2) **le_medicao()** : este método, já utilizado pelo método principal *mostra_dados* e descrito na seção 3.3.4, realiza a leitura de um *record*, retornando uma *String*.

3) **envia_dados()** : o método envia_dados() é responsável por abrir uma conexão HTTP POST com o servidor destino, enviando para o mesmo as três *Strings* obtidas pelo método le_medição(). Estas três *Strings* são as seguintes: *buffer*, tempo e num_bytes. No servidor destino, como será detalhado na seção 3.4, foi desenvolvido um aplicativo PHP que recebe esta conexão HTTP POST e grava as três *Strings* em um único arquivo texto, de forma concatenada. Este arquivo texto fica então armazenado no servidor até que uma nova medição seja enviada, sobrepondo-o. O código fonte do método envia_dados() está descrito no apêndice A.

4) **fecha_rs()** : este método, já utilizado pelo método principal *recebe* e descrito na seção 3.3.3, realiza o fechamento do *record store* rs.

3.3.6 Método *recebe_envia*

Basicamente, este método é uma composição dos métodos já apresentados nas seções 3.3.3, 3.3.4 e 3.3.5. Ele recebe as amostras da medição através do método recebe_medicao() e grava o conteúdo, o *timestamp*, e também o tamanho da medição em *bytes* na memória persistente do MIDlet, utilizando-se do método grava_medicao(). Em seguida, após um atraso de tempo – incluído propositalmente – de alguns segundos, ele acessa a memória persistente do MIDlet e envia o conteúdo dos três *records* para o servidor destino. Os métodos componentes do método principal *recebe_envia* são os seguintes:

1) **abre_rs()** : este método, já utilizado pelo método principal *recebe* e descrito na seção 3.3.3, realiza a abertura do *record store* rs.

2) **recebe_medicao()** : este método, já utilizado pelo método principal *recebe* e descrito na seção 3.3.3, abre uma conexão para recebimento do conteúdo da medição.

3) **grava_medicao()** : este método, já utilizado pelo método principal *recebe* e descrito na seção 3.3.3, realiza a gravação persistente dos três *records* alocados no *record store* rs.

4) **wait()** : método responsável pela adição de um atraso de tempo entre o recebimento e o envio dos dados. Este atraso tem o intuito de mostrar ao usuário a tela de resultado do

recebimento da medição. Para que, logo após alguns segundos, possa ser mostrada a tela de resultado do envio dos dados. Sem este atraso, devido à velocidade de execução do MIDlet, somente a tela de resultado do envio seria percebida pelo usuário.

5) **le_medicao()** : este método, já utilizado pelo método principal *mostra_dados* e descrito na seção 3.3.4, realiza a leitura de um *record*, retornando uma *String*.

6) **envia_dados()** : este método, já utilizado pelo método principal *envia* e descrito na seção 3.3.5, é responsável por abrir uma conexão HTTP POST com o servidor destino, enviando para o mesmo as três *Strings* obtidas pelo método le_medição().

7) **fecha_rs()** : este método, já utilizado pelo método principal *recebe* e descrito na seção 3.3.3, realiza o fechamento do *record store* rs

A figura 3-10 ilustra o diagrama de execução do método principal *recebe_envia*, com seus sete métodos componentes.

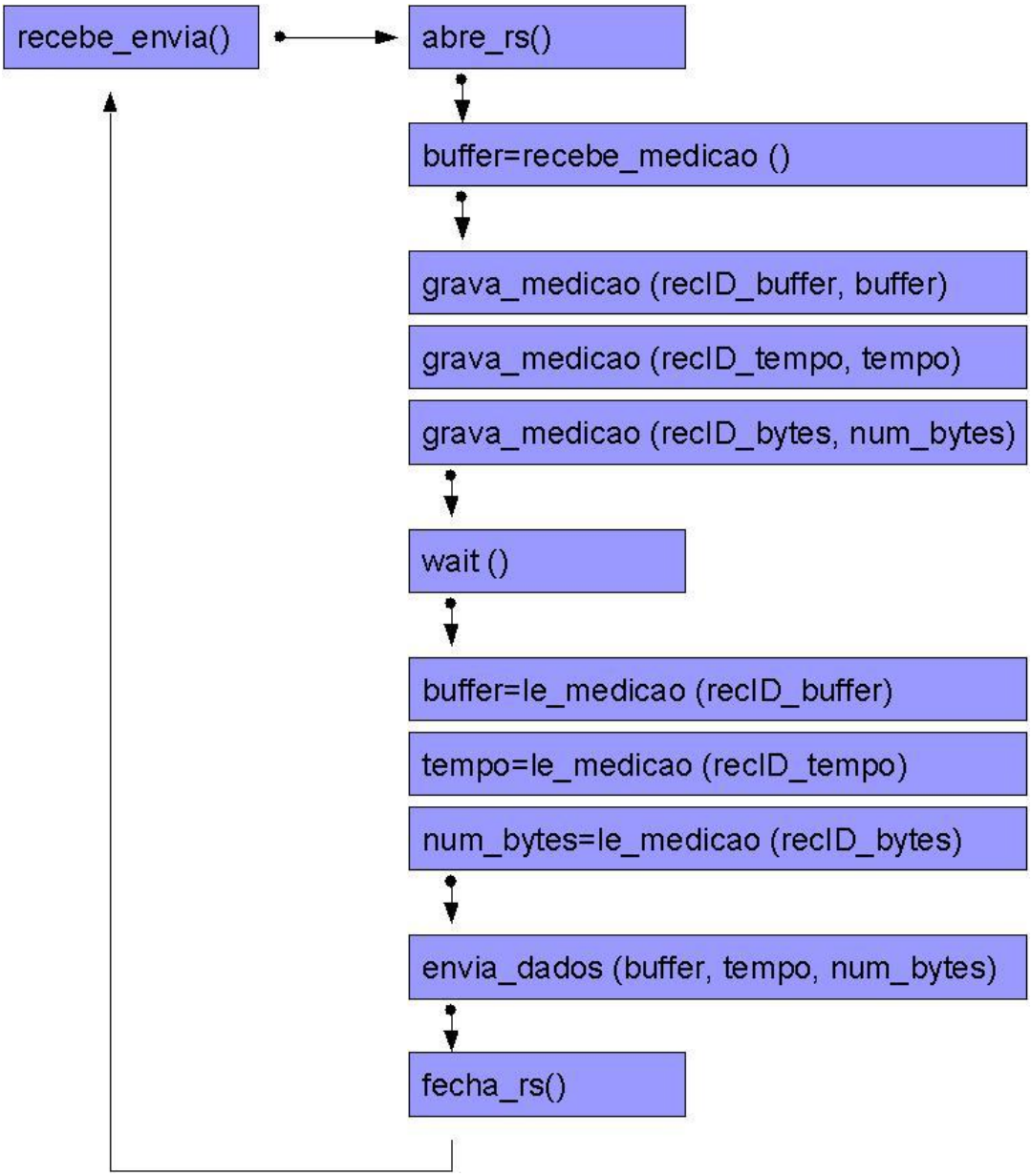


Figura 3-10: Método recebe_envia

3.4 Servidor Destino

3.4.1 Servidor Destino – Aplicativo PHP

Como servidor destino de nossas conexões HTTP, utilizamos o próprio servidor web do Departamento de Ciência da Computação da UFMG. Nele desenvolvemos um simples aplicativo PHP (PHP versão 4.3.3) que recebe a conexão HTTP e salva seu conteúdo (dados de eletrocardiograma) em um arquivo texto, ficando o mesmo disponível para posterior utilização. O código .php utilizado no servidor destino está listado no apêndice C. A figura 3-11 mostra a arquitetura da solução utilizada no servidor destino.

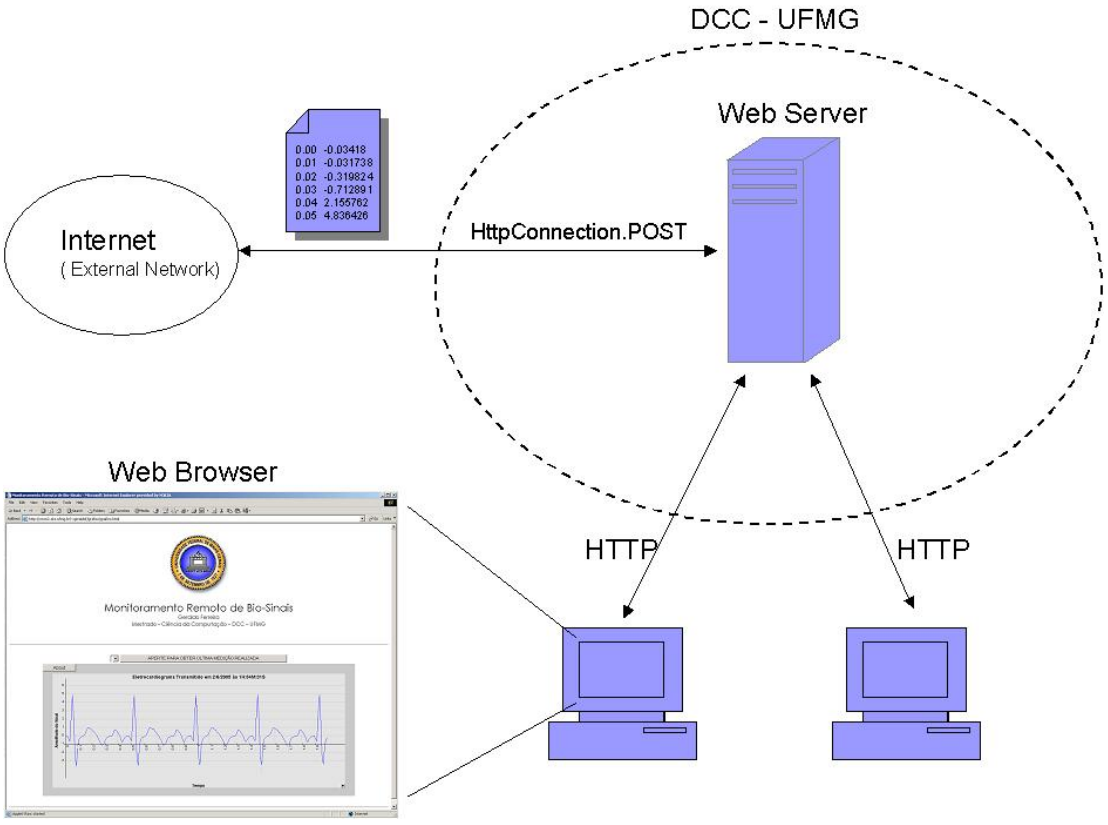


Figura 3-11: Servidor Destino das Medições de Eletrocardiograma

3.4.2 Servidor Destino – Aplicativo Java

Finalmente, com o intuito de disponibilizar as medições de eletrocardiograma para qualquer usuário conectado à internet, desenvolvemos uma página de web contendo um aplicativo Java Applet. Quando acessada, a página http://www2.dcc.ufmg.br/~geraldof/grafico/grafico.html mostra o gráfico relativo a cada medição realizada (arquivo texto gravado pelo aplicativo PHP), bem como o *timestamp* e o tamanho da medição em *bytes*. O *timestamp* torna possível a identificação exata da medição pela comparação com o *timestamp* gerado na aplicação MIDP do aparelho celular cliente. A figura 3-12 mostra um acesso à página do servidor destino.

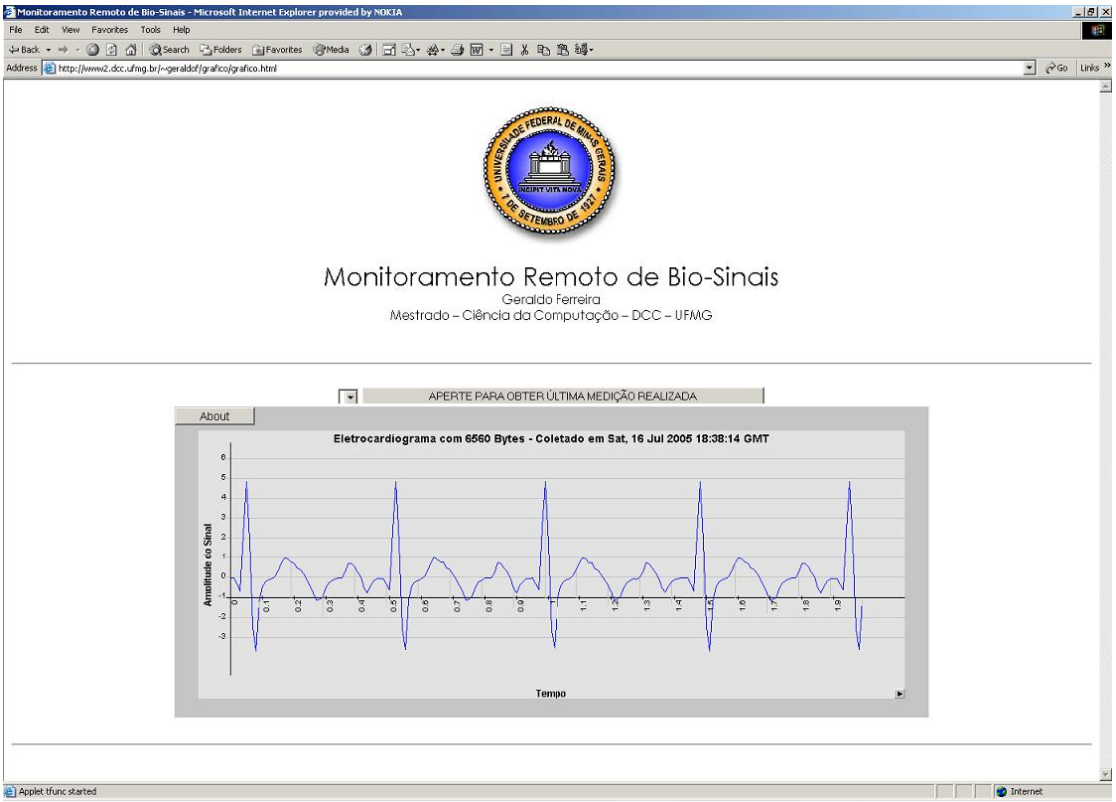


Figura 3-12: Gráfico do Eletrocardiograma Recebido no Servidor Destino

3.5 Infra-Estrutura de Rede

Conforme apresentado no início do capítulo 3, figura 3-1, nossa proposta de monitoramento remoto utiliza-se da rede celular GSM/GPRS. Para acessar os recursos desta rede é necessário que o aparelho celular com o MIDlet ECG esteja corretamente configurado. Além disso, o aparelho deve estar registrado em alguma operadora que ofereça o serviço GPRS.

Visando realizar os testes reais com a rede GPRS, adquirimos as seguintes assinaturas de telefonia móvel celular:

- Plano GSM/GPRS Pré-Pago TIM – São Paulo (11-8434-0972)
- Plano GSM/GPRS Pré-Pago TIM – Belo Horizonte (31-9254-4952)
- Plano GSM/GPRS Pré-Pago Claro – São Paulo (11-9495-4373)

Os testes foram bastante satisfatórios em ambas as cidades. Detalharemos estes resultados no capítulo 4. Abaixo as configurações de conexão de rede que utilizamos, elas se referem ao aparelho Nokia 6600. Se, porventura, o MIDlet ECG for instalado em outro modelo/fabricante, a operadora deve ser consultada, para o fornecimento da configuração específica.

Configuração da Conexão GPRS para a Operadora TIM

1) Menu > Ferramentas > Configurações > Conexão > Pontos de acesso
2) Criar uma conexão com os seguintes valores:
    a) Nome da conexão : **TIM WAP FAST**
    b) Portadora de dados: **GPRS**
    c) Nome do pt. de acesso: **wap.tim.br**
    d) Nome do usuário: **wap@tim**
    e) Solicitar senha: **Não**
    f) Senha: ********
    g) Autenticação: **Normal**
    h) Homepage: **http://wap.tim.com.br**
3) Opções > Configs. Avançadas
    a) End. IP do telefone: **Automático**
    b) Serv. nomes princip. : **0.0.0.0**
    c) Serv. nomes secund. : **0.0.0.0**
    d) Endereço do proxy: **200.244.116.065**
    e) Número porta proxy: **8080**

Configuração da Conexão GPRS para a Operadora Claro

1) Menu > Ferramentas > Configurações > Conexão > Pontos de acesso
2) Criar uma conexão com os seguintes valores:
    a) Nome da conexão : **Claro Dados GPRS**
    b) Portadora de dados: **GPRS**
    c) Nome do pt. de acesso: **claro.com.br**
    d) Nome do usuário: **claro**
    e) Solicitar senha: **Sim**
    f) Senha: **claro**
    g) Autenticação: **Normal**
    h) Homepage: **http://wap.claro.br**
3) Opções > Configs. Avançadas
    a) End. IP do telefone: **Automático**
    b) Serv. nomes princip. : **0.0.0.0**
    c) Serv. nomes secund. : **0.0.0.0**
    d) Endereço do proxy: **200.169.126.011**
    e) Número porta proxy: **8799**

Capítulo 4 – Resultados

Nas seções 4.1 e 4.2 mostraremos respectivamente os resultados encontrados com as ferramentas *Sun J2ME Wireless Toolkit* e *Nokia Developer's Suíte.* Já na seção 4.3 mostraremos o resultado da execução real no aparelho Nokia 6600.

4.1 Simulação no Sun J2ME Wireless Toolkit

A ferramenta *Sun J2ME Wireless Toolkit 2.2* foi usada na simulação, *debugging* e confecção do arquivo JAR do MIDlet ECG. As simulações demonstraram o correto funcionamento do código, executando corretamente as quatro opções disponíveis no menu inicial do MIDlet ECG, que são:

- **Recebe e Grava**
- **Mostra o *Buffer***
- **Envia**
- **Recebe/Grava/Envia**

Para tanto, a máquina onde a simulação foi efetuada estava propriamente conectada à Internet.

A seqüência da figura 4-1 abaixo mostra as telas do *Sun J2ME Wireless Toolkit* durante a execução da opção Recebe/Grava/Envia no MIDlet ECG. O resultado é idêntico à execução no dispositivo real Nokia 6600.

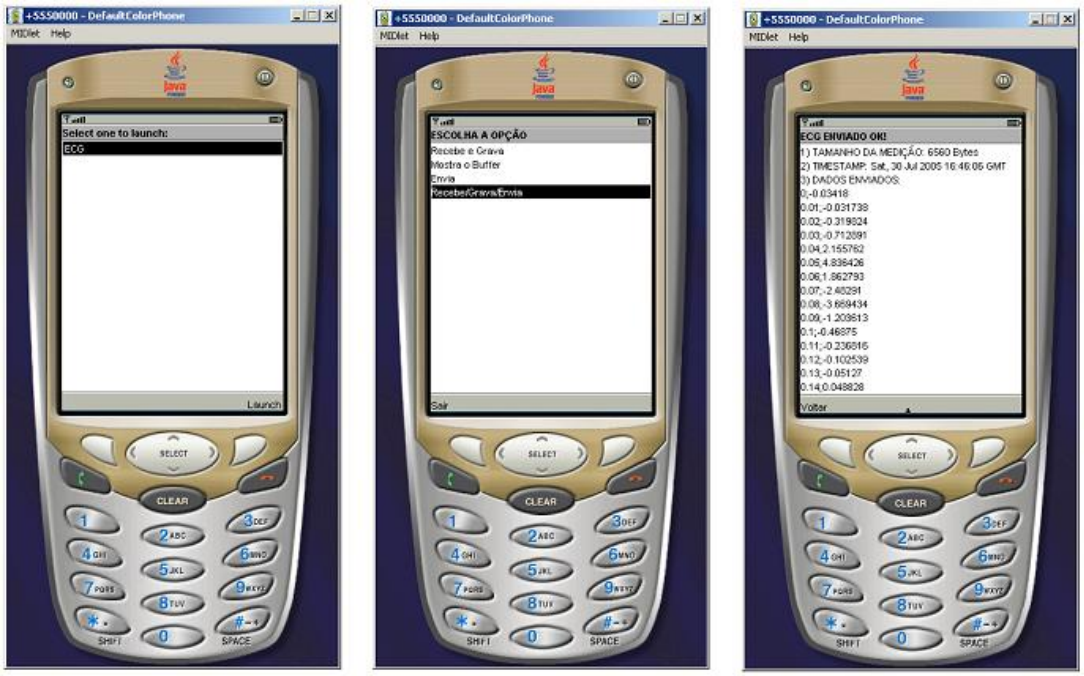


Figura 4-1: Simulação no *Sun J2ME Wireless Toolkit*

4.1.1 Network Monitor

A ferramenta *Sun J2ME Wireless Toolkit* dispõe de uma poderosa opção de monitoramento das conexões de rede, o *Network Monitor*. Para habilitá-lo basta seguir o seguinte caminho: *Edit* > *Preferences* > *Monitor* > *Enable Network Monitoring*. Através do *Network Monitor* é possível certificar-se que a conexão de rede, HTTP POST no nosso caso, está sendo corretamente estabelecida. Isto é tremendamente importante para o *debugging* do código fonte uma vez que nos possibilita "ver" se os dados estão sendo corretamente enviados, ou recebidos. Assim, podemos confirmar se determinado problema está sendo causado pelo aparelho celular, pela rede ou pelo servidor destino. A figura 4-2 ilustra o resultado da execução da opção Recebe/Grava/Envia no MIDlet ECG,

feita com o *Network Monitor* ativado. Nela, à esquerda, podemos notar a seqüência das quatro conexões HTTP, duas de envio e duas de recebimento. Já à direita, é detalhado o conteúdo do POST de envio dos dados.

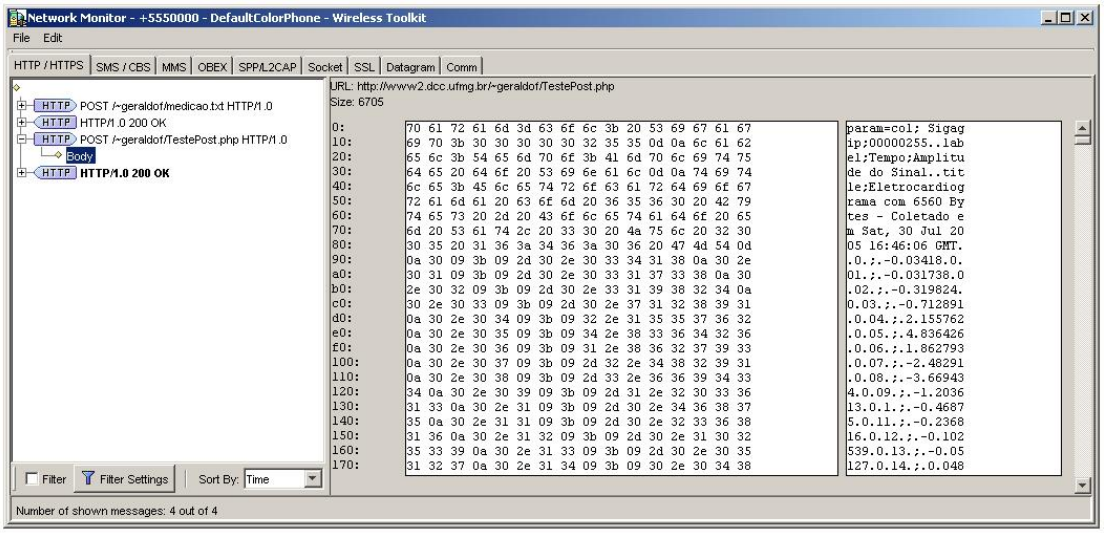


Figura 4-2: Simulação no *Network Monitor*

4.2 Simulação no Nokia Developer's Suíte

A ferramenta *Nokia Developer's Suíte* foi usada em complemento à *Sun J2ME Wireless Toolkit*. Ela nos permite definir, quando da simulação, o tipo do aparelho utilizado. No nosso caso o modelo era o Nokia 6600, aparelho da série 60 e desenvolvido sobre o sistema operacional *Symbian*. Com o *Nokia Developer's Suite* foi então possível aproximar-se bastante da execução no dispositivo real Nokia 6600, principalmente no que tange aos aspectos de interface de usuário.

A seqüência da figura 4-3 abaixo mostra as telas do *Nokia Developer's Suite*

durante a execução da opção Recebe/Grava/Envia no MIDlet ECG. O resultado é idêntico à execução no dispositivo real Nokia 6600.

Figura 4-3: Simulação no *Nokia Developer's Suíte*

4.3 Execução Real no Aparelho Nokia 6600

Confirmando-se o funcionamento do MIDlet ECG nos simuladores, partimos então para os testes de execução real no aparelho celular. O arquivo executável ECG.jar foi gerado pelo *Sun J2ME Wireless Toolkit* através da opção:

Project > *Package* > *Create Package*

Tal opção gera o arquivo ECG.jad, o descritor, e também o executável ECG.jar, gravando ambos no diretório: C:\WTK22\apps\ECG\bin\

O próximo passo foi transferir o arquivo ECG.jar para o aparelho celular. Isto foi feito através de uma conexão infravermelha, estabelecida entre a máquina utilizada (um laptop IBM T41) e o aparelho Nokia 6600. Porém, nada impede que sejam usadas outras opções de transferência, tais como conexão *bluetooth* ou *download* de um servidor *web*. No caso do *download*, é necessário que os arquivos ECG.jar e ECG.jad estejam corretamente disponibilizados no servidor *web*.

Depois de enviado e instalado, o MIDlet ECG repetiu os resultados das simulações, executando de maneira satisfatória todas as opções do menu inicial. Isto é, recebimento dos dados, gravação dos mesmos de maneira persistente, apresentação do conteúdo na tela e, finalmente, envio dos dados de medição para o servidor destino.

Conseguimos confirmar o funcionamento do MIDlet ECG em diferentes cidades, indicando-nos a independência da aplicação frente a diferentes infra-estruturas de rede. Tais cidades foram: São Paulo, Belo Horizonte, Rio de Janeiro, Curitiba e Porto Alegre.

A título de informação, verificamos que o custo médio de envio de uma medição de 6560 *bytes* foi de R$0,12. Tal média de custo foi obtida usando-se a linha pré-paga GSM/GPRS 11-9495-4373 da operadora Claro, na cidade de São Paulo. No total 30 conexões foram feitas para a obtenção da média, havendo pouquíssima variação entre elas.

O resultado da execução real do MIDlet ECG no aparelho Nokia 6600 pode ser visto na figura 3-5 do capítulo 3.

Capítulo 5 – Conclusões e Trabalhos Futuros

Este capítulo está dividido em três seções. Na seção 5.1 mostraremos as dificuldades encontradas ao longo do trabalho, comentando possíveis maneiras de contorná-las. Na seção 5.2 apresentaremos as conclusões que obtivemos depois de toda a implementação. E por fim, na seção 5.3, fazemos sugestões de melhoramentos.

5.1 Dificuldades Encontradas

a) Implementação do MIDlet no Aparelho Celular

Verificamos que existe pouca informação prática sobre a implementação do MIDlet no aparelho celular. Questões como as seguintes surgiram no início do trabalho:

– Como é gerado o código executável? Qual ferramenta é usada para isto?

– Como o código executável é transferido para o aparelho celular?

– Quais arquivos precisam ser instalados? O arquivo JAR ou o arquivo JAD? Ou ambos?

As respostas para as questões acima estão presentes no capítulo 4. Para sanar outras que porventura surjam, sugerimos primeiramente consultar a página da Sun Microsystems [24]. Nela é possível encontrar uma variedade de artigos, *tutorials* e *toolkits*, aplicáveis a princípio a aparelhos de todos os fabricantes. Uma segunda sugestão é consultar o material do fabricante do aparelho utilizado. No nosso caso, consultamos o material do Fórum Nokia [25]. E, por fim, consultar o material disponível na Internet, por meio de algum *site* de busca.

b) Simulações e Execução Real

Mesmo contribuindo de maneira decisiva para o desenvolvimento do MIDlet, as simulações foram também fonte de alguns problemas. Durante a confecção e *debugging* do código fonte, descobrimos que nem sempre o resultado gerado pela simulação é fielmente igual ao resultado da execução no aparelho celular. Por isso sugerimos que a cada nova versão de código, o mesmo seja também executado no aparelho real, para confirmar o funcionamento.

c) Configurações de Rede

Encontramos problemas com as configurações de rede no aparelho celular. Mesmo funcionando nas simulações, as conexões HTTP disparadas pelo MIDlet ECG não funcionavam no aparelho real. O motivo de tal problema era provavelmente alguma restrição de segurança imposta pela operadora da rede celular. As configurações que funcionaram na prática estão descritas na seção 3.5. Sugerimos que, no caso de desenvolvimento de qualquer aplicação que envolva conexões de rede, a operadora seja consultada.

d) Aplicação no Servidor Destino

Como pode ser visto na seção 3.4, nossa solução de monitoramento remoto emprega aplicativos no servidor destino, para recebimento e apresentação dos sinais medidos. Encontramos dificuldades na implementação destas aplicações no servidor *web*. De início nossa idéia era desenvolver aplicativos JSP, que proporcionariam uma solução Java fim-

a-fim. Acontece que a implementação de um aplicativo JSP demandaria configurações no servidor *web*, e provavelmente até um *reset* para a ativação. Isto seria impraticável, em se tratando do próprio servidor *web* do Departamento de Ciência da Computação da UFMG. Como solução, conforme descrito na seção 3.4, utilizamos então um simples aplicativo PHP para recebimento dos dados e gravação no servidor *web*. Um outro aplicativo Java Applet foi desenvolvido numa página HTML, proporcionando a visualização dos dados recebidos em um gráfico. Assim, nenhuma configuração mais profunda foi necessária no servidor *web*.

5.2 Conclusões

Consideramos bastante satisfatórios os resultados alcançados e mostrados no capítulo 4. Observamos nas simulações e também no aparelho celular real o correto funcionamento de todos métodos empregados em nossa aplicação MIDP: interface de usuário, armazenamento persistente, apresentação do resultado e, finalmente, envio para o servidor destino. Apesar das dificuldades encontradas, comentadas na seção 5.1, enxergamos que a área de desenvolvimento de aplicações Java para aparelho celular tende a crescer bastante nos próximos anos. Existem grandes oportunidades ainda inexploradas para desenvolvedores, fabricantes e operadoras, principalmente no que tange ao suporte a nichos, como demonstrado aqui no monitoramento remoto de bio-sinais.

5.3 Trabalhos Futuros

Apontaremos a seguir alguns aspectos nos quais nosso trabalho pode ser melhorado ou simplesmente incrementado, para o caso de prosseguimento em outra dissertação de mestrado ou tese de doutorado.

a) Integração da Solução

Como pode ser visto no capítulo 3, nosso trabalho já apresenta a característica de solução fim-a-fim, mostrando inclusive o gráfico da medição de eletrocardiograma numa página *web*. Tal característica pode ser melhorada ainda mais pelo desenvolvimento de novas aplicações no servidor do hospital. Uma nova aplicação no servidor poderia, por exemplo, administrar o recebimento de medições de vários pacientes, armazenando-as num banco de dados para futura consulta.

b) Segurança

Devido à considerável extensão de nosso trabalho, o aspecto segurança ficou reservado para um futuro desenvolvimento. Com o MIDP 2.0 já é possível implementar conexões seguras HTTPS, conforme mostrado em [26]. Neste caso elas substituiriam as conexões HTTP empregadas em nossa solução. Um outro aspecto de segurança que pode melhorado é a transformação do MIDlet ECG em um MIDlet confiável, *trusted*. Para tanto é necessário que o MIDlet ECG seja certificado conforme a especificação X-509 de chave pública. Em [27] é possível obter mais informações sobre o processo de

certificação digital de um MIDlet.

c) Recebimento de Medições Reais

Na seção 3.3.3 comentamos que o método recebe_medição() implementa uma conexão HTTP POST com um servidor *web*, onde está armazenado um arquivo contendo os dados de uma medição real de eletrocardiograma. Este artifício foi utilizado para emular o recebimento de medições reais, que em um futuro desenvolvimento deve ser implementado através de uma conexão sem-fio *bluetooth* ou cabo RS-232, dependendo do modelo do aparelho celular utilizado. Portanto, a adequação do método recebe_medição() ao recebimento de dados via *bluetooth* ou RS-232 é o aspecto que pode ser melhorado em uma futura versão do MIDlet ECG.

Capítulo 6 – Referências Bibliográficas

[1] Pattichis, C. S., Kyriacou E., Voskarides S., and Istepanian, R.S.H., "Wireless Telemedicine Systems: An Overview", IEEE Antennas and Propagation, Vol.44, 2, pp.143-153, 2002.

[2] Tachkara, S.,Istepanian, R.S.H., Wang, H, and Y. H. Song., “Mobile E-Health; The Unwired Evolution of Telemedicine”, Telemedicine and E-Health Journal, Vol.9, 3, pp. 247-257, 2003.

[3] Woodward, B.; Istepanian, R.S.H.; Richards, C.I., “Design of a telemedical system using a mobile telephone”, Information Technology in Biomedicine, IEEE Transactions on , Vol5, 1,pp.13-5, 2001.

[4] Escola Paulista de Medicina - Universidade Federal de São Paulo. Material do Setor de Telemedicina. Disponível: http://www.unifesp.br/dis/set/disciplina/materialdeapoio/. Acesso em julho de 2005.

[5] Biomed. Projeto “Emergency 112”. Disponível: http://www.biomed.ntua.gr/emergency112/. Acesso em julho de 2005.

[6] S. Pavlopoulos, A. Prentza, E. Kyriacou, S. Marinos, A. Stasis, D. Kalivas, and D. Koutsouris, “Mobile Medical Data (MOMEDA) - A Personalized Medical Information System”, STUDIES IN HEALTH TECHNOLOGY AND INFORMATICS, IOS Press, Vol.72, pp 125-132, 2000.

[7] A. Pitsillides, G. Samaras, M. Dikaiakos, E. Christodoulou, “DITIS: Collaborative Virtual Medical team for home healthcare of cancer patients”, Conference on the Information Society and Telematics Applications, Catania, Italy, 1999.

[8] T.S. Rappaport, “Wireless Communications: Principles and Practice”, Prentice Hall, New Jersey, 2002.

[9] Sun Microsystems, “MIDP 1.0 Specification -Mobile Information Device Profile (JSR-37)”, 2000. Disponível: http://jcp.org/aboutJava/communityprocess/final/jsr037/index.html. Acesso em julho de 2005.

[10] Sun Microsystems, “CLDC Specification - Connected, Limited Device Configuration (JSR-30)”, 2000. Disponível: http://jcp.org/aboutJava/communityprocess/final/jsr030/index.html. Acesso em julho de 2005.

[11] Forum Nokia, “MIDP 1.0: Introduction to MIDlet Programming”, 2004. Disponível: http://www.forum.nokia.com. Acesso em julho de 2005.

[12] ANATEL, “Serviço Móvel Celular”. Disponível: http://www.anatel.gov.br/comunicacao_movel/. Acesso em julho de 2005.

[13] Cisco, “Overview of GSM, GPRS, and UMTS”. Disponível: http://www.cisco.com/univercd/cc/td/doc/product/wireless/moblwrls/cmx/mmg_sg/cmxgsm.htm. Acesso em julho de 2005.

[14] Symbian OS Web Page. Disponível: http://www.symbian.com/. Acesso em julho de 2005.

[15] Kevin Dixon, “Symbian OS Version 7.0s Functional Description”, 2003. Disponível: http://www.symbian.com/technology/7.0s_functional_description2.1.pdf. Acesso em julho de 2005.

[16] Nokia Brasil Web Page. Disponível: http://www.nokia.com.br/. Acesso em julho de 2005.

[17] Nortel Networks Web Page. Disponível: http://www.nortelnetworks.com/. Acesso em julho de 2005.

[18] GSM Association Web Page. Disponível: http://www.gsmworld.com/technology/gsm.shtml. Acesso em julho de 2005.

[19] Sun Microsystems, “MIDP 2.0 Specification -Mobile Information Device Profile (JSR-118)”, 2002. Disponível: http://www.jcp.org/aboutJava/communityprocess/final/jsr118/. Acesso em julho de 2005.

[20] A. Tanenbaum, “Computer Networks”, 4th ed., Prentice Hall, 2003.

[21] W. Stallings, “Data and Computer Communications”, 7th ed., Prentice Hall, 2003.

[22] Saltzstein, W.E., “Bluetooth: The Future of Wireless Medical Technology?”, Medical Device & Diagnostic Industry, 24(2): 44-52, 2002.

[23] Eric Giguere, “Databases and MIDP, Part 1: Understanding the Record Management System”, Sun Technical Article, 2004. Acesso em julho de 2005.

[24] Sun Microsystems Web Page. Disponível: http://java.sun.com/. Acesso em julho de 2005.

[25] Forum Nokia. Disponível: http://www.forum.nokia.com. Acesso em julho de 2005.

[26] Forum Nokia, “MIDP 2.0: Introduction to Secure MIDlet Communication”, 2004. Disponível: http://www.forum.nokia.com. Acesso em julho de 2005.

[27] Forum Nokia, “MIDP 2.0: Tutorial On Signed MIDlets”, 2004. Disponível: http://www.forum.nokia.com. Acesso em julho de 2005

[28] Claudionor J. N. Coelho Jr., Antonio O. Fernandes, Julio C. D. Conway, Fabio L. Correa Jr., Hervaldo S. Carvalho, Jose M. Mata, “A biomedical wearable device for remote monitoring of physiological signals”, 2003 IEEE Conference on Emerging Technologies and Factory Automation, Lisboa - Portugal, v. 2, p. 708-713, 2003.

Capítulo 7 – Apêndices

Apêndice A – Código .java do MIDlet ECG

```
/*=====================================================================
Arquivo: ECG.java
Descrição: este código fonte realiza todas as funções de recebimento, armazenamento
persistente e envio de dados de medição de eletrocardiograma para um servidor remoto.
Elaborado por: Geraldo Ferreira
Data: 28 Junho de 2005
=====================================================================*/



/*=====================================================================
                Importação dos pacotes de classes necessários
=====================================================================*/

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.microedition.io.*;
import java.io.*;
import java.util.*;
import java.lang.String;
import javax.microedition.rms.*;



/*=====================================================================
                Classe ECG
=====================================================================*/

public class ECG extends MIDlet implements CommandListener {
   // display
   Display display = null;

   // itens do menu de opções
   List menu = null;

   // record store rs
   private RecordStore rs = null;

   // string que armazena a data e hora do recebimento/gravação da medição de ECG
   String tempo = null;

   // string contendo os dados de medição para envio ao servidor de web
   String buffer = null;
```

```
// url de recebimento dos dados de medição
String url_recebe = "http://www2.dcc.ufmg.br/~geraldof/medicao.txt";

// url do servidor de web para envio dos dados de medição
String url_envia = "http://www2.dcc.ufmg.br/~geraldof/TestePost.php";

//nome do record store
String NOME_RS="rs_ECG";

//quantidade de bytes recebidos da medição
String num_bytes=null;

//record ID do buffer de dados
int recID_buffer=1;

//record ID do Timestamp
int recID_tempo=2;

//record ID do número de bytes da medição
int recID_bytes=3;



// comandos de interface de usuário
static final Command backCommand =
        new Command("Voltar", Command.BACK, 0);
static final Command mainMenuCommand =
        new Command("Principal", Command.SCREEN, 1);
static final Command exitCommand =
        new Command("Sair", Command.STOP, 2);
String currentMenu = null;



public ECG() {
}

// inicia o MIDlet criando uma lista de opções
public void startApp() throws MIDletStateChangeException
        {

    //cria os 3 records para armazenamento de dados
        cria_records();

    //display das opções do menu
    display = Display.getDisplay(this);
        menu = new List("ESCOLHA A OPÇÃO", Choice.IMPLICIT);
        menu.append("Recebe e Grava", null);
    menu.append("Mostra o Buffer", null);
        menu.append("Envia", null);
        menu.append("Recebe/Grava/Envia", null);
```

```
                menu.addCommand(exitCommand);
                menu.setCommandListener(this);
                mainMenu();

                }


//pausa o MIDlet
public void pauseApp() {display = null; menu = null;}



//destrói o MIDlet
public void destroyApp(boolean unconditional) {notifyDestroyed();}



// Manuseio das opções
public void commandAction(Command c, Displayable d)
{
  String label = c.getLabel();
  if (label.equals("Sair")) { destroyApp(true); }
  else if (label.equals("Voltar"))
        {
          if(currentMenu.equals("menu1") ||
                  currentMenu.equals("menu2") ||
          currentMenu.equals("menu3") ||
                  currentMenu.equals("menu4"))  {mainMenu();}
         }
        else { final List down = (List)display.getCurrent();
              Thread thrd = new Thread()
                                  { public void run()
                                    {
                                    switch(down.getSelectedIndex())
                                      {
                                       case 0: recebe();break;
                                       case 1: mostra_dados();break;
                                       case 2: envia();break;
                                       case 3: recebe_envia();break;
                                      }
                                    }
                                  };thrd.start();
                      }
}



// menu principal
void mainMenu()
{
  display.setCurrent(menu);
```

```
    currentMenu = "Principal";
   }


/*=========================================================================
                recebimento e gravação dos dados
=========================================================================*/

   public void recebe() {

   // abre o record store para posterior armazenagem persistente da medição
   abre_rs();

   // chama o método recebe_medicao
   try{ buffer=recebe_medicao (); }
   catch(Exception ex) { ex.printStackTrace(); }

   // chama o método grava_medicao para gravar o buffer de dados
   try{ grava_medicao (recID_buffer, buffer); }
   catch(Exception ex) { ex.printStackTrace(); }

   // chama o método grava_medicao para gravar o Timestamp
   try{ grava_medicao (recID_tempo, tempo); }
   catch(Exception ex) { ex.printStackTrace(); }

   // chama o método grava_medicao para gravar o número de bytes da medição
   try{ grava_medicao (recID_bytes, num_bytes); }
   catch(Exception ex) { ex.printStackTrace(); }

   // fecha o record store
   fecha_rs();

   currentMenu = "menu1";
   }


/*=========================================================================
                mostra dados gravados no buffer RMS
=========================================================================*/

   public void mostra_dados() {

   // abre o record store para posterior armazenagem persistente da medição
   abre_rs();

   // chama o método le_medicao para ler o record contendo buffer de dados
   try{ buffer=le_medicao (recID_buffer); }
   catch(Exception ex) { ex.printStackTrace(); }

   // chama o método le_medicao para ler o record contendo o Timestamp
```

```
    try{ tempo=le_medicao (recID_tempo); }
    catch(Exception ex) { ex.printStackTrace(); }

    // chama o método le_medicao para ler o record contendo o número de bytes da medição
    try{ num_bytes=le_medicao (recID_bytes); }
    catch(Exception ex) { ex.printStackTrace(); }

    //mostra os dados
    Form f = new Form("ECG GRAVADO NO BUFFER");
    f.append("1) TAMANHO DA MEDIÇÃO: "+ "\r"+num_bytes+" Bytes"+ "\r\n"+
      "2) TIMESTAMP: "+ "\r"+tempo+"\r\n"+"3) DADOS DISPONÍVEIS NO BUFFER: "+
      "\r"+buffer);
      display.setCurrent(f);
      f.addCommand(backCommand);
      f.setCommandListener(this);

    // fecha o record store
    fecha_rs();

    currentMenu = "menu2";

    }


/*=============================================================================
                    envio dos dados
=============================================================================*/

    public void envia() {

    // abre o record store para posterior armazenagem persistente da medição
    abre_rs();

    // chama o método le_medicao para ler o record contendo buffer de dados
    try{ buffer=le_medicao (recID_buffer); }
    catch(Exception ex) { ex.printStackTrace(); }

    // chama o método le_medicao para ler o record contendo o Timestamp
    try{ tempo=le_medicao (recID_tempo); }
    catch(Exception ex) { ex.printStackTrace(); }

    // chama o método le_medicao para ler o record contendo o número de bytes da medição
    try{ num_bytes=le_medicao (recID_bytes); }
    catch(Exception ex) { ex.printStackTrace(); }

    // chama o método envia_dados
    try{ envia_dados(buffer); }
    catch(Exception ex) { ex.printStackTrace(); }

    // fecha o record store
    fecha_rs();
```

```
    currentMenu = "menu3";

    }


/*=========================================================================
                    recebimento, gravação e envio dos dados
=========================================================================*/

    public void recebe_envia() {

    // variáveis usadas para esperar tempo entre recebimento e envio
    boolean flag = false;
    int contador = 0;

    // abre o record store para posterior armazenagem persistente da medição
    abre_rs();

    // chama o método recebe_medicao
    try{ buffer=recebe_medicao (); }
    catch(Exception ex) { ex.printStackTrace(); }

    // chama o método grava_medicao para gravar o buffer de dados
    try{ grava_medicao (recID_buffer, buffer); }
    catch(Exception ex) { ex.printStackTrace(); }

     // chama o método grava_medicao para gravar o Timestamp
    try{ grava_medicao (recID_tempo, tempo); }
    catch(Exception ex) { ex.printStackTrace(); }

     // chama o método grava_medicao para gravar o Timestamp
    try{ grava_medicao (recID_bytes, num_bytes); }
    catch(Exception ex) { ex.printStackTrace(); }

     // espera alguns segundos para prosseguir
     outloop:while (!flag)
     {
     if (contador++ > 5000000) break outloop;
     try { } catch (Exception e) {}
     }

      // chama o método le_medicao para ler o record contendo buffer de dados
     try{ buffer=le_medicao (recID_buffer); }
     catch(Exception ex) { ex.printStackTrace(); }

     // chama o método le_medicao para ler o record contendo o Timestamp
     try{ tempo=le_medicao (recID_tempo); }
     catch(Exception ex) { ex.printStackTrace(); }

     // chama o método le_medicao para ler o record contendo o número de bytes da medição
```

```
    try{ num_bytes=le_medicao (recID_bytes); }
    catch(Exception ex) { ex.printStackTrace(); }

     // chama o método envia_dados
    try{ envia_dados(buffer); }
    catch(Exception ex) { ex.printStackTrace(); }

     // fecha o record store
     fecha_rs();

    currentMenu = "menu4";

    }



/*=========================================================================
                    método de recebimento dos dados
=========================================================================*/

    private String recebe_medicao() throws IOException
    {
    HttpConnection http = null;
    InputStream iStrm = null;
    String str = null;

    try
    {
      // cria a conexão HTTP para recebimento dos dados de medição
      http = (HttpConnection) Connector.open(url_recebe);

      // configura o método da conexão como POST
      http.setRequestMethod(HttpConnection.POST);

      // configura o cabeçalho tipo do conteúdo
      http.setRequestProperty("Content-Type","application/x-www-form-urlencoded");

      // se a conexão está OK, continua
      if (http.getResponseCode() == HttpConnection.HTTP_OK)
      {
         // obtém o timestamp da medição por um campo do cabeçalho
         tempo=http.getHeaderField(0);

         // obtém número de bytes recebidos da medição por um campo do cabeçalho
         num_bytes = String.valueOf(http.getLength());
         System.out.println("valor de num_bytes " + num_bytes);

        // abre o InputStream para leitura dos cabeçalhos da conexão http
        iStrm = http.openInputStream();
        int length = (int) http.getLength();

        if (length != -1)
```

```
        {
         // se o comprimento está disponível, lê os dados de uma vez
         byte serverData[] = new byte[length];
         iStrm.read(serverData);
         str = new String(serverData);
        }

       else
        {
         // caso o comprimento não esteja disponível, lê os dados caracter por caracter
         ByteArrayOutputStream bStrm = new ByteArrayOutputStream();
         int ch;
         while ((ch = iStrm.read()) != -1) bStrm.write(ch);
         str = new String(bStrm.toByteArray());
         bStrm.close();
        }
     }
   }
   finally
   {
     // fecha o InputStream e a conexão http
     if (iStrm != null) iStrm.close();
     if (http != null) http.close();
   }

   return str;

  }



/*==========================================================================
                método de fechamento do record store
==========================================================================*/

   public void fecha_rs()
                {
                try { rs.closeRecordStore();}
                catch (Exception e) {db(e.toString());}
                }



/*==========================================================================
                método de impressão das exceções
==========================================================================*/

   private void db(String str)
                {
                System.err.println("mensagem de exceção: " + str);
                }
```

```
/*=====================================================================
                método de gravar no record store
=====================================================================*/

   public void grava_medicao(int record, String dado) {

   try {
         //  data output stream<- byte array output stream
         ByteArrayOutputStream baos = new ByteArrayOutputStream();
         DataOutputStream dos = new DataOutputStream(baos);

         // data output stream <- medição dado
         dos.writeUTF(dado);

         // limpa dado bufferizado, se existir
         dos.flush();

         // byte array data <- data output stream
         byte[] data = baos.toByteArray();

         // grava a medição no record de ID especificado
         rs.setRecord(record, data, 0, data.length);

         // mostra o resultado da gravação na tela do celular
         Form f = new Form("ECG RECEBIDO e GRAVADO! ");
                f.append("1) TAMANHO DA MEDIÇÃO: "+"\r"+num_bytes+" Bytes"+ "\r\n"+ "2)
TIMESTAMP: "+
                  "\r"+tempo+"\r\n"+"3) DADOS RECEBIDOS e GRAVADOS: "+
                              "\r"+buffer);
                display.setCurrent(f);

          // após mostrar o resultado, fornece a opção de voltar ao meu inicial
          f.addCommand(backCommand);
                f.setCommandListener(this);
       }
      catch (IOException e)
                { System.out.println("Erro de IO"); }
      catch (ArrayIndexOutOfBoundsException e)
                {System.out.println("Registro muito grande"); }
      catch (InvalidRecordIDException e)
                { System.out.println("ID inexistente"); }
      catch (RecordStoreNotOpenException e)
                { System.out.println("O Record Store esta fechado"); }
      catch (RecordStoreException e)
                { System.out.println("Outro erro"); }
   }



/*=====================================================================
```

```
                método de ler o record store
==========================================================================*/

   private String le_medicao(int record) {
      String leitura = "";
      try {
         //obtém o tamanho do record
         int recordSize = rs.getRecordSize(record);

         // byte array do tamanho do record
         byte[] data = new byte[recordSize];

         // byte array input stream <- byte array
         ByteArrayInputStream bais = new ByteArrayInputStream(data);

         // data input stream <- byte array input stream
         DataInputStream dis = new DataInputStream(bais);

         // número de bytes da medição
         int numBytes = rs.getRecord(record, data, 0);

         // lê o record , alocando o resultado na string leitura
         leitura = dis.readUTF();

         // fecha o byte array input stream e o data input stream
         bais.reset();
         bais.close();
         dis.close();
      }
      catch (IOException e)
                { System.out.println("Erro de IO"); }
      catch (ArrayIndexOutOfBoundsException e)
                { System.out.println("Registro muito grande");}
      catch (InvalidRecordIDException e)
                { System.out.println("ID inexistente");}
      catch (RecordStoreNotOpenException e)
                { System.out.println("O Record Store está fechado"); }
      catch (RecordStoreException e)
                { System.out.println("Outro erro");}
      return leitura;
   }



/*==========================================================================
                método de envio dos dados
==========================================================================*/

   public void envia_dados (String buffer) throws IOException
   {
     int recordID = 1;
     try {
```

```
                byte[] recData = new byte[50000];

                // lê do byte array
          ByteArrayInputStream bin = new ByteArrayInputStream(recData);

         // bin -> din
         DataInputStream din = new DataInputStream(bin);

          rs.getRecord(recordID, recData, 0);

                buffer = din.readUTF();

          // fecha o bin e o din
          bin.reset();
          bin.close();
          din.close();

          } catch (RecordStoreException e ) {  db(e.toString());}

                // System.out.println(medicao);

       HttpConnection http = null;
       OutputStream os = null;
       InputStream in = null;
       ByteArrayOutputStream baos = null;

     try {
                // cabeçalho para construção do gráfico no servidor web
                String header = "col"+";"+" Sigagip"+";"+"00000255"+"\r\n"+
                "label"+";"+"Tempo"+";"+"Amplitude do Sinal"+"\r\n"+
                "title"+";"+"Eletrocardiograma com "+num_bytes+" Bytes - Coletado em
"+tempo+"\r\n";
                //System.out.println(tempo);
                String medicao = header+buffer;

                //abre a conexão http com o servidor web para envio dos dados
                http = (HttpConnection)Connector.open(url_envia);

          // configura o método de conexão como POST
          http.setRequestMethod(HttpConnection.POST);

          // configura o cabeçalho content type
          http.setRequestProperty("Content-Type", "application/x-www-form-urlencoded");

          // abre a output stream e envia os dados
                os = http.openOutputStream();
          byte data[];
                data = ("param="+medicao).getBytes();
          os.write(data);
          //os.flush();

                //imprime o eletrocardiograma enviado na tela do aparelho celular
```

```
                Form f = new Form("ECG ENVIADO OK!");
                f.append("1) TAMANHO DA MEDIÇÃO: "+"\r"+num_bytes+" Bytes"+ "\r\n"+
                                "2) TIMESTAMP: "+"\r"+tempo+"\r\n"+"3) DADOS ENVIADOS: "+
                                "\r\n"+buffer);
                display.setCurrent(f);

        // após mostrado o resultado, fornece a opção de voltar ao menu inicial
        f.addCommand(backCommand);
        f.setCommandListener(this);

         // lê a resposta do servidor, se houver
                in = http.openInputStream();
                byte[] b = new byte[16];

         // fecha o input stream e a conexão http
                } finally {
                                if (in != null) in.close();
                                if (http != null) http.close();
                                }
   }


/*=====================================================================
                método para criação dos 3 records de armazenamento
=====================================================================*/

public void cria_records() {
try {
      abre_rs();
      int id=1;

    // se não existe os 3 records no record store rs, eles são então criados
    if (rs.getNumRecords() == 0)
    {
    while (id!=3)
    {
    String nome="vazio";
    ByteArrayOutputStream baos = new ByteArrayOutputStream();
    DataOutputStream dos = new DataOutputStream(baos);
    dos.writeUTF(nome);
    dos.flush();
    byte[] data = baos.toByteArray();
    id = rs.addRecord(data, 0, data.length);
    baos.close();
    dos.close();
    }
    }

    fecha_rs();

    }
```

```
    catch (IOException e)
                { System.out.println(" Erro de IO"); }
    catch (RecordStoreFullException e)
                {System.out.println("Não existe espaço disponível");}
    catch( RecordStoreNotOpenException e )
                {System.out.println(" O Record Store esta fechado");}
    catch( RecordStoreException e )
                { System.out.println("Outro erro");}

   }


/*=====================================================================
                método para abrir o record store store
=====================================================================*/

   public void abre_rs() {
   try {
         rs = RecordStore.openRecordStore(NOME_RS, true);
      }
      catch (RecordStoreNotFoundException e)
                { System.out.println("RecordStore inexistente");
      }
      catch (RecordStoreException e)
                { System.out.println("Outro erro"); }
   }


/*=====================================================================
                método para deletar record store
=====================================================================*/

   public void deleteRMS() {
   if (RecordStore.listRecordStores() != null)
     { try
         {
          RecordStore.deleteRecordStore(NOME_RS);
         }
         catch (Exception e)
                  { db(e.toString()); }
     }
  }


} // FIM GERAL
```

Apêndice B – Especificações do Aparelho Nokia 6600

Especificações Técnicas:

- Tri-band GSM E900/1800 + 1900.
- Antena Interna.
- Codificadores de voz: HR, FR, EFR.
- GPRS.
- Câmera VGA Integrada (640 x 480 pixels) com lente de alta qualidade.
- Memória interna de 7 Mbytes.
- Dimensões: 10,86cm (comprimento) x 5,82cm (largura) x 2,37cm (espessura).
- Cartão de memória adicional (32 MBytes).
- Conexão sem fio por Bluetooth ou Infravermelho[6].
- Alerta vibratório interno.
- SIM card (1.8 e 3.0 V).

Características:

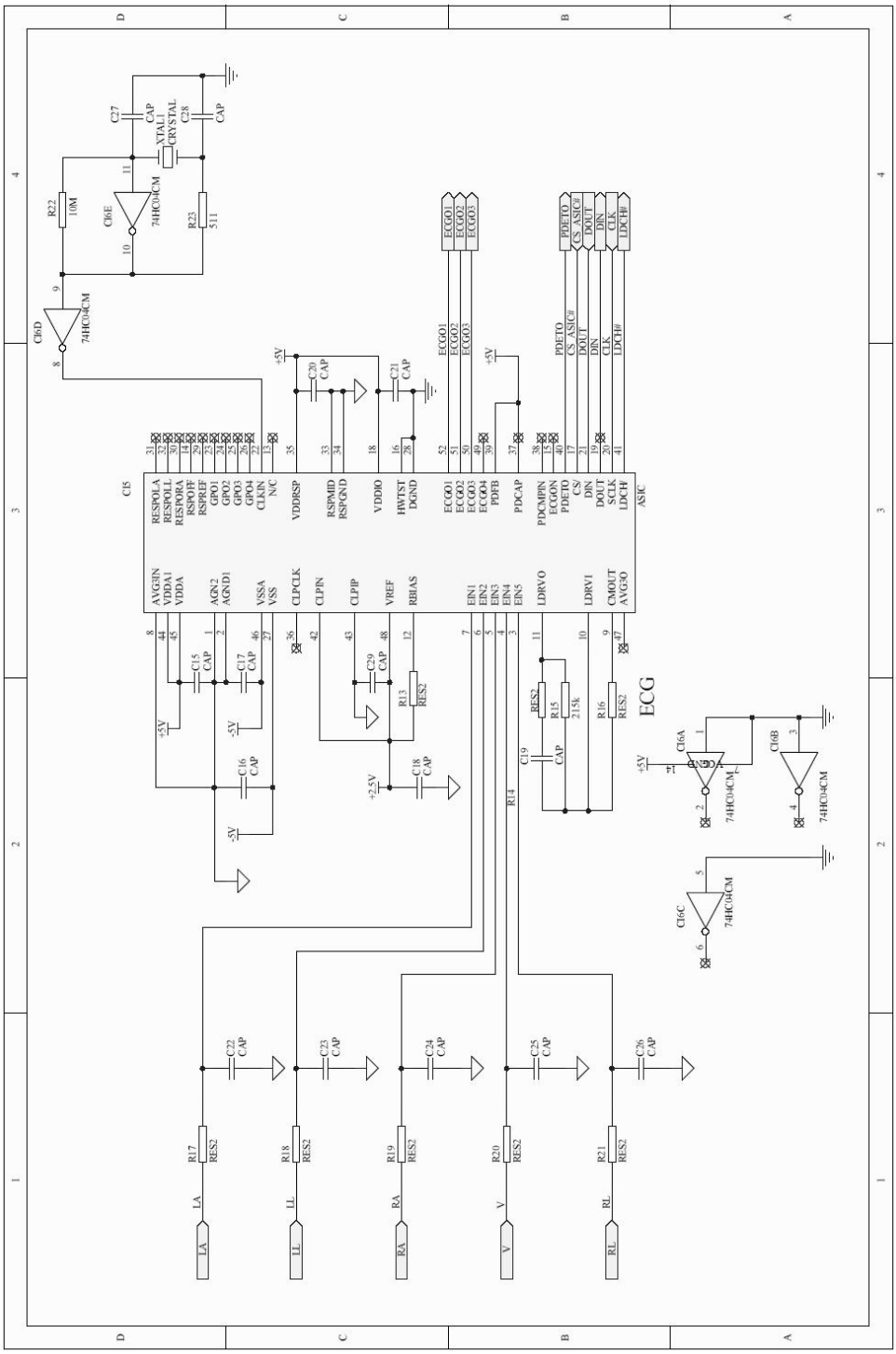
- Câmera Integrada.
- Botão de atalho para conexão rápida.
- Viva-voz integrado.
- Relógio com real time (conversão mundial).
- Câmera com 3 modos (Padrão/Retrato/Noite) e 2x Zoom Digital.
- Gravador de vídeo que suporta tamanhos e 2x Zoom Digital.

- Galeria de imagens e vídeos.
- Carregador de imagens.
- Mensagens Multimídia.
- Mensagens de texto SMS.
- Mensagens com fotos.
- E-mail.
- Previsibilidade de entrada de texto (facilitador de digitação de texto).
- Lista de contatos com imagens.
- Calendário.
- Lista de tarefas.
- Notas.
- Gravador de voz.
- Calculadora.
- Relógio.

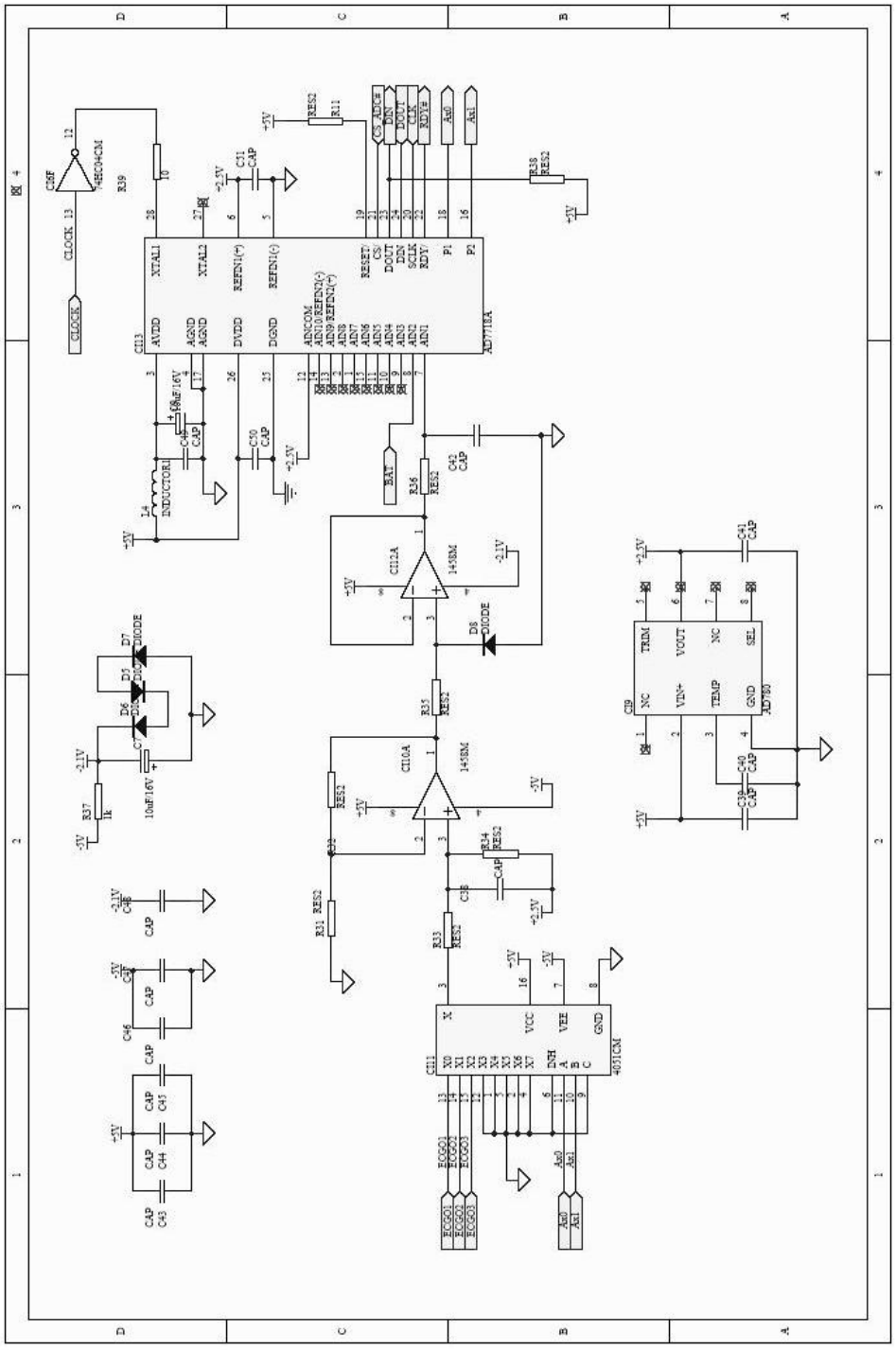
Apêndice C – Código .php do Servidor Destino

```
<?php

if ($HTTP_POST_VARS['param'])
{
   $param = $HTTP_POST_VARS['param'];

   echo($param);

   $conteudo = $param;

   $arquivo = "./grafico/txt/m_0197.txt";

   //TENTA ABRIR O ARQUIVO TXT
   if (!$abrir = fopen($arquivo, "w")) {
   echo "Erro abrindo arquivo ($arquivo)";
   exit;
   }

   //ESCREVE NO ARQUIVO TXT
   if (!fwrite($abrir, $param)) {
   print "Erro escrevendo no arquivo ($arquivo)";
   exit;
   }

   echo "Arquivo gravado com sucesso !!";

   //FECHA O ARQUIVO
   fclose($abrir);
}
else
   {
      echo("falhou.....parametro/arquivo não chegou!");
   }
?>
```

Apêndice D – Diagrama do Bloco ECG ASIC



Apêndice E – Diagrama do Bloco ADC



Apêndice F – Diagrama do Bloco uC 8051