UNIVERSIDADE FEDERAL DE PERNAMBUCO

GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

CENTRO DE INFORMÁTICA

2005.1

IDENTIFICAÇÃO E EXEMPLIFICAÇÃO DE ASPECTOS DA COMPUTAÇÃO DISTRIBUÍDA NO DESENVOLVIMENTO DE APLICAÇÕES PARA CELULARES USANDO J2ME E BLUETOOTH

TRABALHO DE GRADUAÇÃO

Aluno – André Luiz de Sousa Lima, alsl@cin.ufpe.br

Orientador – Carlos André Guimarães Ferraz, cagf@cin.ufpe.br

Recife, Agosto de 2005

Agradecimentos

Primeiramente agradeço aos meus *pais e irmã* pelo carinho e amor dedicados a mim, e por me proporcionarem com muito esforço boas oportunidades de estudo em instituições de qualidade.

À minha namorada *Luciana* pelo carinho e amor dedicado a mim durante nosso tempo juntos, além do suporte e compreensão das necessidades de conclusão desse trabalho.

Ao meu grande amigo *Ivo Frazão* que me incentivou e ajudou bastante na confecção, elaboração e finalização desse trabalho.

Aos meus colegas de trabalho e de faculdade, pois neles encontrei muito boas relações de amizade apesar do pouco tempo de convívio, e que por causa dessas amizades não perder em momento algum o gosto pela Ciência da Computação.

E ao meu orientador *Carlos Ferraz*, pela confiança depositada em mim e pelo suporte dado durante o desenvolvimento desse trabalho.

Resumo

Este trabalho apresenta as principais características das tecnologias J2ME e Bluetooth no tocante ao desenvolvimento de aplicações wireless para dispositivos móveis. A apresentação dessas características ocorre através de comparações com outras tecnologias semelhantes, detalhamento do funcionamento tanto da plataforma J2ME como da tecnologia Bluetooth além das restrições de desenvolvimento características desses tipos de aplicações. Também é sugerido nesse trabalho, algumas técnicas de programação para otimização de desempenho e algumas diretrizes que conduzirão os desenvolvedores a uma melhora no processo de desenvolvido dessas aplicações.

Abstract

This work presents the main characteristics about J2ME and Bluetooth Technologies in wireless applications for mobile devices development. The presentation of these characteristics occurs through comparisons with others similar technologies, detailing of the functioning in such a way of platform J2ME as of the Bluetooth technology beyond the characteristic restrictions of development of these kinds of applications. Also it is suggested in this work, some techniques of programming for optimization of performance and some lines of directions that will lead the developers to an improvement in the development process of these applications.

Índice

Lista de Figuras

Lista de Tabelas

2 – Introdução

A cada dia, a nova geração de dispositivos móveis (Capítulo 3), vem ganhando mais e mais espaço em nossas vidas. Como exemplo da inserção desses dispositivos no nosso dia-a-dia, podemos citar o uso de telefones celulares, PDAs (Personal Digital Assistants) e laptops por um numero cada vez mais crescente de pessoas ao redor do mundo.

Isso está de certa forma ligado à evolução das funcionalidades desses dispositivos. Agendas eletrônicas, câmeras fotográficas e jogos são alguns exemplos de novas funcionalidades inseridas em celulares e PDAs.

Junto com o a inserção de novas funcionalidades para dispositivos móveis, também tem aumentado a disponibilização de novos meios de comunicação entre esses dispositivos. Como exemplos desses novos meios, temos a tecnologia Infravermelho que, por exemplo, já é bastante utilizada em controles remotos de Televisores e portas de garagens, a tecnologia WI-FI e a tecnologia Bluetooth que serão mais detalhadas nos **capítulos 3** e **4**.

2.1 – Contexto

Com o lançamento da API Java (JSR-82) voltada para dispositivos móveis, que dá suporte a comunicação wireless entre esses dispositivos através da tecnologia Bluetooth, o desenvolvimento de aplicações distribuídas para dispositivos móveis está tendo um impulso fora do previsto, crescendo rapidamente. Esse crescimento já pode ser visto levando em consideração o aumento dos números de celulares e outros dispositivos móveis que são produzidos com suporte à Bluetooth. Nesse contexto, novas aplicações inovadoras são esperadas, para que problemas do dia-a-dia sejam mais facilmente resolvidos.

Em contrapartida ao rápido crescimento da utilização de redes Bluetooth em dispositivos móveis, está à escassez de trabalhos relacionados ao desenvolvimento de softwares distribuídos para esses dispositivos, bem como exemplos de aplicações que façam uso dos conceitos apresentados nesses trabalhos.

2.1 – Objetivos

Para solucionar essa escassez de referências, este trabalho tem dois objetivos básicos: a produção de uma literatura referência que servirá para possíveis trabalhos que venham a fazer uso das tecnologias J2ME e Bluetooth, no desenvolvimento de aplicações distribuídas para celulares, através da identificação de alguns aspectos referentes à Computação Distribuída que devem ser levados em consideração durante o desenvolvimento desse tipo de aplicações. E a exemplificação de desses aspectos através de um estudo de caso de uma aplicação previamente desenvolvida.

3 - Dispositivos Móveis

Este capítulo tem como objetivo, dar uma breve descrição sobre o que são, como funcionam, quais softwares e tecnologias são utilizados pelos vários dispositivos móveis que estão hoje no mercado, além de apresentar algumas características de como eles realizam suas comunicações.

3.1 - Introdução aos dispositivos móveis

Chamamos de dispositivos portáteis ou móveis, os dispositivos que podem carregar consigo mesmo sua fonte de energia. Como exemplo desses dispositivos, temos os PDA's (Palm e Pocket PC), celulares e notebooks em geral.

Desde o aparecimento do primeiro PDA lançado pela Apple em 1993, o que ocasionou a popularização da sigla PDA, a cada dia que se passa temos novidades no âmbito das melhorias das tecnologias e serviços oferecidos por esses dispositivos. Hoje, os palmtops têm telas de ótima qualidade, chips velozes, executam programas complexos, tocam MP3, gravam voz e vídeos, rodam jogos e fotografam. Ou seja, são realmente acessórios muito úteis para o dia-a-dia.

Muitos dos dispositivos móveis, principalmente os celulares, disponíveis no mercado apresentam recursos de transmissão de dados, contudo, a transferência desses dados via serviços da operadora ainda é muito caro aqui no Brasil.

Um sistema móvel é definido como uma rede de comunicações por rádio que permite mobilidade contínua por meio de células. A comunicação sem fio, por outro lado, implica em comunicação por rádio sem necessariamente requerer passagem de uma célula para outra durante a conversação [18].

Os sistemas móveis são convencionados nas seguintes gerações:

- **Primeira geração**: celulares analógicos, somente comunicação por voz, surgiram em meados dos anos 70 e inicio dos anos 80 [18];

- **Segunda geração**: em meados dos anos 90, os primeiros modelos de celulares digitais começaram a surgir, fazendo uso de uma tecnologia que fazia a multiplexação dos canais de transmissão de informação através do tempo, e com eles foi possível à comunicação de dados e voz;
- **Geração 2,5**: é representada pelas novas tecnologias de transmissão por pacotes, como GPRS e CDMA 1x e, principalmente, pelos serviços diferenciados, como o acesso permanente à web, possíveis pelo aumento das velocidades e tarifação por volume de dados. A cada ligação de voz há um canal de dados à disposição do usuário, basta acessá-lo. Porém não é possível acessar a web e receber ligações simultaneamente;
- **Terceira geração ou 3G** : inaugurada no Japão em 01 de outubro de 2001. O serviço de 3G da empresa japonesa **NTT DoCoMo** chama-se **FOMA** (Freedom of Mobile Multimedia Access) e utiliza a tecnologia **W-CDMA3** (Wideband-Code Division Multiple Access) . Esse novo serviço oferece qualidade na transmissão de voz, com mínima interferência e barulho e suporte a diversos conteúdos de multimída, como transmissão de vídeos, imagens, música e jogos, além de grande capacidade de comunicação de dados. Também oferece cobertura em diversos países;

De um modo geral, as previsões são de que o atual conceito de células geográficas que deu origem ao popular "celular", deverá ser expandido para células de tamanhos diferentes (pico, micro e macro-células); cada célula deverá adotar uma tecnologia diferente para a "interface aérea" ou transmissão dos sinais de rádio; velocidades elevadas de transmissão e integração completa de tecnologias de telecomunicações e eletrônica de estado sólido servirão de suporte para sistemas multimídia com enorme poder de processamento.

Como disse David Levin, presidente-executivo da Symbian, em entrevista à revista Info Exame de setembro de 2003, “Nos últimos cem

anos construímos telefones para os ouvidos. Agora, podemos construir telefones que se conectem com os Olhos" [22].

3.2 - Sistemas Operacionais

Os sistemas operacionais que estão dividindo o mercado brasileiro são o Palm OS, o Symbian OS e o Windows CE. Ainda encontram-se o BREW, tecnologia da Qualcomm e em PDAs alguns produtos com Linux.

As empresas IBM, Hitachi, NEC, Panasonic, Philips, Samsung, Sony, Sharp e Toshiba estão se unindo para levar o Linux para o celular. Montaram o CE Linux Fórum, cujo objetivo é o desenvolvimento de soluções e aplicativos de código aberto com a mesma dinâmica da comunidade Linux.

Na home page desse grupo[1] há um fórum de discussão e seção para as especificações que serão publicadas sobre o CE Linux.

3.2.1 – Symbian OS

Symbian OS é o nome dado a um projeto integrado pelos maiores fabricantes mundiais de telefones móveis, como a Sony Ericsson, Nokia, Motorola e Siemens, que procura desenvolver um padrão comum para o futuro.

Trata-se de um sistema operacional 32 bits multitarefa robusto, desenhado especialmente para o ambiente de comunicação sem fio (Wireless) e para as restrições dos telefones celulares. É baseado em cinco pontos chaves: pequenos dispositivos móveis, o mercado de massa, conexão wireless ocasional, variedade de produtos e plataforma aberta para desenvolvimento de terceiros.

Symbian OS pretende promover o mercado de massa para aparelhos de informação móvel de duas maneiras:

- Através do desenvolvimento de software, aplicações e ferramentas para desenvolvimento de aparelhos móveis multimídia, tais como PDAs, smartphones e comunicadores.

[1] www.celinuxforum.org

- A universalização dos padrões para interoperabilidade dos aparelhos para informação móvel.

A plataforma aberta é dirigida para o desenvolvimento de aplicações com compromisso com os padrões abertos, suporta o desenvolvimento principalmente em C++ e Java, facilitado pelo uso de SDK's. A versão, 7.0, ganha suporte a MIDP 2.0, melhorias em conectividade, além de um framework multimídia multithreaded, que traz melhorias de desempenho na manipulação de vídeo e sons. A comunicação se faz através dos protocolos HTTP[2] ou WAP[3].

Existem bibliotecas (API´s) para gestão de dados, gráficos, e texto. Pode-se também gerenciar Banco de Dados, ClipBoard e acesso à impressoras, uma infraestrutura pronta, permitindo criar aplicações adicionais em cima de Agenda, gerenciador de contatos, planilhas, ferramenta de ajuda, ferramenta de gráficos, e conversor de formato de texto. O ambiente de aplicação comporta a possibilidade de desenvolver em vários conjuntos de caracteres, particularmente os orientais (Japonês, Chinês).

3.2.2 - Windows Mobile e Pocket PC

Em meados de junho de 2003, a plataforma para dispositivos móveis, handhelds e celulares inteligentes, da Microsoft passou a se chamar Windows Mobile. Essa nova marca foi apresentada junto com a nova versão do sistema operacional, Windows Mobile 2003. Segundo a Microsoft será mais fácil identificar o sistema operacional dos equipamentos, lembrando que Pocket PC continua sendo a denominação para o hardware que roda a plataforma da Microsoft.

A novidade do Windows Mobile 2003 é a facilidade para conexões sem fio, isto é, suporte nativo às tecnologias Wi-Fi e Bluetooth sem a necessidade de baixar drivers adicionais para acessá-las. Oferece ainda integração com o Microsoft Exchange Server 2003, para a sincronização

[2] HyperText Transfer Protocol
[3] Wireless Application Protocol

de e-mails e agendas pessoais e mais suporte ao Windows Media Player 9 series.

3.2.3 - BREW

A plataforma de desenvolvimento de aplicações móveis Binary Runtime Environmente for Wireless, (ambiente binário de tempo de execução para aplicativos sem fio), Brew, que pertence à Qualcomm, oferece uma solução completa e aberta para configuração de dispositivos, além de desenvolvimento, distribuição, faturamento e pagamento de aplicativos.

A solução Brew inclui plataforma de aplicativos Brew e ferramentas de portagem para fabricantes de dispositivos; o Brew SDK; para desenvolvedores e o Brew Distribution System, BDS, sistema de distribuição do Brew, controlado e gerenciado pela Qualcomm, o que lhes permite obter aplicativos dos desenvolvedores, comercializa-los e coordenar os processos de faturamento e pagamento.

Ele apresenta alguns atrativos que a diferencia de sistemas como o J2ME, como a capacidade de processamento off-line, recursos multimídia e grande riqueza gráfica. Alem de não definir limites à aplicações, que por sua vez podem acessar áreas de memória de qualquer parte do celular, compartilhar dados com outros aplicativos.

O fato do C/C++ permitir acesso via ponteiros à memória, o que o Java não permite, ajuda o BREW a ser mais flexível, porém exige que o programador saiba o que está fazendo no tocante ao desenvolvimento da aplicação. Por esse motivo o custo de desenvolvimento no BREW é mais alto, pois toda aplicação tem que ser testada e certificada pela Qualcomm para garantir que não se introduza na rede aplicações mal intencionadas ou simplesmente mal feitas e com isso ponha-se em risco toda a credibilidade da plataforma BREW.

O Brew está presente em todos os chips CDMA fabricados pela Qualcomm ,e comercialmente, foi lançado por algumas operadoras no mundo, como a Verizon, nos Estados Unidos e a KTF na Coréia. Não há compatibilidade com os aparelhos TDMA.

Tanto a plataforma quanto seu suporte técnico são oferecidos pela Qualcomm Internet Services (QIS), divisão responsável pelo desenvolvimento de serviços e aplicações baseados na tecnologia CDMA.

O Brew foi projetado para permitir que o desenvolvedor escolha a linguagem para a codificação dos programas. Oferece suporte nativo à linguagem C/C++, mas aceita a integração de browsers e aplicações em outras linguagens, inclusive Java e XML.

3.3 - Comunicação entre dispositivos móveis

A comunicação entre os vários dispositivos móveis já citados, pode se dar através de várias formas (meios de comunicação). Dentre essas formas de comunicação, podemos citar os famosos meios de comunicação via airtime[4] como GPRS, Bluetooth, WI-FI, Infravermelho, além da maneira usual que não é airtime (conexão via cabos).

A seguir daremos uma breve descrição de como é realizada a comunicação de dispositivos móveis através desses meios.

3.3.1 – GPRS

O GPRS (General Packet Radio Service) é um serviço de dados que permite o envio e o recebimento de informações através de pacotes de uma rede sem fio que suporte IP e X.25 como por exemplo a rede de celulares móveis que utiliza a comunicação via GSM.

A tecnologia GPRS é freqüentemente chamada de tecnologia 2.5G porque representa a primeira etapa que uma operadora GSM empreende quando inicia a transição à terceira geração (3G) dos sistemas de comunicação móveis.

Como utiliza o mesmo padrão universal utilizado na Internet (o protocolo IP), o GPRS otimiza as comunicações de dados através das redes móveis. Sendo baseado em pacotes, permite que a utilização da infra-estrutura da rede de comunicação existente se dê somente quando uma operação de transmissão e ou recepção é requerida ao invés de

[4] O limite de tempo que utiliza um sistema celular, entre o tempo que efetua com sucesso uma chamada e o termo dessa chamada.

estabelecer um link permanente. Isto incrementa a eficiência do meio de comunicação e melhora a qualidade dos serviços de conexão.

O grande atrativo da tecnologia GPRS está na possibilidade de se manter uma conexão "permanente" de dados ("always on") e assim, os usuários não precisam conectar o sistema toda vez que necessitarem de acesso aos serviços. Outra vantagem é que a tarifação é feita apenas sobre os dados efetivamente transmitidos e dessa forma o usuário não paga pelo tempo de conexão. Sendo uma tecnologia de rede sem fio, o GPRS oferece velocidades de dados de 115Kbps e um throughput[5] médio de 30Kbps a 40Kbps.

Uma rede baseada em GPRS permite o acesso à Internet em qualquer tempo e lugar, possibilitando a navegação em páginas WEB, o envio e recebimento de e-mails e outras facilidades. Entretanto, para estabelecer um acesso Internet via GPRS é necessário um aparelho celular que suporte a tecnologia GPRS e ativar esse serviço junto à operadora de telefonia celular.

3.3.2 – Bluetooth

Bluetooth é uma tecnologia de curto alcance via rádio, que propicia uma conexão wireless para terminais móveis como notebooks, impressoras, telefones móveis, e PDA's e os fazem poder trocar dados entre si. [5] Ou seja, a tecnologia Bluetooth veio para tentar resolver alguns problemas como a necessidade de uso de cabos para realizar a conexão e a substituição dos diferentes tipos de entrada de hardware utilizados por esses dispositivos.

Cada dispositivo Bluetooth pode se comunicar com quaisquer outros dispositivos dentro de uma pequena rede wireless, a qual chamamos de piconet. Todos os dispositivos mantém uma conexão lógica com os outros dispositivos e essa conexão lógica apenas é convertida em uma conexão física quando existe a necessidade de troca de dados entre os dispositivos.

[5] Capacidade total de um canal em processar e transmitir dados durante um determinado período de tempo.

A tecnologia bluetooth opera a uma freqüência que varia entre 2,402 e 2,480 GHz, essa banda ISM[6] é livre e não é licenciada em todo o mundo. Uma técnica de mudança de faixas de freqüência muito rápida é usada para fazer com que a freqüência utilizada para a troca de dados seja trocada várias vezes a cada segundo. Com isso, o Bluetooth evita a interferência e o choque de dados causados pela troca de dados entre vários dispositivos numa mesma piconet.

Os componentes da rede Bluetooth, são identificados através de um endereço único, de uma forma bastante parecida com o endereço MAC numa rede Ethernet. Esse endereçamento e identificação são usados para regular autorização, funcionalidades e segurança dos vários participantes.

A velocidade sinalizada é de 1Mbit/s das quais 721Kbit/s estão disponíveis para transferência de voz e de dados.O alcance máximo do Bluetooth é de 10 ou 100 metros, dependendo da potência utilizada pelo dispositivo. No entanto, existem problemas no que se diz respeito às barreiras físicas como paredes por exemplo, que ocasiona uma redução desse alcance da piconet.

3.3.3 – WI-FI (802,11b)

A tecnologia WI-FI (Wireless Fidelity) é composta por produtos que são responsáveis pela comunicação sem fio e que atendem às normas 802.11 estabelecidas pela IEEE (Institute of Electrical and Eletronic Engineers). Dentre essas normas, a mais conhecida é a norma 802.11b, que faz uso da freqüência 2,4 GHz, e transfere dados a uma velocidade de 11Mb/s.

Essa tecnologia esta destinada em sua primeira instância à computadores desktops, pois seu uso em dispositivos móveis esta restrita a alguns poucos modelos de PDA's e ainda não está disponível para a maioria dos telefones celulares.

Atualmente, o uso dessa tecnologia esta relacionado com a vontade das empresas de diminuir a quantidade de ligações físicas para a realização da comunicação, como exemplo, podemos citar uma LAN

[6] Industrial, Scientific, and Medical.

que tenha cinco computares ligados a um servidor e a uma impressora sem a necessidade de um único cabo de rede.

As redes WI-FI tem dois modos de funcionamento:

- Em modo de infra-estrutura cliente-servidor, onde cada dispositivo WI-FI se comunica com um ponto de acesso para poder ter acesso a outros dispositivos na rede;
- Em topologia de rede ad-hoc, onde os dispositivos trocam informações diretamente uns com os outros, ou seja, sem a necessidade de um ponto servidor para fazer o controle de aspectos da rede como segurança ou controle de banda;

Em geral, para a implementação de uma rede WI-FI, deverá ser feita à instalação de um hardware de recepção de sinais de rádio e de pontos de acesso, para a emissão e recepção de sinais. Uma das principais desvantagens do WI-FI, esta relacionado ao alto gasto de energia quando comparado a outras tecnologias wireless, pois para que ele possa cobrir uma área grande de rede, é necessário um alto consumo de energia, já que o alcance do WI-FI está diretamente relacionado com a potência do transmissor dos sinais de rádio. Ou seja, quanto mais distante um dispositivo estiver, mais forte deverá ser o sinal.

3.3.4 – IrDA – Transmissão via Infravermelho

Os sistemas de transmissão Infravermelho (IR) usam freqüências do espectro muito altas, um pouco abaixo do espectro da luz, para transmitir informações. O Comprimento de onda se situa entre 780 ηm a 950 ηm. Não é necessário uma licensa para o uso dessa faixa de comprimento de onda.

Os sinais de IR não conseguem penetrar em objetos opacos, por isso são usados direcionados ou com difusão (reflexão). Objetos escuros são usados para a absorção dos raios enquanto objetos claros são usados para reflexão desses raios.

A distância da transmissão direcionada é muito pequena (poucos metros) e seu uso é restrito a Redes Pessoais (PANs) e algumas aplicações específicas em Redes Locais sem fio (WLANs). No caso da

difusão, ou reflexão, a transmissão usando Infravermelho não necessita ser direcionada, mas fica restrita a ambientes fechados (salas), pois os equipamentos sem fio dependem das estruturas do ambiente, tais como tetos e paredes, para refletir os sinais das portadoras.

A transmissão Infravermelho de alta performance é impraticável em Computação Móvel, só sendo usada em subredes fixas.[12]

O IrDA (Infrared Data Association) é uma organização fundada em 1993 e dedicada ao desenvolvimento de padrões para transmissão sem fio usando infravermelho. Os produtos usando IrDA começaram a aparecer em 1995.

Através de portas IrDA, um laptop ou PDA podem trocar dados com um computador desktop ou então usar uma impressora sem a necessidade conexão física através de cabos/fios. Ele é usado normalmente para transmissão de dados entre os seguintes equipamentos: notebooks, desktops, handhelds, computadores, impressoras, telefones, pagers, modems, máquinas fotográficas,dispositivos de acesso a rede, equipamentos médico-industriais e relógios. [12]

Algumas características do IrDA são:

- Possui uma base instalada de mais de 50 milhões de unidades;
- É comprovadamente uma tecnologia que funciona em todo mundo;
- Suporta um grande número de plataformas diferentes;
- Funciona Ponto-a-Ponto, substituindo cabos e fios;
- Total compatibilidade com os padrões anteriores;
- Aplicações do tipo controle remoto de TV (point-and-shoot), com no máximo 30º de ângulo de desvio;
- Não interfere com outros dispositivos eletrônicos;
- Baixo nível de segurança em dispositivos estacionários;
- Alta taxa de transmissão, vai de 9600b/s a 4Mb/s e está em fase de melhoramento para suportar 16Mb/s;
- Distância máxima entre transmissor e receptor de poucos metros;

O IrDA é composto de uma camada física IrDA-SIR (IrDA Serial IR), que provê uma conexão half-duplex (envia ou recebe, cada um a seu tempo) com o dispositivo receptor. O protocolo da camada de enlace é o IrLAP (Infrared Link Access Protocol), uma adaptação do HDLC (High-Level Data Link Control), e o
IrLMP (Infrared Link Management Protocol) é usado para gerenciar o handshaking e prover a multiplexação de duas ou mais transmissões diferentes ao mesmo tempo. [12]

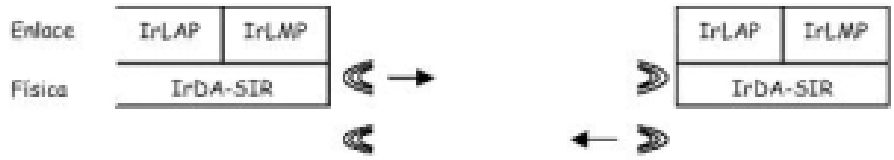

Figura 3.1 - Conexão half-duplex, a transmissão só pode ser em um sentido de cada vez [12]

4 – A Tecnologia Bluetooth

Este capítulo tem como principal objetivo, decorrer sobre o funcionamento da tecnologia Bluetooth: Seus protocolos, sua forma de comunicação, além de algumas comparações com outras tecnologias de comunicação como infravermelho e WI-FI (802,11b).

4.1 – Como surgiu o Bluetooth

O nome Bluetooth veio de Rei da Dinamarca Harald Blätand, que era chamado de bluetooth. Esse rei se destacou dos demais reis, pois conseguiu unir a Dinamarca e a Noruega sobre a religião católica em meados do século 10.

Em 1994, a Ericsson deu inicio aos primeiros estudos e pesquisas para a construção de uma tecnologia wireless que ligasse dispositivos moveis e acessórios entre si. Posteriormente, em 1997, a Ericsson fundou o Bluetooth SIG (Bluetooth Special Interested Group). Assim, outras empressas passaram a difundir e a fazer uso dessa tecnologia pelo mundo.

As primeiras empresas a juntar-se à Ericsson foram: IBM, Intel, Nokia e Toshiba. Mais tarde em 1999, após o lançamento da versao 1.0 da especificação Bluetooth, o grupo aumentou com o acréscimo de mais quatro empresas: 3Com, Agere, Microsoft e Motorola. Hoje em dia, o Bluetooth SIG é composto por mais de 2000 membros. [6]

4.2 – Dispositivos Bluetooth

A comunicação wireless entre dispositivos móveis é realizada tanto pela luz como através de sinais de rádio. A tecnologia infravermelho é a forma de comunicação wireless mais comum de se encontrar dentre os dispositivos que já estão no mercado, e ela utiliza a luz como principio de comunicação.

Por outro lado, o uso de dispositivos Bluetooth para a realização dessa comunicação vem crescendo muito nos últimos anos e essa tecnologia é um exemplo de comunicação através de sinais de Rádio.

A tabela a seguir mostra as varias formas de comunicação wireless e suas respectivas faixas de freqüência de rádio.

Item	**Alcance de Freqüência**
Rádio AM	535kHz – 1.6MHz
Controladores de Porta de Garagens	40MHz
Babás Eletrônicas	49MHz
Canais de TV 2-6	54 MHz – 88MHz
Rádio FM	88MHz – 108 MHz
Canais de TV 7-13	174 MHz – 216 MHz
Canais de TV 14-83	512 MHz – 806 MHz
Telefones Celulares CDMA	824 MHz – 894 MHz
Telefones Celulares GSM	880 MHz – 960 MHz
Telefones Sem Fio	900 MHz
Global Positioning System (GPS)	1.227 GHz – 1.575 GHz
Telefones Celulares PCS	1.85 GHz – 1.99 GHz
802.11b	2.4 GHz – 2.483 GHz
802.11g	2.4 GHz – 2.483 GHz
Bluetooth	2.4 GHz – 2.483 GHz
Telefones Sem Fio	2.4 GHz
802.11ª	5.15 GHz – 5.35 GHz

Tabela 4.1: Freqüências de Rádio Comuns. Adaptado de [6]

A seguir, mostraremos exemplos de como os diversos dispositivos Bluetooth são implementados em forma de hardware e como esses hardware são disponibilizados para o consumo.

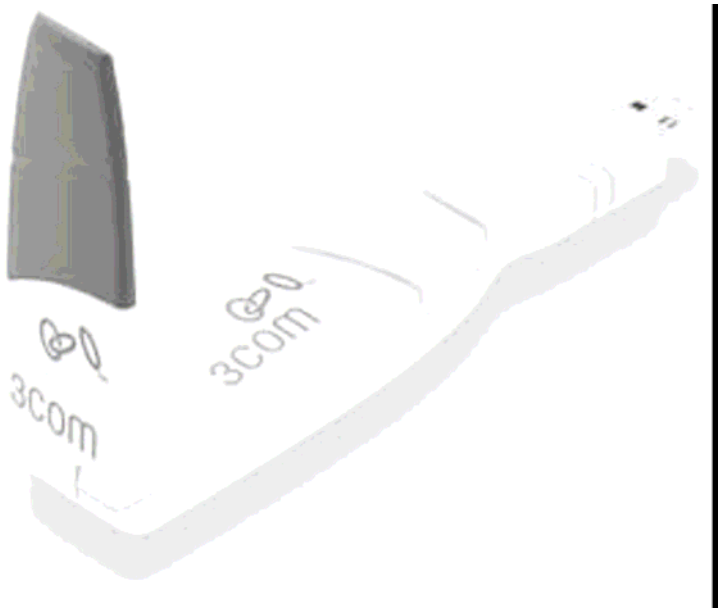

Figura 4.1: Adaptador USB Bluetooth da 3COM [6]

Figura 4.2: O BlueCore da CSR. Uma solução em chip que inclui um microprocessador, RAM, I/O, Controlador e uma implementação Bluetooth num único pacote. [6]

Figure 4.3: Bluetooth Card da Palm para dispositivos da Palm OS.[6]

4.2.1 – Comunicação Ponto-a-Ponto e Multiponto

Um fator importante que distingue os dispositivos Bluetooth dos vários outros dispositivos capazes de se comunicar via wireless é o fator de potencialidade de conexão. Ou seja, se um dispositivo Bluetooth pode apenas se comunicar com outro dispositivo via comunicação Ponto-a-ponto, então ele irá se comunicar com apenas um dispositivo por vez.

A figura a seguir demonstra esse exemplo:

Figura 4.4: Você pode se comunicar com apenas um dispositivo Bluetooth caso você tenha um hardware Bluetooth que suporta conexão ponto-a-ponto.

Hoje em dia, comunicação ponto-a-ponto não é necessariamente uma coisa ruim. Por exemplo, se você está utilizando seu telefone para

comunicar-se com outro dispositivo, você provavelmente só necessitará de uma única conexão, pois por conta de limites do próprio celular, você não conseguirá fazer duas ou mais coisas ao mesmo tempo.

Por outro lado, dispositivos como impressoras ou notebooks, que são capazes de se comunicar com mais de um dispositivo ao mesmo tempo, podem fazer uso da conexão multiponto que a tecnologia Bluetooth oferece. Com essa conexão, esses dispositivos são capazes de comunicar-se com até sete outros dispositivos ao mesmo tempo, desde que esses dispositivos estejam ao alcance da rede Bluetooth.
A figura a seguir ilustra um exemplo de conexão multiponto:

Figura 4.5: Você pode conectar até sete dispositivos Bluetooth ao mesmo tempo caso você tenha um hardware com capacidade de conexão multiponto.

4.2.2 - Piconets e Scatternets

Quando dois ou mais dispositivos Bluetooth estão sobre o mesmo alcance e estabelecem uma conexão, uma personal area network (PAN) é criada [6]. Existem duas formas diferentes de PAN, a primeira é chamada de Piconet e outra é chamada de Scatternet.
A figura a seguir ilustra como é uma piconet.

Figure 4.6: PAM em forma de Piconet [6]

Uma piconet Bluetooth tem dentre seus elementos, um dispositivo que chamaremos de dispositivo Master e até mais sete elementos Slaves. Não importando que tipo de dispositivo seja (notebook, telefone celular, impressora, PDA), o dispositivo que inicia uma conexão será o dispositivo Master. Subsequentemente, os dispositivos que aceitam essa conexão serão os Slaves.

No entanto, quando um novo dispositivo deseja conectar-se a uma rede onde já existem um Máster e seus sete slaves, acontece a criação de uma scatternet, ou seja, caso algum dos dispositivos slaves da piconet tenha capacidade de se conectar multiponto, esse dispositivo se tornara um Máster de uma nova piconet e o dispositivo que queria juntar-se à antiga piconet ira participar dessa nova piconet e assim irá fazer parte da nova scatternet.

A figura a seguir ilustra como é o funcionamento de uma scatternet.

Figure 4.7: Pelo menos um dos participantes de uma scatternet é ao mesmo tempo Mater e Slave.

4.2.3 – Classes de Potência de Dispositivos Bluetooth

Os dispositivos de hardware Bluetooth são divididos em três classes, a tabela a seguir mostra essas classes e as descreve com relação à potência utilizada e ao alcance em metros.

Classes	Potência	Alcance
Classe 1	100 mW	100 metros
Classe 2	2,5 mW	20 metros
Classe 3	1 mW	10 metros

Tabela 4.2: **classes de potência de dispositivos Bluetooth**

4.3 – A Pilha de protocolos Bluetooth

Um computador seja ele desktop ou notebook, é um exemplo de um poderoso dispositivo. Ele tem um processador, memória, barramento, disco rígido e outros objetos que compõem sua configuração de hardware. Por outro lado, infelizmente o computador não tem a habilidade de lidar sozinho com alguns periféricos como leitores e gravadores de CD/DVD, placas de vídeo, modem, impressoras e outros periféricos. Para que esses periféricos sejam aproveitados satisfatoriamente, os computadores necessitam de drivers para prover

ao computador as funcionalidades do dispositivo e dar uma transparência de uso para estes [6].

A pilha de protocolos Bluetooth funciona de maneira similar à esse relacionamento. A pilha é um agente de controle, podendo tomar forma de software, hardware, firmware[7] ou uma combinação desses três elementos e assim provê o controle do dispositivo Bluetooth. A pilha Bluetooth provê duas funcionalidades básicas, a comunicação com outros dispositivos Bluetooth e o controle sobre o próprio dispositivo. Ou seja, sem a pilha de protocolos o dispositivo não conseguiria fazer nada para a realização da comunicação, como exemplo poderíamos dizer que seria um computador sem Sistema Operacional, ou mesmo um computador sem os principais dispositivos periféricos como mouse, teclado etc.

A seguir (**Sessão 4.4**) detalharemos cada elemento da pilha do protocolo Bluetooth.

4.4 – Camadas da pilha de protocolo Bluetooth

A pilha de protocolos Bluetooth pode se dividir em dois itens principais: Camadas e Perfis [6]. Dividindo cada um dos elementos da pilha bluetooth em componentes individuais, poderemos chamar esses elementos de camada da pilha, ou seja, todas as camadas do protocolo bluetooth formam a pilha de protocolos. A seguir detalharemos cada um desses elementos, no tocante ao funcionamento e para que servem essas camadas.

Elementos da Pilha Bluetooth:

- Host Controller Interface (HCI)
- Logical Link Control and Adaptation Protocol (L2CAP)
- Service Discovery Protocol (SDP)
- RFCOMM
- Telephony Control Protocol Specification (TCS-BIN)
- Wireless Access Protocol (WAP)
- Object Exchange (OBEX)

[7] Firmware é um programa de computador armazenado permanentemente numa PROM, ROM ou de modo semipermanente numa EPROM

- Bluetooth Network Encapsulation Protocol (BNEP)
- Human Interface Device Protocol (HID)

Você pode notar que algumas dessas camadas são chamadas de protocolos, isso acontece porque eles são subprotocolos da pilha de protocolo Bluetooth. Porém outros protocolos também presentes na pilha tais como TCP/IP, OBEX, WAP não foram criados pelo Bluetooth SIG, e sim foram adicionados à pilha durante a criação da mesma.
A figura a seguir mostra um ilustração da pilha de protocolos Bluetooth:

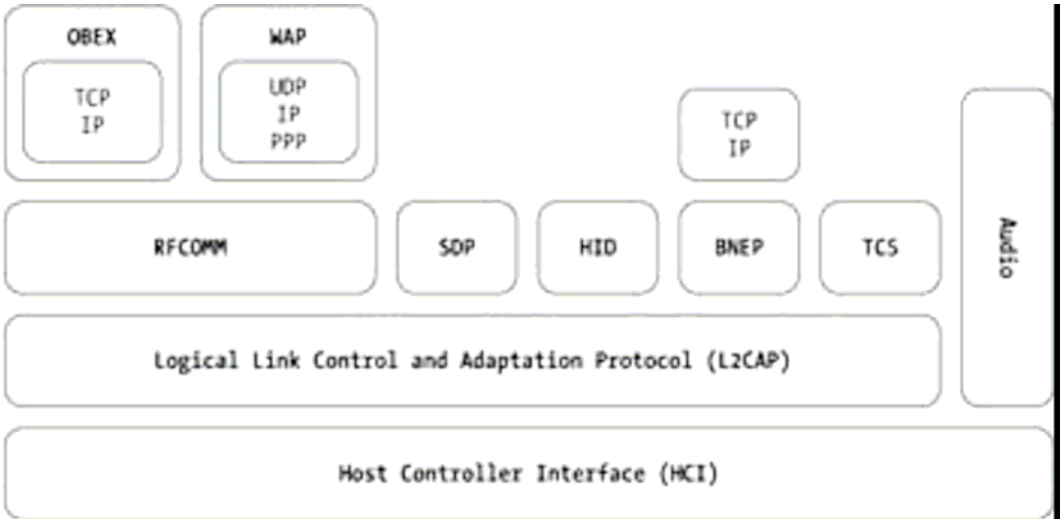

Figura 4.8: Pilha de Protocolos Bluetooth [6]

4.4.1 - Host Controller Interface (HCI)

A interface de controle de Host (HCI) é uma camada do software por onde passam todos os dados da aplicação para o dispositivo físico do bluetooth propriamente dito.

Vamos supor que você esteja tentando comunicar-se via wireless do seu computador (Host) a algum outro dispositivo qualquer via uma interface Bluetooth que esteja instalado em sua porta USB. Assim você precisará de uma camada que entenda as chamadas da porta USB e possa enviar informações para as camadas superiores da pilha. Qualquer informação, seja ela dado ou voz obrigatoriamente irá passar pela interface de controle de host (HCI).

4.4.2 - Logical Link Control and Adaptation Protocol (L2CAP)

O protocolo de adaptação e controle lógico de link (L2CAP) é a camada da pilha por onde todo e qualquer dado irá passar. L2CAP provê algumas características interessantes e poderosas como, por exemplo, segmentação de pacotes e remontagem de dados, tão bem como um protocolo de multiplexação de dados.

Um exemplo de como o L2CAP funciona, é por exemplo no caso de uma aplicação que esteja enviando pacotes de dados muito grandes para serem enviados via Bluetooth, sendo assim, o L2CAP quebra esses pacotes grande e os envia em pacotes menores para seu destino. Outra funcionalidade do L2CAP, é que ele é o responsável pela remontagem desses pacotes quando eles são recebidos por outro dispositivo, ou seja, ele funciona inversamente quando ele esta recebendo dados e não enviando.

Como um protocolo de multiplexação, L2CAP pode receber dados de mais de um protocolo superior como por exemplo os protocolos SDP e RFCOMM (**Sessões 4.4.3 e 4.4.4**). Porém como já foi dito, o protocolo L2CAP apenas transmite dados, uma vez que áudio tem acesso direto à camada de HCI.

4.4.3 - Service Discovery Protocol (SDP)

O protocolo de descoberta de services (SDP) é utilizado por dispositivos Bluetooth para a realização da procura e associação de serviços à outros dispositivos. Por exemplo, caso você esteja usando um telefone celular e queira utilizar um fone de ouvido Bluetooth[8], o SDP ira se responsabilizar pela descoberta desse serviço. Isso é feito, de maneira que o próprio fone de ouvido disponibiliza uma mensagem que por exemplo seria interpretada como “Eu sou um fone de ouvido, para me utilizar conecte-me ao seu dispositivo!”. Daí por diante é só fazer com que o telefone celular mantenha uma conexão e faça o uso desse

[8] Celulares e fones da motorola por exemplo

serviço para que o usuário possa falar com outras pessoas através do fone de ouvido Bluetooth.

4.4.4 - RFCOMM

O protocolo RFCOMM é assim chamado, pois faz referencia às portas seriais COMM1 e COMM2. Esse protocolo é conhecodo como uma porta serial para comunicação wireless ou protocolo de substituição de cabos. Um exemplo de como o RFCOMM funciona é quando um PDA está com sua pilha de bluetooth ativa e sincroniza seus dados com um computador através de um dispositivo Bluetooth usando a camada RFCOMM. Dessa maneira, eles irão trocar seus dados como se estivessem interligados através de cabos físicos e utilizando a porta serial.

4.4.5 - Telephony Control Protocol Specification (TCS, TCS Binary, TCSBIN)

A especificação do protocolo de controle de telefone (TCS, TCS Binary, TCS-BIN) é usada para enviar sinais de controle para dispositivos que querem empregar as potencialidades de áudio dentro da pilha Bluetooth. Como exemplo, um telefone sem fio Bluetooth poderá usar essa camada para indicar à base que ele deseja encerrar a ligação, ou mesmo deixar a chamada em espera.

4.4.6 - Wireless Access Protocol (WAP)

O protocolo WAP, é um protocolo já existente e consolidado no mercado e por isso ele foi adotado pelo Bluetooth SIG e incorporado à pilha de protocolos da tecnologia Bluetooth. Por sua vez, para que o protocolo WAP seja utilizados pela pilha Bluetooth, deverão estar presentes também os protocolos de rede PPP[9], IP[10] e UDP[11] [6].

[9] Point-to-Point Protocol
[10] Internet Protocol
[11] User Datagram Protocol

4.4.7 - Object Exchange (OBEX)

OBEX é um protocolo de comunicação inicialmente definido pela IrDA (Infrared Data Association).

Assim como o WAP, OBEX não foi definido Bluetooth SIG e adotado a pilha de protocolos Bluetooth, pela sua capacidade de ser bastante transparente quando se deseja transferir objetos como arquivos entre dispositivos Bluetooth.

OBEX não requer que os protocolos TCP/IP estejam presentes na pilha, mas o fabricante do dispositivo está livre para realizar essa implementação caso seja requerido pelo grupo que está desenvolvendo esse novo dispositivo Bluetooth.

4.4.8 - Bluetooth Network Encapsulation Protocol (BNEP)

O protocolo de encapsulamento de rede Bluetooth é uma camada da pilha que permite que outro protocolo de rede transmita dados sobre o Bluetooth, esse protocolo é o Ethernet. Existem várias formas de se implementar os protocolos de rede TCP/IP em um dispositivo Bluetooth, mas o protocolo BNEP é a escolha mais usual pois ele encapsula pacotes TCP/IP em pacotes do protocolo L2CAP antes de eles serem entregues ao próprio protocolo L2CAP.

4.4.9 - Human Interface Device Protocol (HID)

O protocolo de interface humana é um outro protocolo que foi incorporado pelo Bluetooth SIG à pilha de protocolos Bluetooth. Ele teve sua definição na especificação do USB e serve para especificar e delimitar regras de transmissão de dispositivos controlados basicamente por humanos como teclados, mouses, controles remotos ou de videogames.

4.5 - Profiles

Os profiles de Bluetooth são um conjunto de funcionalidades projetadas para dispositivos Bluetooth[5]. Um exemplo desse aspecto que esta presente na tecnologia Bluetooth é no caso de existirem um PDA e telefone celular ambos com dispositivos Bluetooth e capazes de

se comunicarem entre si, ou seja ambos tem sua pilha Bluetooth implementadas. Sendo assim, para que possamos determinar se os dois objetos possam, por exemplo, interagir entre si e trocarem, suas agendas telefônicas, ou mesmo tentar determinar se poderemos navegar através de um PDA Internet Browser via modem Bluetooth do telefone celular, que o Bluetooth SIG definiu o que chamamos de profiles.

Para responder aos exemplos dados no parágrafo anterior, digamos que ambos os dispositivos, o telefone celular e o PDA, tenham suporte ao profile de Sincronização (Synchronization Profile) a fim de possibilitar a troca de dados sincronizados entre si. Assim como para eles poderem trocar dados de objetos como arquivos .vcf de um PDA para o telefone celular, ambos os dispositivos devem ter suporte ao Object Push Profile. E finalmente para que a navegação na internet através de um modem wireless, ambos os dispositivos devem dar suporte ao Dial-Up Networking Profile.

No apêndice A, você terá uma lista de todos os profiles do protocolo Bluetooth juntamente a uma breve descrição sobre cada um desses profiles.

4.5.1 – Interdependências de Profiles

Os profiles são muito dependentes uns dos outros e todos eles dependem principalmente do funcionamento do perfil de acesso genérico (Generic Access Profile). Os Bluetooth profiles foram projetados para serem “building blocks”, ou seja, os profiles que estejam mais acima num bloco de profiles existentes dependerão das funcionalidades dos profiles que estejam em camadas mais baixas nesse bloco.

Observe a figura a seguir para ter uma idéia de como está à organização de dependências desses profiles.

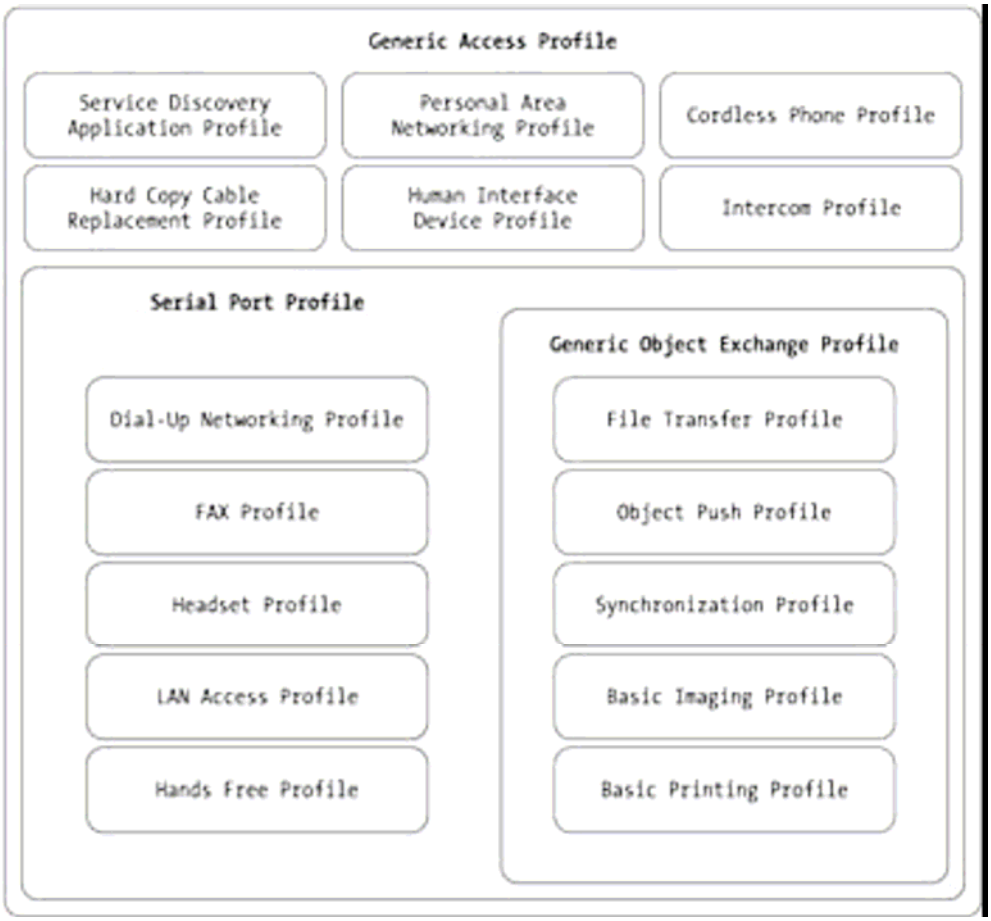

Figura 4.9: Interdependências de perfis Bluetooth [6]

Como exemplo, para que possamos dizer que um PDA suporte ao Synchronization Profile, ele também deve ter suporte ao Generic Object Exchange Profile, Serial Port Profile e finalmente ao Generic Access Profile, pois sem esses outros profiles, o Synchronization Profile não irá funcionar.

4.6 – Bluetooth e Outros Dispositivos

A comparação entre a tecnologia Bluetooth e outras tecnologias é um assunto onde se deve observar, principalmente, não só as limitações técnicas e suas vantagens com relação às outras, mas sim, devemos também observar os objetivos de criação de cada uma das tecnologias e a que problemas essas tecnologias foram projetadas para resolver para resolver.

4.6.1 – Bluetooth vs. Infravermelho

Sabemos que a comunicação wireless entre dispositivos não é uma coisa muito recente para os padrões da computação. A comunicação entre dispositivos via infravermelho vem acontecendo já há algum tempo. Porém alguns empecilhos ao infravermelho é que os dispositivos devem estar a poucos passos de distância, além de que a comunicação via infravermelho acontece "eye to eye", ou seja, as interfaces infravermelho devem estar apontadas uma para a outra, caso contrário não existirá ou ficará a caso da reflexão de outras superfícies sua comunicação.

Bluetooth supera a primeira limitação por possuir um alcance de no mínimo 10,00 m. E ele supera a segunda limitação, pois funciona via sinais de ondas de rádio, sendo assim a transmissão não depende que as interfaces físicas estejam apontadas umas com as outras para que a comunicação aconteça e consequentemente não dependerá também de superfícies refletoras.

4.6.2 - Bluetooth vs. WI-FI

O padrão de comunicação 802.11b também conhecido como WI-FI é um padrão que como o Bluetooth utiliza a freqüência de 2.4GHz[12]. Porém essa tecnologia tem propósitos diferentes da tecnologia Bluetooth.

O objetivo principal da tecnologia WI-FI, é a conexão entre dispositivos de relativo grande porte como computares e notebooks que geralmente possam ser utilizados em sua maior parte do tempo junto à uma tomada de energia elétrica.

Em geral, a tecnologia WI-FI consegue ter um alcance de até 90 metros à uma banda de 11 Mb/s. Essa tecnologia é muito útil para administradores de redes que querem aumentar suas LANs por locais por onde a passagem de cabos é inconveniente.

Por outro lado, Bluetooth veio para tentar resolver problemas de conexão para dispositivos de pequeno porte como PDAs, telefones

[12] Mesmo operando na mesma freqüência, e numa mesma área de alcance, não necessariamente essas tecnologias irão interferir uma na outra.

celulares à uma curta distancia (10 metros) e a uma banda passante de no máximo 1Mb/s.

Justamente por ter um alcance menor e menos banda passante, permite aos dispositivos Bluetooth um consumo de energia até 500 vezes menor comparado ao consumo de dispositivos WI-FI [6] o que faz muita diferença para dispositivos móveis no tocante a durabilidade de suas baterias.

Por essas diferentes características, podemos dizer que dificilmente uma irá substituir a outra. Ainda mais quando o Bluetooth é, por exemplo, deficiente na transferência de arquivos de tamanho grande entre os dispositivos e não possui um grande alcance comparado ao WI-FI[13]. Já pelo lado da tecnologia WI-FI, sabemos que dificilmente pode ser usado para comunicação entre periféricos, requer muita potência de energia para provê a comunicação, geralmente perde dados em transferências de pequenos dados e não foi projetado para comunicação de voz.

[13] Exceto pela classe 1 de 100m de alcance.

5 – Tecnologia Java para Dispositivos Móveis

Java não é somente uma linguagem de programação [9]. Ela é formada por um conjunto de elementos que envolvem, além da sintaxe da linguagem de programação, os formatos de arquivos (.java e .class), um conjunto de APIs (classes, componentes, frameworks), e uma Máquina Virtual Java (JVM).

Já a linguagem de programação Java propriamente dita, é uma linguagem de alto nível, que segue o paradigma de programação orientado a objetos.

As principais características de Java:

- Compilada e Independente de Plataforma (Portabilidade):
- Segurança
    - Tipagem forte de dados,
    - Uso de exceções
- Garbage Collector - Coleta de lixo eficiente para a liberação de memória.
- Multithreading – Suporte a programação concorrente

Com essas qualidades, Java pode ser utilizada para a criação de vários tipos de aplicativos, desde aplicações *standalone* (local) até aplicações designadas para serem controladas pelo software que as executa, tais como APPLETS (são pequenos programas
escritos em Java que podem ser inseridos em documentos de hipertextos – HTML) que são carregados pela *web* e executados dentro de um *browser*, SERVLETS, que são aplicações para serem executados dentro de um servidor *web*, MIDLETs, que são
aplicações designadas para serem executados dentro de dispositivos móveis (telefones celulares, *pagers*, etc), XLETS, que são aplicações para receptores de TV digital ou S*et Top Boxes* (dispositivo que estende a funcionalidade de um receptor de TV digital), entre outros [klessis_wescley].

5.1 - Introdução ao J2ME

Java 2 Micro Edition (J2ME) é a edição da linguagem Java para ser usada em dispositivos de computação portáteis e móveis, que possuem as seguintes características: mobilidade, baixa capacidade de processamento e pouca memória disponível, alimentação elétrica por baterias, pequenas áreas de display, e limitados e variados métodos de entrada e saída. Alguns exemplos destes dispositivos são os telefones celulares, pagers,
PDAs, Palms, entre outros [10].

J2ME não define um novo tipo de Java, mas adapta sua plataforma para que seja possível executar programas em dispositivos, como os citados anteriormente. Sendo assim, todo programa desenvolvido para J2ME poderá ser executado sem nenhum problema nas edições Standard (J2SE) e Enterprise (J2EE), assumindo que as APIs usadas estejam presentes para estas plataformas.

Essencialmente, a plataforma J2ME busca criar um conjunto de conceitos para homogeneizar o desenvolvimento em pequenos dispositivos, transparecendo ao desenvolvedor todos os detalhes proprietários do fabricante como arquitetura de hardware e sistema operacional do dispositivo sobre o qual esteja trabalhando.

No passado, todo dispositivo era oferecido com um conjunto fixo de funcionalidades cuja programação era realizada exclusivamente pelo fabricante, sobre uma tecnologia altamente proprietária. Através de J2ME, torna-se possível desenvolver, atualizar e instalar novas aplicações segundo as necessidades particulares de cada usuário [10]. Além disso, é possível depois de feito o download da aplicação para o dispositivo, trabalhar conectado à rede (on-line) ou desconectado da rede (off-line), no caso de se estar desconectado da rede, quando a rede estiver disponível pode-se fazer a sincronização dos dados e das informações utilizadas anteriormente. As aplicações para a plataforma J2ME vão desde jogos, aplicações que acessam banco de dados etc.

Utilizar Java na programação de pequenos aparelhos significa ganhar todas as vantagens que a tecnologia traz consigo:

- Dinamismo: novas aplicações podem ser baixadas da rede e instaladas no dispositivo a qualquer tempo;
- Segurança: verificação de classes, forte tipagem, garbage colection etc, garantem a proteção das informações carregadas pelo dispositivo. Dados de uma aplicação não são acessíveis por outras aplicações;
- Portabilidade: aplicações podem ser portadas entre dispositivos de diferentes fabricantes e de diferentes tipos;
- Orientação a Objetos: alto nível de abstração do código, modularização e reusabilidade.

5.2 - Maquinas Virtuais J2ME

Em poucas palavras, a maquina virtual Java é um mecanismo que permite executar código em Java em qualquer plataforma. Segundo a definição da SUN, a principal responsável pela criação da linguagem Java, a JVM pode ser entendida como "uma máquina imaginária implementada via software ou hardware que executa instruções vindas de bytecodes".

Uma Máquina Virtual Java é o fundamento para a Tecnologia Java, permitindo que aplicações escritas na linguagem de programação Java sejam portáveis através de ambientes de hardware e sistemas operacionais diferentes.

A máquina virtual faz a ponte entre a aplicação e a plataforma utilizada, convertendo o bytecode da aplicação em código de máquina apropriado para o hardware e sistema operacional utilizados. Além de gerenciar a execução dos bytecodes da aplicação, a máquina virtual lida com tarefas relacionadas à administração da memória do sistema, provendo
segurança contra código malicioso e administrando as threads dos programas.

A máquina virtual Java usada na versão standard (padrão) e na versão enterprise (corporativa) foi desenvolvida para ser utilizada em sistemas desktop e servidores respectivamente. Em J2ME deve-se utilizar uma máquina virtual Java que seja apropriada para os

dispositivos como telefones celulares, pagers e PDAs que possuem tamanho de memória reduzido e outra para
dispositivos com um poder um pouco maior de memória tais como: set-top boxes, sistemas de navegação de carros, pcs handhelds e outros [9].
A seguir uma ilustração da arquitetura J2ME:

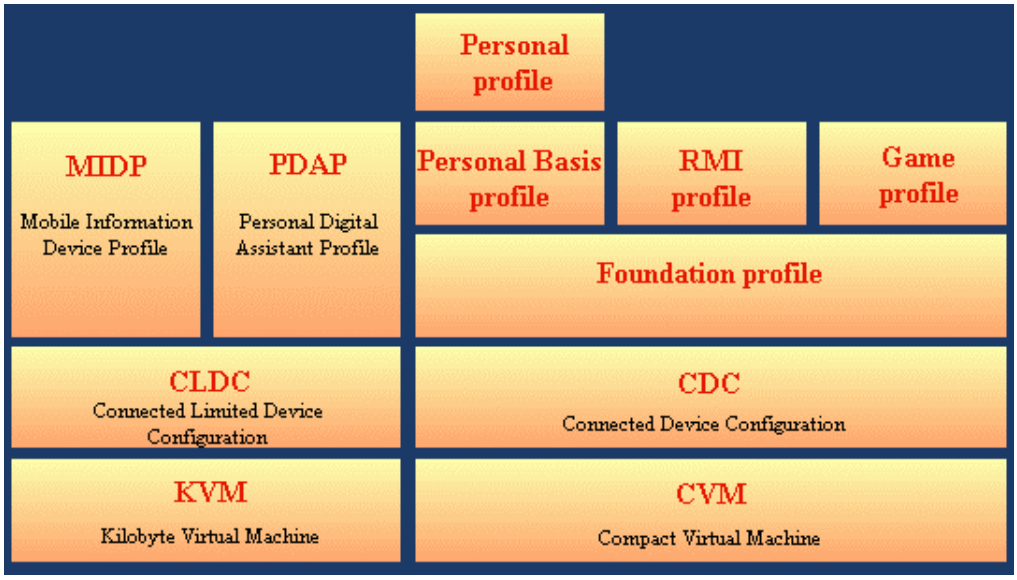

Figura 5.1: Arquitetura J2ME

Em virtude disso, duas máquinas virtuais foram projetadas. Uma para ser utilizada na configuração CLDC, Kilo Virtual Machine (KVM), e outra para a configuração CDC, a Compact Virtual Machine (CVM). Ambas as máquinas virtuais, serão apresentadas a seguir.

5.3 - A máquina virtual K (KVM)

A KVM ou Kilo Virtual Machine é a mais nova e otimizada máquina virtual Java para dispositivos com limites de restrições. Possuindo cerca de 40 a 80K de memória, tornou-se bastante apropriada para dispositivos como pagers, telefones celulares e PDAs [9].

A KVM pode executar em qualquer sistema que possua um processador de 16 ou 32 bits e um total de memória de 160 a 512K. Além disso, não suporta tipos de dados longos e de ponto flutuante. Seu projeto foi baseado em algumas importantes considerações, incluindo o

tamanho reduzido para conservar um melhor espaço em memória quanto possível (tanto em termos de
armazenamento quanto execução) e a capacidade de rodar em processadores de pequeno poder computacional [9].

A meta do projeto para a construção da KVM era o de criar uma máquina virtual compacta que mantivesse os aspectos mais importantes da linguagem de programação Java, e que conseguisse executar em um dispositivo com recursos limitados, com só algumas dezenas ou centenas de kilobytes de memória disponível (daí vem o K, de kilobytes) [9].

Mais especificamente, a KVM é projetada para ser:

- Pequena e com baixo requisito de memória;
- Enxuta e portável;
- Modular e customizável;
- Tão completa e rápida quanto possível.

A KVM é implementada na linguagem C, por isso pode ser portável para outras plataformas que contenham um compilador C.

5.4 - A máquina virtual C (CVM)

A máquina virtual C foi desenvolvida para adicionar maior funcionalidade aos dispositivos da segunda categoria dos quais a KVM não suporta. Praticamente esta máquina virtual engloba quase todas as características de uma máquina virtual Java convencional, só que de forma mais otimizada. Eis algumas características da CVM:

- Melhor desempenho em aplicações real time;
- Coleta automática de Lixo (garbage collection) otimizada;
- Mapeamento direto de threads Java para threads nativos;
- Execução de classes diretas da memória ROM;
- Portabilidade;
- Sincronização com um reduzido número de instruções;

5.5 – CLDC

A configuração CLDC consiste de uma máquina virtual, a KVM, e um conjunto de bibliotecas de classes para serem utilizados dentro de um perfil definido pela indústria, tal como o MIDP (sessão 5.6) [3].

A CLDC foi projetada pela Sun para ser uma configuração padrão, portável, com requisitos mínimos, para ser utilizada em dispositivos móveis, pequenos e com grande restrição de recursos, tais como pagers, telefones celulares, assistentes pessoais digitais e terminais de ponto de venda.

Os dispositivos que se enquadram nesta configuração apresentam como características:

- Processadores de 16 ou 32 bits;
- Requerem de 160Kb a 512Kb de memória total disponível para a plataforma Java;
- Baixos consumos de potência, freqüentemente são dispositivos operados por bateria;
- Conectividade com alguma espécie de rede, freqüentemente com uma conexão intermitente, sem fio e com largura de banda limitada (9600bps ou menos).

A especificação da CLDC não impõe nenhum requisito de hardware específico, a não ser o requisito de memória de 160Kb-512Kb. Mais especificamente:

- 128Kb de memória não volátil para a máquina virtual e bibliotecas CLDC;
- Pelo menos 32Kb de memória volátil para o ambiente de execução e
  objetos alocados.

A CLDC abrange as seguintes áreas:

- Características da máquina virtual e da linguagem;
- Entrada/saída;
- Acesso à rede;
- Segurança;

- Internacionalização;
- Bibliotecas de classes e APIs suportadas (java.util.*, java.lang.*) .

As seguintes áreas não são cobertas pela CLDC, devendo ser abrangidas por profiles montados no topo da CLDC:

- Interface com o usuário;
- Manipulação de eventos;
- Gerenciamento do ciclo de vida da aplicação (instalação, remoção, etc.).

5.6 – MIDP

Mobile Information Device Profile (Perfil de Dispositivo de Informações Móvel), é um perfil J2ME direcionado a pequenos dispositivos, ligado ao CLDC.

Acrescenta à ele outras APIs para interface com o usuário, regras de display, métodos de entrada de dados, armazenamento persistente, suporte à rede e temporização, conexão HTTP 1.0 usando o Generic Connection.

Um MIDlet é uma aplicação MIDP. Uma suíte de MIDlets é um agrupamento de MIDlets distribuído em um pacote JAR, que contém além dos próprios MIDlets, arquivos de recursos (imagens e propriedades) [8]. Vários fabricantes adicionam ao MIDP algumas outras APIs, tais como suporte a uma forma simplificada de AWT (Abstract Window Toolkit), controles de vibração, som e display.

Cada fabricante adiciona livremente o que considerar importante. Isso dificulta a portabilidade da aplicação entre os vários modelos de aparelhos e fabricantes, a menos que se use sempre o conjunto mínimo especificado no MIDP 1.0, para garantir o "Write Once, Run Anywhere" (WORA).

A maioria das implementações atuais utilizam a MIDP 1.0. Algumas características:

- 128 Kb de memória não-volátil para a implementação MIDP
- 32 Kb de memória volátil para o heap de runtime
- 8 Kb de memória não-volátil para persistência de dados

- Tela com pelo menos 96 x 54 pixels
- Entrada de dados, seja por teclado, teclas de telefone ou touch screen
- Conexões de rede (two-way), possivelmente intermitente

5.6.1 – MIDP 2.0

O padrão MIDP 2.0 New Generation oferece novos recursos como aumento de performance, melhorias nas APIs para interfaces gráficas , maior segurança e APIs para jogos (funcionalidades 2D) e conectividade, tornando possível o desenvolvimento e a utilização de uma maior variedade de aplicações.

A segurança neste padrão está mais confiável. Através da assinatura criptográfica a Suite de MIDlets a ser baixada para o telefone fica mais confiável, validando a origem dos MIDlets, podendo ser assinados e tratados semelhante à J2SE.

O conceito de sandbox também foi melhorado. As aplicações confiáveis (trusted) tem acesso a recursos específicos, enquanto as não-confiáveis (untrusted) precisam de confirmação do usuário para acessar os protocolos HTTP e HTTPS.

Quanto à rede, além do HTTP há suporte a cinco protocolos : HTTPS, Comunicação Serial, Sockets, Server Socketse Datagramas, através do Generic Connection Framework. Também suporta a tecnologia Push: MIDlets podem registrar um dispositivo para receber eventos (conexões de rede, mensagens enviadas), mesmo quando o dispositivo não esteja em modo de execução de aplicações Java.

É requerida a implementação do OTA Recommended Practice (Over The Air), padronizando o download e a instalação de MIDlets através do browser embutido. A padronização inclui notificações para o servidor quando as aplicações são instaladas, gerenciamento de atualizações de MIDlets e o suporte a sessões através de URL-Rewriting.

Os conceitos de Perfil, Configurações e Máquinas Virtuais formam as camadas da Arquitetura J2ME.

5.7 – J2ME Profiles vs. Bluetooth Profiles

É bastante comum, que após a introdução dos conceitos de profiles tanto de J2ME como de Bluetooth, um desenvolvedor que esteja iniciando um desenvolvimento tenha esses conceitos misturados em sua cabeça. J2ME profiles são um conjunto de classes em Java que servirão para extender as configurações do próprio J2ME.

Por outro lado, Bluetooth profiles são independentes de plataforma e de linguagem de programação e servem apenas para definir um conjunto de funcionalidades que os dispositivos poderão executar. Ou seja, um dispositivo como um telefone celular pode ter o Object Push Profile implementado em Java e um PDA ter esse mesmo profile implementado em Assenbly.

5.8 – MIDlets

São as aplicações do perfil MIDP. São empacotadas, colocadas dentro de um container como os Applets, Servlets, EJBlets, como arquivo JAR. Possuem arquivo descritor da aplicação - JAD que acompanha o arquivo JAR.

O arquivo descritor contém informações pertinentes à Suite, tais como: nome do vendedor, versão, tamanho, nome de cada MIDlet, ícones, imagens.

Uma MIDlet Suite é chamada quando tem mais de um MIDlet dentro da aplicação. O MIDlet Suite é uma camada de segurança, uma Suite não acessa outra, mas MIDlets empacotadas juntas podem acessar informações uma da outra.

Suporta ciclo de vida específico, sujeito ao comportamento da JVM. O Application Manager controla a instalação e execução.

O Banco de dados não é SQL, é orientado a registros (Point Base) camada do JDBC. O acesso ao RMS (Record Management System)está restrito ao MIDlet que o criou.

O processo de programação é um pouco mais complexo: Editar o código, compilar, pré-verificar, emular[14], enviar para o dispositivo.

[14] Caso seja necessário ou o desenvolvedor queira realizar.

5.8.1 – Ciclo de vida do MIDlet

A classe MIDlet oferece três métodos abstratos que são usados pela aplicação. Eles são chamados à partir do gerenciador de aplicações do dispositivo, e são usados para se comunicar com os aplicativos que estão rodando. De acordo com Martins o método "startApp" é chamado imediatamente depois do construtor e cada vez que um aplicativo é ativado, estiver visível. Isto quer dizer que este método não é chamado somente quando o aplicativo é iniciado, uma aplicação pode fazer a transição entre os estados de ativa e inativa muitas vezes durante o tempo em que está sendo rodada, e portanto deve-se evitar colocar código de inicialização que só pode ser rodado uma única vez [1].

Ainda de acordo com o mesmo autor, o método "destroyApp" é chamado pelo gerenciador de aplicativos para indicar que uma aplicação está prestes à ser terminada. Não como o método "startApp", este método será chamado uma única vez durante o tempo de execução de uma aplicativo, portanto é recomendado o uso de "código de limpeza" aqui. [1].

Isto porque a MIDP não possui a capacidade de finalizar seus objetos, então eles devem ser terminados, liberados no "destroyApp".

O método abstrato, "pauseApp", é utilizado para notificar o aplicativo que está rodando está para ser pausado porque o usuário acabou de começar a rodar outro aplicativo ou está utilizando uma função do dispositivo que prevenirá seu aplicativo de continuar rodando. A maioria dos dispositivos móveis não possuem poder de processamento para estarem realmente "multitasking" (executando várias funções ao mesmo tempo). Este método, segundo Martins, provavelmente será chamado com uma certa freqüência e deve-se manter em mente que os recursos sendo utilizados pelo aplicativo devem ser liberados neste momento. Quando a aplicação volta à rodar, o método "startApp" será chamado pelo gerenciador de aplicativos[1].

6 - Desenvolvimento de Aplicações Móveis usando J2ME e Bluetooth

Neste capítulo, mostraremos aspectos importantes a serem levados em consideração durante o desenvolvimento de aplicações móveis que se comuniquem via Bluetooth. Também iremos expor algumas técnicas e práticas de boa programação que irão aperfeiçoar os códigos produzidos e dar exemplos de quando esse casamento entre J2ME e Bluetooth não deve ser utilizado para o desenvolvimento de uma aplicação wireless móvel.

6.1 – Restrições, Técnicas e diretrizes de desenvolvimento.

Desenvolver software para as plataformas de servidores e *desktops* é uma tarefa diferente quando se for comparada ao desenvolvimento de aplicações para dispositivos móveis. Sistemas para servidores e *dektops* utilizam plataformas que possuem grande quantidade de memória e alto poder de processamento. Quando se desenvolvem aplicações para dispositivos móveis, novos obstáculos surgem devido aos recursos limitados disponíveis nos dispositivos, como tamanho da tela reduzida, pouca memória disponível, e baixo poder de processamento. Além disso, outros obstáculos surgem em virtude dos ambientes que os dispositivos operam; mobilidade e redes sem fio que tipicamente oferecem largura de banda mais baixa e menos confiabilidade no tráfego em comparação com as redes com fio [11].

Essas limitações exigem que a construção de aplicações sem fio seja bem planejada em relação à interface com o usuário, que deve ser simples e intuitiva. Além disso deve-se administrar a utilização da memória de forma econômica e eficiente, diminuir o tempo de interação com o sistema, para que não se utilize em demasia a bateria do dispositivo e avaliar as condições ambientais em que os dispositivos móveis operam, devido à baixa largura de
banda e segurança na transmissão dos dados.

6.1.1 - Restrições em Aplicações Sem Fio

No desenvolvimento desse tipo de aplicações, muitos fatores são de extrema importância para o sucesso do desenvolvimento do software. Dentre esses fatores, a escolha certa das tecnologias a serem utilizadas tem um percentual bastante crucial. Sabemos que por se utilizar de uma maquina virtual e por ser interpretada e compilada, Java não é a melhor escolha em aplicações que requerem muito desempenho ou exijam tempo de resposta curto, ainda mais quando se trata de dispositivos moveis quando outros fatores limitantes de performance que serão explorados mais adiante (**Sessões 6.1.2 e 6.1.3**) que já restringem as a certos patamares essas aplicações.

As aplicações sem fio devem trabalhar dentro das restrições dos dispositivos, tais como:

- Pouca memória: os dispositivos como telefones celulares e *pagers* possuem memórias limitadas, obrigando a considerar o gerenciamento de memória um fator primordial.
- Baixo poder de processamento: os dispositivos de sem fio também possuem uma potência de processamento limitada (variando de 32Kbytes a 64Mbytes).
- Entrada de dados: as capacidades de entrada são limitadas. A maioria dos telefones celular dispõe de uma entrada com doze botões: sendo dez números e alguns símbolos especiais como (*) e (#)
- Tela: O vídeo pode ter uma dimensão pequena (96 *pixels* de largura por 54 *pixels* comprimento) e monocromático. A quantidade de informações que se pode escrever em uma tela de um telefone celular é bastante limitada.

É fácil perceber as limitações dos dispositivos, mas, além disso, não se pode deixar de mencionar em que condições esses dispositivos operam:

- As Redes *Wireless* possuem baixa largura de banda.

- As Redes *Wireless* estão sujeitas a mais erros do que a redes com fio.
- A grande mobilidade dos dispositivos pode gerar perda da conexão.

Para aplicações em telefones celulares que fazem uso de J2ME e Bluetooth em especifico, temos um numero ainda pequeno de aparelhos nos quais a versão do JDK tenham acesso a pilha de protocolos do Bluetooth. Alguns exemplos desses aparelhos são o Nokia 6600 e o Sony Ericson P900.

6.2 - Desafios para o desenvolvimento sem fio

Como mencionado anteriormente, desenvolver aplicativos para dispositivos sem fio é uma tarefa desafiadora. Esta seção apresentará alguns dos principais pontos da computação distribuída onde os desenvolvedores de aplicações sem fio devem estar atentos antes de iniciar a construção de uma nova aplicação.

6.2.1 - Transmissão de Erros

As mensagens enviadas sobre conexões sem fio estão sujeitas a interferência e à demora, isso pode alterar o conteúdo recebido pelo usuário do dispositivo de destino ou pelo servidor. É necessário assegurar que o aplicativo esteja preparado para resolver estes problemas. Erros de transmissão podem ocorrer em qualquer ponto de uma transmissão sem fio e em qualquer ponto quando se envia ou recebe uma mensagem. Eles podem acontecer depois que um pedido foi iniciado, no meio da transmissão, ou
depois que uma resposta foi enviada. Os protocolos de rede de sem fio são capazes de descobrir e corrigir alguns erros, mas é preciso que o programador faça estratégias para tratamentos de erros que são prováveis de acontecer [11].

6.2.2 – Latência

Latência é o tempo que uma mensagem leva até chegar ao seu destino. A Latência é principalmente afetada pela natureza de cada sistema que emite a mensagem, e pelo tempo de processamento necessário em cada nó de origem até o destino.

É importante lembrar que uma mensagem pode, numa aplicação que faz uso da tecnologia Bluetooth, ser entregue para um usuário muito tempo depois de ser enviada. Uma demora longa poderia ser devido a problemas de cobertura ou erros de transmissão ou o dispositivo do usuário pode estar desligado ou ter ficado sem a bateria. Alguns sistemas continuam tentando transmitir a mensagem até que seja entregue. Outros sistemas armazenam a mensagem que não pode ser enviada em determinado

momento, e quando o dispositivo for reconectado à rede, a mensagem é enviada. A grande preocupação é evitar que sejam enviadas informações desatualizadas[11].

6.2.3 – Segurança

Quaisquer informações transmitidas sobre redes sem fio estão sujeitas a interceptações. Algumas informações podem ser confidenciais, como números de cartão de crédito e outras informações pessoais. Para prover uma solução de segurança fim a fim, deve-se implementar a segurança nas duas partes, tanto no cliente quanto no servidor, e estar assegurado que os sistemas

intermediários, também sejam seguros. Uma solução para utilizar quando as informações manipuladas são altamente confidenciais seria a criptografia, onde o remetente codifica os dados antes de transmiti-los pela rede sem fio, e o receptor autorizado recebe

os dados codificados e os decifra usando uma tecla especificada.

As aplicações de Comércio Eletrônico utilizam o Protocolo de Transferência de Hipertexto Seguro (HTTPS), que é o HTTP sobre a Camada de *Sockets* Seguros1 (SSL). O SSL trabalha de forma eficiente com comércio eletrônico e existem expectativas que ele seja bastante

utilizado pelo mcommerce2 também. Para J2ME já existe uma versão do SSL para dispositivos móveis chamado kSSL(*Kilo Safe Sockets Layer)*[11].

6.3 - Diretrizes para o Desenvolvimento de Aplicações Móveis

Agora, mostraremos alguns aspectos do contexto da aplicação que servirão como diretrizes para serem levadas em consideração durante o desenvolvimento de aplicações moveis distribuídas.

6.3.1 – Ambiente

Fazer uma pesquisa sobre o ambiente em que a aplicação será utilizada, antes de começar o desenvolvimento da aplicação. Deve-se primeiro saber as necessidades dos usuários em potenciais, os requisitos impostos pelas redes e em que plataformas o aplicativo funcionará (telefones celulares, *palmtops*, etc).

6.3.2 – Tarefas da Aplicação

É preciso pensar cuidadosamente quanto à decisão de quais operações devem ser feitas no servidor e quais devem ser feitas no dispositivo móvel. MIDlets permitem localizar muitas da funcionalidades do aplicativo no dispositivo, podem recuperar dados do servidor, processá-los e exibi-los na tela do dispositivo localmente. Esta abordagem reduz a interação com a rede sem fio, conseqüentemente diminuindo o tráfego da rede.[11]

6.3.3 – Dados

Os dados podem ser representados de muitas formas, uns mais compactos do que outros. Deve-se considerar representações disponíveis e escolher aquelas que exigem menos bits para ser transmitidos[11]. Por exemplo, números normalmente são muito mais compactos se transmitidos na forma binária do que se transmitidos na forma de string.

6.3.4 – Latência da Mensagem

Em algumas aplicações, pode ser possível fazer outras tarefas enquanto uma mensagem está sendo processada[11]. Se a demora é considerável, é importante manter o usuário informado do progresso da transmissão.

6.3.5 – Interface

Manter a interface do aplicativo simples e intuitiva, de forma que o usuário raramente precise consultar o manual do usuário para fazer uma tarefa[11]. É importante reduzir a quantidade de informações exibidas no dispositivo; apresentar as seqüências das entradas do usuário com um número mínimo de acionamento de botão ou teclas; oferecer sempre que possível listas de seleção para o usuário.

6.4 - Técnicas para otimizar a performance em aplicações J2ME

Aplicações utilizadas em dispositivos móveis devem ser de rápida execução, pois quanto menor o tempo de execução, menos tempo de uso da bateria, acarretando uma maior satisfação para o usuário [19]. Nesta seção, são apresentadas algumas técnicas de programação para aumentar o desempenho de aplicações J2ME:

- Sempre que possível utilizar variáveis locais ao invés de variáveis de classes, assim o acesso aos atributos de objeto é mais rápido com o uso das variáveis locais[19];
- Tentar diminuir as chamadas de métodos, pois quando um método é chamado, a máquina virtual Java aloca um novo nodo da pilha de execução, isso faz com que a memória seja sobrecarregada[19];
- Minimizar a criação de objetos, pois quando um objeto é criado, na maioria das vezes ele será destruído, isso leva à uma diminuição da performance da aplicação. Para evitar a criação de muitos objetos devese utilizar objetos que possam ser "reciclados" ou mantidos na memória o máximo de tempo possível[19];

- Tentar não concatenar strings, a concatenação com o operador + leva a criação de um novo objeto e por conseqüência utiliza mais a memória e o processamento[19];
- Evitar a sincronização, se um método demora algumas frações de segundos para executar, deve-se colocar a sua chamada em uma *thread* em separado[19].

6.5 – JSR-82

A JSR-82 é o nome formal para a API Bluetooth Wireless Technology (JABWT). A especificação da JSR-82 define a arquitetura de uma aplicação J2ME com Bluetooth, bem como também define as classes e métodos que devem ser implementados por qualquer aplicação compatível com a JSR-82. A API JSR-82 é baseada nas configurações CLDC e opera na camada CLDC da arquitetura de implementação da plataforma J2ME, mas na pratica, ela é usada como uma extensão de funcionalidades da camada de profiles do J2ME. Como já dito no capítulo 6, o profile mais utilizado para desenvolvimento para celulares é o MIDP.

A API JSR-82 tornou-se um padrão de desenvolvimento rápido para aplicações Bluetooth que rodem em telefones celulares. Porem o objetivo dessa API é definir um conjunto de padrões que irão permitir um ambiente aberto de desenvolvimento para dispositivos com recursos limitados utilizando a tecnologia Bluetooth[13].

Numa aplicação Bluetooth, existem alguns componentes básicos que o desenvolvedor deverá ficar atento. São eles:

- Stack initialization
- Device Inquiry
- Device management
- Name Discovery
- Service discovery
- RFCOMM/L2CAP Communication

A primeira ação que deve ser tomada durante a aplicação é a inicialização da pilha Bluetooth. Em algumas aplicações essa inicialização é feita automaticamente, porém em outras aplicações é necessário que seja adicionado uma pequena quantidade de código para que a pilha seja iniciada. Por exemplo a porta por onde será realizada a comunicação ou a taxa de comunicação entre os dispositivos devem ser especificados. Abaixo um trecho de código retirado de [15] para a ilustração de como ela pode ser feita.

```
...
// set the port number
BCC.setPortNumber("COM1");
// set the baud rate
BCC.setBaudRate(50000);
// set the connectable mode
BCC.setConnectable(true);
// set the discovery mode to Limited Inquiry Access Code
BCC.setDiscoverable(DiscoveryAgent.LIAC);
```

6.5.1 – Device Inquiry

O passo seguinte numa aplicação que utilize conexão através de Bluetooth é a descoberta de serviços que estão dentro do alcance do dispositivo inicializado.

Uma aplicação MIDlet pode escolher por fazer o Device Inquiry tanto por General/Unlimited Inquiry Access Code (GIAC) como por Limited Inquiry Access Code (LIAC). Através do general inquiry que é uma forma de descobrir dispositivos pelo GIAC, ou seja, o dispositivo poderá ser descoberto a todo instante sem considerar o limite de tempo.

Uma outra alternativa é usar o LIAC, onde o dispositivo poderá ser descoberto apenas dentro de um determinado limite de tempo (por exemplo, um minuto). E após esse limite de tempo estourar, o dispositivo irá mudar suas características para que os serviços dos outros dispositivos não possam mais encontrar esse dispositivo.

Na classe *LocalDevice*, os métodos *getDiscoverable()* e *setDiscoverable()* são usados para pegar e modificar o modo em que o dispositivo se encontra.

A classe *DiscoveryAgent* provê os métodos necessários para a realização do device inquiry. O device inquiry inicia com o método *startInquiry()*.Os retornos desse método são feitas por um handler que implementa a interface *DiscoveryDevice*, sendo assim quando o método *deviceDiscovered()* é chamado, ou *inquiryComplete()* é executado os dispositivos serão listados em um *Vector* de elementos.

Poderá ser feita também uma filtragem dos dispositivos encontrados através dos Class of Device (COD) dos dispositivos. Isso é interessante, pois dependendo de sua aplicação, não será interessante que você mantenha uma conexão com determinados dispositivos. Por exemplo, digamos que esteja sendo desenvolvido um jogo para um determinado celular, sendo assim provavelmente não será interessante que se mantenha uma conexão com dispositivos como impressoras, car kits, ou fones de ouvido. Para a realização dessa filtragem, na interface *DiscoveryListener*, quando o metodo *deviceDiscovered()* for executado, ira passar um parâmetro chamado *DeviceClass* que poderá ser usado nessa filtragem de dispositivos.

Uma forma de realizar o device inquiry é que a aplicação tenha uma lista de dispositivos pré-selecionados. O método *retrieveDevices()* da classe *DiscoveryAgent* pode ser usado para requisitar a lista de dispositivos remotos. Contudo o método *retrieveDevices()* não provê o COD, portanto a filtragem dos dispositivos ficaria compromentida.

6.5.2 – Device Management

A API Java contém as classes *LocalDevice* e *RemoteDevice*, pelas quais podemos fazer o gerenciamento das funcionalidades dos dispositivos definidas pelo Generic Access Profile. O *LocalDevice* representa o dispositivo local, ou seja o dispositivo Bluetooth onde a aplicação esta sendo executada. Já o *RemoteDevice* representa cada um dos dispositivos encontrados em uma possível busca de dispositivos.

```
…
// retrieve the local Bluetooth device object
LocalDevice local = LocalDevice.getLocalDevice();
// retrieve the Bluetooth address of the local device
String address = local.getBluetoothAddress();
// retrieve the name of the local Bluetooth device
String name = local.getFriendlyName();
....
```

De maneira similar, podem-se obter as informaçoes sobre um dispositivo remoto.

```
RemoteDevice remote = RemoteDevice.getRemoteDevice(
                        javax.microedition.io.Connection c);
// retrieve the Bluetooth address of the remote device
String remoteAddress = remote.getBluetoothAddress();
// retrieve the name of the remote Bluetooth device
String remoteName = local.getFriendlyName(true);
...
```

A classe *RemoteDevice* tambem prove metodos para autenticação, autorização e encriptação de dados que serao transferidos entre o dispositivo local e o remoto

6.5.3 – Name Discovery

O método *getFriendlyName(boolean alwaysAsk)* que foi utilizado no exemplo da sessão anterior (Sessão 7.5.2) da classe *RemoteDevice* faz a descoberta de nomes de dispositivos remotos numa aplicação MIDlet. Esse método pode contactar os dispositivos caso o nome desse dispositivo não seja conhecido ou se o parâmetro alwaysAsk estiver com valor igual a “true”. É importante observer que dispositivos

diferentes podem retornar nome iguais, como por exemplo, “Nokia 6600”, ou “Meu Telefone”, etc.

6.5.3 – Service Discovery

O passo seguinte numa aplicação móvel com conexões via bluetooth, é a descoberta de serviços disponíveis de dispositivos dentro do alcance do dispositivo local, pelos quais eu estou interresado. Cada aplicação em particular tem um *UUID* que irá referenciar sua aplicação. O código a seguir retirado de [1] irá ilustrar esse exemplo.

```
// UUID for this service:
String uuidString = "50FDB90ADBFB49b3AA71D6BA308E45F3";
String params = …;
// set optional parameters as needed
// an RFCOMM (BTSPP) based service:
String url = "btspp://localhost:" + uuidString + params;
try {
      StreamConnectionNotifier connectionNotifier =
            (StreamConnectionNotifier) Connector.open(url);
      StreamConnection connection = (StreamConnection)
            connectionNotifier.acceptAndOpen();
} // send or receive messages to remote peer, etc.
```

Um peer remoto consegue descobrir os serviços executando o service discovery em um ou mais dispositivos encontrados usando o método *searchServices()* da classe *DiscoveryAgent*.

```
public int searchServices(int[] attrSet, UUID[] uuidSet,
                          RemoteDevice btDev,
                          DiscoveryListener discListener)
      throws BluetoothStateException
```

Os parâmetros desse método são: o conjunto de atributos indicando que os dispositivos locais e remotos dão suporte ao serviço, o conjunto de UUIDs que definem cada serviço, o dispositivo onde esses serviços foram encontrados e o evento de discovery listener responsável por disparar o handler para a chamada do dispositivo correto.

A aplicação irá executar a descoberta de serviços normalmente para cada dispositivo encontrado que seja interessante para a aplicação. A JSR-82 permite que sejam criadas múltiplas transações de service discovery, e o numero máximo é definido na propriedade "*bluetooth.sd.trans.max*". Caso a aplicação tente criar mais transações desse tipo do que esteja especificado nessa variável de propriedade, uma exceção será levantada, *BluetoothStateException*.

6.5.4 – Comunicação através do protocolo RFCOMM

O protocolo RFCOMM, provê conexões de confiança, bidirecionais e orientadas a stream. Na API JSR-82, a base desse tipo de conexões são as interfaces *StreamConnectionNotifier* e *StreamConnection*. Que vem do Generic Connection Framework definido pelo Connected Limited Device Configuration (CLDC). Um servidor RFCOMM é criado pela chamada *Connector.open()* que irá receber como parâmetro uma URL apropriada para a conexão. O formato dessa URL é definida pela JSR-82 e para que o protocolo RFCOMM seja utilizado, o prefixo será "btspp://". O resto da URL ira conter parâmetros como relação entre Máster/Slave, preferencias e restrições de comunicação como encriptação de dados, autenticação e autorização [13].

Um objeto *StreamConnectionNotifier* é retornado quando uma conexão é aberta com o servidor, e seu método *acceptAndOpen()* pode ser usado para aceitar conexões com clientes remotos. Os conhecidos objetos *InputStream* e *OutputStream* são usados para ser e escrever dados nessa conexão. Clientes RFCOMM serão criados de maneira similar, ou seja, utilizando o método *Connector.open()* utilizando uma URL para o cliente.

6.5.5 – Comunicação através do protocolo L2CAP

A JSR-82 define as interfaces *L2CAPConnection* e *L2CAPConnectionNotifier* para o envio e recebimento de pacotes através do protocolo L2CAP. Essas interfaces são derivadas do CLDC Generic Connection Framework interface Connection. Um servidor

L2CAP é criado pela chamada *Connector.open()* com uma apropriada URL assim como é feita em conexões RFCOMM. A diferença principal da String URL que é passada como parâmetro à chamada do método que abre a conexão com o servidor é o prefixo referente ao protocolo utilizado. Enquanto utilizamos o prefixo "btspp://" para chamadas utilizando o protocolo RFCOMM, será utilizado o prefixo "btl2cap://" para conexões através do protocolo L2CAP. Os outros parâmetros são utilizados de forma similar à conexão via RFCOMM, ou seja, são parâmetros de encriptação de dados, autorização e autenticação da conexão. Porém essa URL também poderá conter parâmetros relacionados à preferência da aplicação com relação ao tamanho máximo de unidade transmitida, Maximum Transmit Unit (MTU), já que esse protocolo tem poder de quebrar e montar pacotes de acordo com suas configurações.

Um objeto da classe *L2CAPConnectionNotifier* é retornado quando uma conexão com o servidor é criada. O método *acceptAndOpen()* pode ser usado para aceitar e abrir conexões com novos clientes. Os pacotes de dados serão enviados e recebidos através dos métodos da classe *L2CAPConnection*. Os clientes L2CAP serão criados de forma similar usando o método *Connector.open()* utilizando uma URL adequada para o cliente.Vale salientar que uma aplicação que utiliza conexões via protocolo L2CAP deverá prover seu próprio fluxo de controle caso seja necessário.

7 - Estudo de Caso

Para a exemplificação de alguns dos aspectos encontrados no decorrer da elaboração desse trabalho, foi criado uma aplicação chamada de BTChatALSL.

7.1 – Aplicação desenvolvida

A aplicação BTChatALSL corresponde à uma implementação de um Chat que envia, recebe e disponibiliza mensagens entre dois usuários.

Quando iniciada, a aplicação irá registrar o nome da pessoa que estará prestes a utilizar o serviço de Chat e logo após esse passo a aplicação estará apta a fazer uma busca de outros dispositivos que estejam presentes na área de alcance desse dispositivo. Quando a busca é completada, uma lista contendo os logins dos usuários que estão com essa aplicação sendo executada dentro do alcance do dispositivo é mostrada na tela do celular, dando poder então ao usuário para que ele possa iniciar um Chat com o usuário selecionado. A partir de então o usuário poderá fazer a troca de mensagens junto ao usuário remoto ate que um deles solicite o encerramento do Chat. Voltando então à tela de busca de dispositivos.

7.2 – O Processo de Desenvolvimento

O código fonte da aplicação BTChatALSL foi todo implementado para aplicações J2me. A ferramenta de edição de arquivos utilizada foi o programa **Gel** [21] que é uma IDE para desenvolvimento de aplicações Open Source. Para a criação e compilação de arquivos na maquina virtual Java para aplicações móveis, utilizamos o J2ME Wireless Toolkit em sua versão 2.2. Através de um componente desse Toolkit, o KtoolBar, podemos emular a aplicação desenvolvida e com isso verificar o funcionamento da aplicação, já que para que pudéssemos testar a aplicação em celulares, necessitaríamos de no mínimo dois celulares nos quais a versão do JDK instalado deveria ter suporte as chamadas à pilha Bluetooth, o que ocorrem em poucos celulares no

mercado ainda. Um exemplo de celular com essas características é o Nokia 6600.

7.3 – Estrutura do BTChatALSL

A aplicação BTChatALSL, foi dividida em cinco classes:

- View;
- Fachada;
- ControlaBluetooth;
- ChatBluetooth;
- BTChatALSL;

Uma observação bastante interessante sobre a implementação das classes Fachada e ChatBluetooth, a qual é importante ser mencionada, é que para qualquer chamada de método que possa envolver troca de dados entre o dispositivo local e o dispositivo remoto, se faz necessário a criação de um *Thread* para que essa troca de dados possa ser feita. A criação dessa *Thread*, é importante pois caso algum dispositivos remoto não possa receber o dado enviado ou o próprio dispositivo local não consiga enviar esse dado, a aplicação na ficará travada esperando que essa transação seja comletada.

7.3.1 – Classe View

A classe View, é responsável pela montagem e ações da tela da aplicação, ou seja, através dessa classe toda a interface gráfica é criada e exibida na tela do celular. A seguir um segmento de código da classe View:

```
...
public class View {

    Display display;

    Displayable anterior;

    Fachada escutador;

    private List areaList;
```

```
        /**
         * Comando para selecionar um item
         */
        static final Command selecionarComando = new
Command("${comando.selecionar}", Command.OK, 1);

        /**
         * Comando para sair do aplicativo
         */
        static final Command sairComando = new Command("${comando.sair}",
Command.EXIT, 2);

    ...

          public void adicionarMensagem(int autor, String mensagem) {
            logChat.append((autor == View.LOCAL ? apelidoLocal :
apelidoRemoto) + ": " + mensagem
                    + "\n");
            if (autor == View.LOCAL)
                entrada.setString("");
            display.setCurrentItem(entrada);

            textoChat.setText(logChat.toString());
            display.setCurrent(chat);
            display.setCurrentItem(entrada);
        }

        public void telaChat(String apelidoLocal, String apelidoRemoto,
boolean servidor) {
            this.anterior = display.getCurrent();
            this.apelidoLocal = apelidoLocal;
            this.apelidoRemoto = apelidoRemoto;
            chat = new Form("${tela.chat}");
            logChat = new StringBuffer();
            textoChat = new StringItem("", logChat.toString());
            chat.append(textoChat);

            if (servidor) {
                iniciaChat();
            } else {
                logChat.append("Aguardando...\n");
```

```
                chat.setTicker(new Ticker("Aguardando a resposta de " +
apelidoRemoto));
                display.setCurrent(chat);
            }
        }

        public void iniciaChat() {
            this.chat.setTicker(null);
            this.logChat.append("Iniciando Chat...\n");
            this.entrada = new TextField("Mensagem", "", 100,
TextField.ANY);
            chat.append(entrada);
            this.chat.addCommand(enviarComando);
            this.chat.addCommand(sairComando);
            this.chat.setCommandListener(escutador);
            this.display.setCurrent(this.chat);
        }

        public void fimChat(boolean chatRecusado) {
            alerta = new Alert("${tela.chat}", chatRecusado ?
"${chat.recusado}" : "${chat.ocupado}",
                    null, AlertType.INFO);
            alerta.setTimeout(Alert.FOREVER);
            display.setCurrent(alerta, anterior);

        }

        public void fechaChat() {
            alerta = new Alert("${tela.chat}", "${chat.encerrado} ",
null, AlertType.INFO);
            alerta.setTimeout(Alert.FOREVER);
            display.setCurrent(alerta, menu);
        }
    ...
```

7.3.2 – Classe Fachada

A classe Fachada foi criada traçar as ações do sistema e ao mesmo tempo provê uma separação das entidades que tem papeis diferentes da aplicação. A seguir um segmento de código da classe Fachada:

```
...
import java.io.IOException;
import javax.bluetooth.BluetoothStateException;
import javax.microedition.lcdui.ChoiceGroup;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Form;
import javax.microedition.lcdui.List;
import javax.microedition.lcdui.TextField;
import javax.microedition.midlet.MIDletStateChangeException;
import javax.microedition.rms.InvalidRecordIDException;
import javax.microedition.rms.RecordStoreException;
import javax.microedition.rms.RecordStoreFullException;
import javax.microedition.rms.RecordStoreNotFoundException;
import javax.microedition.rms.RecordStoreNotOpenException;


public class Fachada implements CommandListener {

    /**
     *
     */
    BTChatALSL bTChatALSL;

    /**
     * View do aplicativo, controlado por esta classe
     */
    View view;

    /**
     * Responsável pela transmissão de dados via bluetooth
     */
    ControlaBluetooth controlaBluetooth;

    /**
     * Responsável pela comunicação do chat
     */
    ChatBluetooth chatBluetooth;
...
```

```
    public void solicitaChat(List lista) {
        try {
            controlaBluetooth.finalizarBusca();
            int selecionado = lista.getSelectedIndex();
            final String apelidoLocal =
                  controlaBluetooth.achados[0].apelido;

            view.telaChat(apelidoLocal,
                        lista.getString(selecionado), false);

            final String endereco =
                  controlaBluetooth.getEnderecoRemoto(selecionado);
            if (endereco.length() == 0) {
                //TODO erro, vai pra area
            }
            new Thread() {
                public void run() {
                    try {
                        chatBluetooth.cliente(endereco,
                                                apelidoLocal);
                    } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                    }
                }
            }.start();
        } catch (RecordStoreException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

...
```

7.3.3 – Classe ControlaBluetooth

A classe ControlaBluetooth é responsável pela inicialização dos serviços Bluetooth, e pela busca dos outros dispositivos que estão com

a aplicação rodando. A seguir um segmento de código da classe ControlaBluetooth.

```
. . .
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.util.Vector;

import javax.bluetooth.BluetoothStateException;
import javax.bluetooth.DataElement;
import javax.bluetooth.DeviceClass;
import javax.bluetooth.DiscoveryAgent;
import javax.bluetooth.DiscoveryListener;
import javax.bluetooth.LocalDevice;
import javax.bluetooth.RemoteDevice;
import javax.bluetooth.ServiceRecord;
import javax.bluetooth.UUID;
import javax.microedition.io.Connector;
import javax.microedition.io.StreamConnection;
import javax.microedition.io.StreamConnectionNotifier;
import javax.microedition.rms.InvalidRecordIDException;
import javax.microedition.rms.RecordStoreException;
import javax.microedition.rms.RecordStoreNotOpenException;

public class PerfilBluetooth {

    private UUID uuid;

    private static final String UUID_STRING =
                        "00112233445566778899AABBCCDDEEFF";

    private DiscoveryAgent agente;

    View view;

    private Discoverer discoverer;

    private Vector encontradoRedeDTO;

    private Vector chatRecord;
...
```

```
public void iniciarBusca(PerfilDTO perfilDesejado) throws
                                          IOException {
        System.out.println("iniciando busca");
        encontradoRedeDTO = new Vector(4);
        chatRecord = new Vector(4);
        discoverer = new Discoverer(this);
        agente.startInquiry(DiscoveryAgent.GIAC, discoverer);
    }

public void buscaDispositivoCompletada(RemoteDevice[]
                              remoteDevice, String mensagem) {

      System.out.println("busca por dispositivos completada");
      //caso não encontre nenhum dispositivo termina a busca
        if (remoteDevice.length == 0) {
            view.telaInfo("${alerta.bluetooth.naoencontrou}",
                                               View.menu);
            return;
        }
        Discoverer discoverer2 = new Discoverer(this);
        //return service name attribute
        int[] attrSet = { 0x0100 };
        UUID[] uuidSetPerfil = new UUID[] { uuid };
        UUID[] uuidSetChat = new UUID[] { ChatBluetooth.uuid };
        for (int i = 0; i < remoteDevice.length; i++) {
            try {
                agente.searchServices(attrSet, uuidSetChat,
                              remoteDevice[i], discoverer2);
            } catch (BluetoothStateException e) {
                e.printStackTrace();
                continue;
            }
        }
    }
...
```

7.2.4 – ChatBluetooth

A classe, faz a conexão entre dois dispositivos bluetooth, e é responsável pela troca de mensagens entre esses dois dispositivos

quando o Chat está aberto. A seguir um segmento de código da classe ChatBluetooth:

```
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;


import javax.bluetooth.UUID;
import javax.microedition.io.Connector;
import javax.microedition.io.StreamConnection;
import javax.microedition.io.StreamConnectionNotifier;
import javax.microedition.rms.InvalidRecordIDException;
import javax.microedition.rms.RecordStoreException;
import javax.microedition.rms.RecordStoreNotOpenException;

public class ChatBluetooth/* implements Runnable */{

    public final static String UUID_STRING =
                            "00112233445566778899AABBCCDDEECC";

    public final static String SERVICO_CHAT_STRING = "ChatBTALSL";

    public static final UUID uuid = new UUID(UUID_STRING, false);

    StreamConnection conexao;

    DataInputStream input;

    DataOutputStream output;

    StreamConnectionNotifier notifier;

    private final static int LIVRE = 0;

    private final static int CONVERSANDO = 1;

    private final static int CONEXAONEGAGADA = 2;

    private final static int CONEXAOACEITA = 3;
```

```
    final static int CONEXAOENCERRADA = 4;

    private int flag;

    View view;

    Fachada fachada;

    String apelidoRemoto;

    ...
    public void servidor() {

    String url = "btspp://localhost:" + uuid.toString() + ";name=" +
                                    ChatBluetooth.SERVICO_CHAT_STRING;

    try {
         notifier = (StreamConnectionNotifier)Connector.open(url);

         while (true) {
             negociacaoServidor(notifier.acceptAndOpen());
         }
    } catch (IOException e) {
         e.printStackTrace();
    } catch (RecordStoreException e) {
         e.printStackTrace();
    }

}

private void negociacaoServidor(StreamConnection conexao) throws
                  IOException,RecordStoreNotOpenException,
                        InvalidRecordIDException, RecordStoreException {
        DataInputStream input = conexao.openDataInputStream();
        DataOutputStream output = conexao.openDataOutputStream();

        switch (flag) {
        case ChatBluetooth.CONVERSANDO:
            output.writeInt(ChatBluetooth.CONVERSANDO);
            output.flush();
```

```
            output.close();
            input.close();
            conexao.close();
            break;
        case ChatBluetooth.LIVRE:
            flag = ChatBluetooth.CONVERSANDO;
            this.output = output;
            this.input = input;
            this.conexao = conexao;
            output.writeInt(ChatBluetooth.LIVRE);
            output.flush();
            apelidoRemoto = input.readUTF();
            this.input = input;
            this.output = output;
            this.conexao = conexao;
            view.confirmarChat(apelidoRemoto);
            break;

        }
    }

    public void aceitaChat() throws IOException {
        output.writeInt(ChatBluetooth.CONEXAOACEITA);
        output.flush();
        new Thread() {
            public void run() {
                receber();
            }
        }.start();
        try {
            view.telaChat(
            controller.perfilDAO.carregarPerfilLocal().apelido,
            apelidoRemoto, true);
        } catch (RecordStoreException e) {
            e.printStackTrace();
        }
    }

    public void negarChat() {
        flag = ChatBluetooth.LIVRE;
        try {
```

```
            output.writeInt(ChatBluetooth.CONEXAONEGAGADA);
            output.flush();
            finalizar();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

    }

    /**
     * Negocia com o servidor a aceitação da conexão
     *
     * @param conexao
     * @param apelidoLocal
     * @throws IOException
     */
    private void negociacaoCliente(StreamConnection conexao,
                                                   String apelidoLocal)
                                                   throws IOException {
        input = conexao.openDataInputStream();
        output = conexao.openDataOutputStream();
        int flagServidor = input.readInt();
        switch (flagServidor) {
        case ChatBluetooth.CONVERSANDO:
            flag = ChatBluetooth.LIVRE;
            view.fimChat(false);
            break;
        case ChatBluetooth.LIVRE:
            output.writeUTF(apelidoLocal);
            output.flush();
            switch (input.readInt()) {
            case ChatBluetooth.CONEXAONEGAGADA:
                finalizar();
                view.fimChat(true);
                break;
            case ChatBluetooth.CONEXAOACEITA:
                view.iniciaChat();
                new Thread() {
                    public void run() {
                        receber();
```

```
                    }
                }.start();
                break;
            }
        }

    }
    ...
```

7.2.5 – Classe BTChatALSL

A classe BTChatALSL, é responsavel pelo inicio da aplicação. Essa classe é a classe que extends de MIDLet e por conseqüência implementa os métodos startApp(), pauseApp() e destroyApp(boolean arg0) que devem ser definidos por qualquer objeto que extends de um MIDLet. A seguir mostramos um segmento de código da classe BTChatALSL.

```
...

import javax.bluetooth.LocalDevice;
import javax.microedition.midlet.MIDlet;
import javax.microedition.midlet.MIDletStateChangeException;


public class BTChatALSL extends MIDlet {

    /**
     *
     */
    public BTChatALSL () {

    }

    /* (non-Javadoc)
     * @see javax.microedition.midlet.MIDlet#startApp()
     */
    protected void startApp() throws MIDletStateChangeException {
              System.out.println(LocalDevice.getProperty
```

```
                        ("bluetooth.master.switch"));
        Fachada fachada = new Fachada(this);

    }

    /* (non-Javadoc)
     * @see javax.microedition.midlet.MIDlet#pauseApp()
     */
    protected void pauseApp() {
        // TODO Auto-generated method stub

    }

    /* (non-Javadoc)
     * @see javax.microedition.midlet.MIDlet#destroyApp(boolean)
     */
    protected void destroyApp(boolean arg0) throws
                                    MIDletStateChangeException {
        notifyDestroyed();
    }

}
...
```

7.4 – Resultados da aplicação

Para visualização da aplicação, mostraremos telas capturadas durante a execução da mesma no emulador de dispositivos móveis (KtoolBar). As principais telas obtidas durante a emulação da aplicação serão mostradas a seguir:

7.4.1 – Tela Logar

Assim que a aplicação começa a ser executada, o usuário deverá informar o login que será utilizado durante a execução da aplicação.

Figura 7.1: Tela logar BTChatALSL

7.4.2 – Tela Busca Dispositivos

Após o usuário inserir seu login e entrar na aplicação, aparecerá uma tela onde o usuário ira realizar a procura de dispositivos.

Figura 7.2: Tela de busca de dispositivos

7.4.3 – Tela Dispositivos Encontrados

Assim que a busca por dispositivos é terminada, os dispositivos encontrados são listados na tela e é oferecida a opção de iniciar chat com um dos usuários encontrados.

Figura 7.3: Tela Usuários encontrados

7.4.4 – Tela aceitar Inicio Chat

Quando um dos usuários faz uma requisição para iniciar um Chat com outro usuário, é exibida uma mensagem perguntando ao usuário remoto se ele deseja iniciar um Chat com a pessoa que fez a requisição.

Figura 7.4: Tela aceitar inicio Chat

7.4.5 – Tela Conversação

Durante a realização do Chat, as mensagens enviadas e recebidas pelos usuários que estão participando do Chat, serão mostradas em ordem em que foram enviadas e com o respectivo login para indicar quem enviou a mensaem.

Figura 7.5: Realização do Chat

7.4.6 – Finalização de Chat

Após um dos usuários solicitar a finalização do chat, a conexão entre os dois usuário será desfeita, o que não ira impedir que eles voltem a refazer essa conexão iniciando um novo Chat.

Figura 7.6: Chat encerrado

8 – Conclusão e Trabalhos Futuros

No decorrer deste trabalho, tivemos a oportunidade de analisar de forma detalhada as tecnologias J2ME e Bluetooth. Essa análise se deu através da descrição e exemplificação dessas tecnologias com relação a um breve histórico, arquitetura, organização e interação e comparação com outras tecnologias similares. Por exemplo, vimos na comparação entre as tecnologias Wireless Bluetooth e Wi-fi, que elas atuam de forma bastante parecidas, inclusive na mesma freqüência de ondas de Rádio. Porém vimos também que elas foram criadas para propósitos diferentes por isso nenhuma das duas irá provavelmente substitui a outra (**Sessão 4.6.2**).

Além dessa descrição detalhada, vimos também como essas tecnologias se comportam diante do desenvolvimento de aplicações móveis distribuídas. Analisamos aspectos como latência, taxa de erros, e segurança em aplicações que usam J2ME e Bluetooth.

E também desenvolvemos uma aplicação de Chat móvel (BTChatALSL) baseada nessas duas tecnologias. Essa aplicação serviu para colocar-mos em prática alguns dos conceitos expostos por esse trabalho.

No entanto, ao decorrer do desenvolvimento desse trabalho, algumas dificuldades foram encontradas, são elas:

- Apesar da melhora de capacidade de processamento e armazenamento de dados dos dispositivos móveis, ainda esbarramos em barreiras como tamanho limite das aplicações J2ME;
- Entrada de dados provenientes de usuários em aplicações móveis é sacrificada, uma vez que em geral é feita através do teclado de um celular;
- Falta de exemplos de aplicações básicas que servissem como referência para desenvolvimento.
- Limitações do emulador utilizado (KtoolBar) que é parte do WTK 2.2 (Wireless ToolKit 2.2).

- Impossibilidade de validação da aplicação em dispositivos móveis reais. Uma vez que poucos modelos de celulares tem seus JDKs que dão suporte ao acesso as chamadas Bluetooth via J2ME.

Como esse trabalho tinha como um de seus objetivos servir de referência para novos desenvolvedores e novas aplicações que viessem a fazer uso das tecnologias J2ME e Bluetooth, ele poderá de maneira geral servir de bibliografia para futuros trabalhos nessa área de conhecimento, ou também poderá servir de documento referência para implementações de aplicações que irão fazer uso dos benefícios dessas duas tecnologias nas diversas áreas de conhecimento.

Como exemplo de implementação de aplicação móvel distribuída envolvendo as tecnologias J2ME e Bluetooth na área jornalística esportiva, poderia ser desenvolvido um software que seria executado em um celular ou PDA, com alta resolução de gravação de vídeos e áudios, e a transmissão seria realizada através desse software para a central responsável pela transmissão. Sendo assim, os repórteres não teriam que carregar aquela quantidade de fios para os campos e quadras de futebol a fim de gravar e transmitir uma entrevista os atletas, treinadores etc.

Referências Bibliográficas

[1] – Nokia Corporation - Introduction To Developing Networked MIDlets Using Bluetooth – Version 1.0; May 11, 2004.

[2] – Sony Ericsson - Developing Applications with the Java APIs for Bluetooth (JSR-82) – January 2004

[3] - Connected, Limited Device Configuration. Specification Version 1.0a
Java 2 Platform Micro Edition, 2000. Disponível em: <http://www.sun.java.com/>.

[4] – Dias, K. L; Sadok, D. F.H. Internet Móvel: Tecnologias, Aplicações e
QoS. XIX Simpósio Brasileiro de Redes de Computadores, 2001.

[5] – RFI Mobile Technologies AG – Bluetooth Technology: Manufacturer-independent information on Bluetooth™ Technology and its economic aspects, 2001.

[6] – Hopkins, B.; Antony, R. – Bluetooth for Java ISBN:1590590783 Edição 2003

[7] – Gupta, V. Dass, A. Chauhan, Y. Cracking the Code – Wireless Programming with J2ME™. Hungry Minds, Inc, 2002.

[8] – Paludo, L - UM ESTUDO SOBRE AS TECNOLOGIAS JAVA DE DESENVOLVIMENTO DE APLICAÇÕES MÓVEIS, Novembro, 2003

[9] – Java 2 Platform Micro Edition (J2ME) Technology for Creating Mobile

Devices. White Paper. Sun, 2000. Disponível em <http://www.java.sun.com/>

[10] – Dias, K. L.; Fontes, W. P. – Desenvolvimento de Aplicações Para Dispositivos Móveis utilizando a plataforma J2ME.

[11] – Mahmoud, Q. H. Wireless Software Design Techniques What every wireless software developer should know. May 2002. Disponível em: <http://developers.sun.com/techtopics/mobility/midp/articles/uidesign/index.html>

[12] – Maia. R. M. F. – Bluetooth – Promessas de uma nova Tecnologia. 2003

[13] - JSR 82: Java™ APIs for Bluetooth disponível em <http://www.jcp.org/en/jsr/detail?id=82> no dia 15 de agosto de 2005

[14] – Mahmoud, Q. H. **Wireless Application Programming with J2ME and Bluetooth.** Disponível através do endereço eletrônico: <http://developers.sun.com/techtopics/mobility/midp/articles/bluetooth1/index.html> em 01/05/2005.

[15] – HOPKINS, BRUCE. Getting Started with Java and Bluetooth. Disponível através do endereço eletrônico: < http://today.java.net/pub/a/today/2004/07/27/bluetooth.html> em 01/05/2005.

[16] – Bluetooth – disponível em <http://www.bluetooth.com/>

[17] – Java 2 Platform Micro Edition (J2ME) – disponível em <http://java.sun.com/j2me/index.jsp>

[18] – TANENBAUM, ANDREW. Distributed Systems – Principles and Paradigms. Prentice Hall, 3a. Edição, 2002.

[19] – Roussel, T. - What Every Indian Developer Should Know About Java Technology, January 29, 2004. Disponível em: <http://developers.sun.com/careers/articles/javaindia.html>

[20] – Pinheiro, J. M. S. – Comunicações Moveis no PC – 14 de março de 2004, disponível em <http://www.projetoderedes.com.br/artigos/artigo_comunicacoes_moveis.php>

[21] – Gel – IDE de desenvolvimento disponível em <http://www.bittracker.com.br/Projetos.asp?session=ftyZA4fQTZ96fSqmavh9TJXj >

[22] – Entrevista com David Levin - Revista InfoExame - Edição Setembro, 2003

Apêndice A – Profiles do Bluetooth

1 - Generic Access Profile

O Generic Access Profile (perfil genérico do acesso) é o perfil o mais comum de Bluetooth. Todos perfis restantes usam este perfil para o estabelecimento básico da conexão. Este é o java.lang.Object no reino do perfil de Bluetooth; cada perfil necessita usar a funcionalidade do Generic Access Profile.

2 - Service Discovery Application Profile

O Service Discovery Application Profile (perfil de descoberta de serviço da aplicação) é um perfil que interaje diretamente com a camada do protocolo da descoberta do serviço (SDP) na pilha protocolos Bluetooth. Este perfil é usado encontrar serviços em dispositivos de Bluetooth habilitados na área.

3 - Serial Port Profile

O Serial Port Profile (perfil de porta serial) é um perfil que interaje diretamente com a camada de RFCOMM na pilha de protocolos Bluetooth. Este perfil é usado criar uma porta serial virtual em seu dispositivo Bluetooth. Por exemplo, alguns jogos de Bluetooth vêm com um driver que permita que seu sistema operacioal se comunique sobre a porta serial virtual como se fosse uma porta serial real.

4 - Dial-Up Networking Profile

Se você já tiver usado um modem antes, então você deve estar familiar com o conceito de conexão dial-up. O Dial-Up Networking Profile (perfil de conexão do dial-Up) permite que você imite a funcionalidade de um modem. Justo como o perfil da porta serial, alguns jogos de Bluetooth vêm com um driver que permita que seu sistema operacional se comunique sobre o modem virtual como se fosse um modem real.

5 - FAX Profile

Usando o FAX Profile(perfil de FAX), um dispositivo Bluetooth pode emitir um fax via wireless a uma máquina de fax Bluetooth ou a um telefone wireless Bluetooth.

6 - Headset Profile

O Headset Profile (perfil de headset) é projetado primeiramente para conectar fone de ouvidos Bluetooth aos telefones via wireless.

7 - LAN Access Profile

Um dispositivo Bluetooth tal como um PC ou um laptop usará o LAN Access Profile (perfil de acesso a LAN) conectar a um ponto de acesso da rede conectado a uma LAN.

8 - Personal Area Networking Profile

O Personal Area Networking Profile (perfil de conexão a área pessoal) é muito similar bonito ao perfil de acesso a LAN, a não ser que tenha também a sustentação para que os dispositivos dêem forma a redes ad hoc entre si. O perfil de conexão a área pessoal exige que o BNEP na pilha de protocolos subjacente.

9 - Cordless Telephony Profile

O Cordless Telephony Profile (perfil de telefone sem fio) permite que você use um telefone Bluetooth conectar a um telefone "landline" de um dispositivo Bluetooth que recebe as chamadas. Por exemplo, com este perfil, você continua a receber chamadas em seu telefone de casa, mas você tem a conveniência de responder a essas chamadas através de seu telefone móvel Bluetooth, sem usar os minutos do seu plano de telefone móvel.

10 - Intercom Profile

Se dois dispositivos Bluetooth-permitidos estiverem dentro do alcance, e suportarem o Intercom Profile (perfil intercom), eles podem funcionar como intercoms regulares.

11 - Generic Object Exchange Profile

O Generic Object Exchange Profile (perfil genérico da troca do objeto) é o perfil genérico que todos os perfis usam se quiserem empregar a funcionalidade do protocolo de OBEX na pilha de Bluetooth.

12 - Object Push Profile

O Object Push Profile (perfil do envio de objeto) fornece a funcionalidade para um dispositivo de enviar e pegar um objeto em outro dispositivo Bluetooth. Usando este perfil embora, você fica limitado a algumas classes de objetos como vCards.

13 - File Transfer Profile

O File Transfer Profile (perfil de transferência de arquivos) é um perfil mais robusto para transferência de objetos.

14 - Synchronization Profile

Você usa o Synchronization Profile (perfil de sincronização) para sincronizar dados entre dois dispositivos Bluetooth. As aplicações as mais comuns para este perfil geralmente sincronizam dados entre um PDA e um PC.

15 - Basic Printing Profile

O Basic Printing Profile (perfil de impressão básico) permite que um dispositivo Bluetooth envie um texto para que uma impressora Bluetooth imprima.

16 - Hard Copy Cable Replacement Profile

O Hard Copy Cable Replacement Profile é o que nós nos chamamos "o perfil de impressão avançado". Com este perfil, você pode imprimir todo o texto original em uma impressora Bluetooth. Se o dispositivo que você estver usando não tiver o driver para a impressora, ela ira providenciar esse driver para o seu dispositivo.

17 - Basic Imaging Profile

O Basic Imaging Profile (perfil básico de imagem) é pretendido para ser usado por dispositivos que criam imagens como câmeras para circuito interno, transferências de imagens e download

18 - Hands Free Profile

Os jogos hands-free Bluetooth nos automóveis usam o Hands Free Profile (perfil das mãos livres) com o objetivo de permitir que o motorista disque e receba chamadas de um telefone Bluetooth.

19 - Human Interface Device Profile

Como você pôde ter suposto, o Human Interface Device (perfil interface homem dispositivo) tem uma exigência que é o protocolo HID que deve existir na subjacência da pilha de protocolos Bluetooth. Este perfil define os cenários do caso para usar dispositivos de interação humana de Bluetooth como teclados e mouses. Um dos objetivos deste perfil é que um dispositivo Bluetooth conforme o perfil HID deve funcionar por três meses em três baterias alcalinas do AAA.