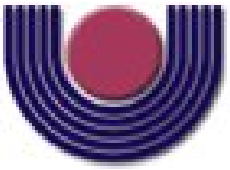


Unioeste - Universidade Estadual do Oeste do Paraná
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
Colegiado de Informática
Curso de Bacharelado em Informática

Estudo da Viabilidade de Aplicação da Tecnologia Bluetooth na Criação de Ambientes Computacionais Ubíquos

Anderson Luiz Menezes

CASCAVEL
2008

ANDERSON LUIZ MENEZES

ESTUDO DA VIABILIDADE DE APLICAÇÃO DA TECNOLOGIA BLUETOOTH NA CRIAÇÃO DE AMBIENTES COMPUTACIONAIS UBÍQUOS

Monografia apresentada como requisito parcial para obtenção do grau de Bacharel em Informática, do Centro de Ciências Exatas e Tecnológicas da Universidade Estadual do Oeste do Paraná - Campus de Cascavel

Orientador: Prof. Marcio Seiji Oyamada

CASCAVEL
2008

ANDERSON LUIZ MENEZES

ESTUDO DA VIABILIDADE DE APLICAÇÃO DA TECNOLOGIA BLUETOOTH NA CRIAÇÃO DE AMBIENTES COMPUTACIONAIS UBÍQUOS

Monografia apresentada como requisito parcial para obtenção do Título de Bacharel em Informática, pela Universidade Estadual do Oeste do Paraná, Campus de Cascavel, aprovada pela Comissão formada pelos professores:

Prof. Marcio Seiji Oyamada (Orientador)
Colegiado de Informática, UNIOESTE

Prof. Luiz Rodrigues
Colegiado de Informática, UNIOESTE

Prof. Reginaldo A Zara
Colegiado de Informática, UNIOESTE

Cascavel, 2 de dezembro de 2008

DEDICATÓRIA

Dedico este trabalho à minha família como um todo, sem exceções, avós, tios e primos. No entanto, gostaria de dedicá-lo especialmente à minha mãe, uma mulher que teve a oportunidade de concluir apenas o ensino fundamental, mas que traçou um objetivo de fornecer aos seus filhos mais do que pôde ter. Infelizmente ela não conseguiu chegar até esta data para ver um de seus filhos cursando o mestrado e o outro concluindo o ensino superior, mas tenho certeza que, ao ver nosso ingresso, já pôde sentir-se realizada e com sua missão cumprida, conseguindo enfim descansar em paz. À Vera Lúcia Ferreira Menezes, Alceu Sebastião Ferreira Menezes e Everson Ferreira Menezes, meus pais e irmão.

AGRADECIMENTOS

Assim como dediquei, não posso deixar de agradecer a minha família por todo apoio dado tanto nos anos de curso quanto em todos os outros momentos da minha vida.

Outras pessoas que foram muito importantes nessa trajetória são os amigos. Este agradecimento vai à todos aqueles que conquistei durante os anos de universidade e, sem dúvida, à aqueles que eu já possuía e consegui cultivar. Devo agradecê-los por permitir que eu passasse a fazer parte de suas vidas, por compartilhar os momentos de alegria e por me ajudar nos momentos de dificuldade (que não foram poucos).

Por fim deixo um agradecimento ao meu orientador Marcio Seiji Oyamada, sempre pronto e disposto para auxiliar no que fosse preciso.

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

ACK	Acknowledgment
ACL	Asynchronous Connection-Less
AODV	Ad-Hoc On-demand Distance Vector
API	Application Program Interface
BCC	Bluetooth Control Center
CDC	Connected Device Configuration
CID	Channel Identifier
CLDC	Connected, Limited Device Configuration
DoD	Department of Defense
DSN	Destination Sequence Number
EDR	Enhanced Data Rate
FCC	Federal Communications Commission
FHSS	Frequency-Hopping Spread-Spectrum
GAP	Generic Access Profile
HCI	Host Controller Interface
HTTP	Hipertext Transfer Protocol
IBM	International Business Machines Corporation
ICM	Industrial, Scientific, Medical
ID	Identification
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
J2EE	Java 2 Enterprise Edition
J2ME	Java 2 Micro Edition
J2SE	Java 2 Standard Edition
JSR	Java Specification Request
JVM	Java Virtual Machine
L2CAP	Logical Link Control and Adaptation Protocol
KB	Kilobyte
LAN	Local Area Network
LMP	Link Manager Protocol
LUNAR	Lightweight Underlay Network Ad-Hoc Routing
MAC	Media Access Control
MANET	Mobile Ad-Hoc Network
MB	Megabyte

Lista de Abreviaturas e Siglas

MIP	Museum Information Point
OBEX	Object Exchange
PAN	Personal Area Network
PC	Personal Computer
PDA	Personal Digital Assistant
PPP	Point-to-Point Protocol
RAM	Random Access Memory
RERR	Route Error
RFCOMM	Radio Frequency Communications Protocol
ROM	Read-Only Memory
RREP	Route Reply
RREQ	Route Request
SCO	Synchronous Connection-Oriented
SDP	Service Discovery Protocol
SIG	Special Interest Group
SMS	Short Message Service
TCP	Transmission Control Protocol
TCS	Telephony Control Signaling Protocol
TDD	Time Division Duplexing
TDMA	Time Division Multiple Access
URL	Uniform Resource Locator
WPAN	Wireless Personal Area Network

Sumário

Resumo

Atualmente a busca por mobilidade e transparência tem se intensificado de forma realmente considerável. Esta realidade associada a velocidade da evolução tecnológica dos dispositivos portáteis, como o surgimento do Bluetooth, nos possibilita pensar na criação de ambientes computacionais ubíquos voltados para tais dispositivos. Felizmente a tecnologia Bluetooth apresenta características que propiciam a formação de redes entre seus dispositivos, o que, com o auxílio de protocolos específicos e com a utilização do conceito de redes Ad-Hoc, permite a geração da comunicação necessária na constante cooperação requisitada pela ubiqüidade. Portanto, o presente trabalho busca utilizar a programação Java para dispositivos móveis no desenvolvimento dos ambientes ubíquos, os quais visam possibilitar a análise do suporte fornecido pela tecnologia Bluetooth a funcionalidades mais específicas.

Palavras-chave: Bluetooth, Computação Ubíqua, J2ME, Redes Ad-Hoc.

Capítulo 1

Introdução

Atualmente, podemos perceber que a busca pela mobilidade tem se intensificado de forma surpreendente. A quantidade de dispositivos móveis existentes no mercado, assim como a quantidade e qualidade dos serviços disponíveis, vem crescendo e se solidificando de maneira visivelmente promissora. Não é difícil nos depararmos com pessoas utilizando seus PDAs (*Personal Digital Assistant*) e *laptops* em centros comerciais, restaurantes, aeroportos, dentre outros diversos locais comuns ao nosso dia-a-dia. Mais popular ainda é o uso dos telefones celulares. Há algum tempo esses aparelhos deixaram de ser simples telefones, passando a possuir características que despertam a curiosidade tanto das pessoas continuamente ligadas aos avanços tecnológicos quanto daquelas mais leigas que, mesmo assim, não abrem mão de sua utilização. Certamente devido a facilidade de acesso, os celulares vem ganhando, a cada dia mais, um número maior de usuários, acarretando no crescimento do interesse pelo estudo e desenvolvimento voltado para tais aparelhos.

Mediante este cenário, o foco do presente trabalho está voltado para os telefones celulares. Como a grande maioria dos aparelhos mais atuais dispõem da tecnologia Bluetooth, este trabalho visa o estudo da utilização desta tecnologia na criação de ambientes computacionais ubíquos. Mesmo após grandes avanços, o Bluetooth ainda pode ser considerado subutilizado, uma vez que possui características e recursos muito relevantes no que diz respeito a criação de redes Ad-Hoc,"indispensáveis" devido sua propriedade de total descentralização. Nas redes Ad-Hoc, todo controle é realizado pelos próprios hospedeiros, permitindo maior mobilidade dos dispositivos, já que estes, por sua vez, não mais dependem de estações base, por onde, nas topologias mais tradicionais, passa todo fluxo de comunicação existente.

Assim como o Bluetooth, atualmente, grande parte dos aparelhos de telefonia celular suporta

a instalação de aplicativos Java, facilitando o desenvolvimento de aplicações para celulares.

Alguns trabalhos já realizados nesta área apresentam apenas propostas de arquiteturas ou testes e resultados provenientes de ambientes simulados (como pode ser encontrado em [31], onde uma arquitetura para jogos multiusuários é proposta e [33], um ambiente virtual para testes de eficiência do Bluetooth), onde boa parte das complicações podem ser ignoradas, como as limitações dos dispositivos e da própria tecnologia Bluetooth. Portanto, o objetivo deste trabalho é estudar a viabilidade da utilização da tecnologia Bluetooth, baseando-se num aprofundamento de sua pilha de protocolos, nos conceitos da computação ubíqua, das redes Ad-Hoc e na programação utilizando J2ME, apresentando, por fim, as observações verificadas através da criação da ubiqüidade em ambientes reais, não limitando-se apenas a simulação. É importante mencionar que questões de otimização, como economia no consumo de energia, ou segurança não estão como elementos primários, podendo ser postergadas para trabalhos futuros, ficando como principal ponto a criação do ambiente computacional ubíquo com a utilização dos dispositivos dotados da tecnologia Bluetooth.

No Capítulo 2 serão apresentados conceitos referentes à Computação Ubíqua, possibilitando um contato com suas principais características. O Capítulo 3 descreve a Tecnologia Bluetooth, apresentando sua origem e alguns pontos que impulsionaram sua escolha como objeto de estudo. A Plataforma de Desenvolvimento J2ME é descrita no Capítulo 4, onde são discutidas as principais características que permitem o desenvolvimento para dispositivos móveis, como os Perfis e Configurações. Nos Capítulos 5 e 6 serão apresentadas, respectivamente, uma introdução às Redes Ad-Hoc e o Protocolo de Roteamento AODV que também é um dos focos deste trabalho, uma vez que trata-se de um robusto protocolo Ad-Hoc. O Capítulo 7 iniciará a discussão sobre os estudos de caso abordados no trabalho, seguido pelos testes e considerações. O Capítulo 8 apresenta a conclusão do trabalho.

Capítulo 2

Computação Ubíqua

A Computação Ubíqua, definida por Mark Weiser [38], propõe a existência de uma nova ideologia, onde a computação estaria presente em todo lugar e a qualquer momento. Segundo Weiser: "*As tecnologias mais brilhantes são aquelas que desaparecem. Elas se misturam com as estruturas presentes no nosso dia-a-dia até que não se possa perceber sua presença*" [38], ou seja, a idéia é que sejam desenvolvidos ambientes nos quais a tecnologia estaria totalmente transparente às necessidades diárias dos usuários. Nesses ambientes, os computadores poderiam estar presentes nos mais simples objetos, como canetas, xícaras, interruptores de luz, maçanetas de portas, dentre inúmeros outros que se possa imaginar. Um conceito fortemente defendido por Weiser é o da invisibilidade. Em outra de suas publicações [39] ele diz: "*Uma boa ferramenta é uma ferramenta invisível. Ser invisível quer dizer que a ferramenta não irá interferir na sua consciência; você foca na tarefa, não na ferramenta*".

Se pararmos para analisar, a ubiqüidade já é uma realidade em nossas vidas. Atualmente a informação deixou de estar presente apenas nos livros e pode ser encontrada em qualquer lugar. Ao caminhar pelas ruas de uma grande cidade, nos deparamos facilmente com quantidades astronômicas de informações, no entanto, acabamos assimilando-as sem, nem ao menos, prestar atenção na forma em que ela está disponível. Ela está presente nas vitrines das lojas, nas placas de trânsito, nos *outdoors*, enfim, a informação já pode ser adquirida de forma totalmente transparente.

Estender essa ubiqüidade para computação não é algo tão simples, ao menos não da forma como Weiser propõe. Para alcançar o que podemos chamar de "ubiqüidade ideal" na computação, os dispositivos teriam que sofrer mudanças significativas, principalmente com relação aos seus tamanhos e sua dinamicidade. Não basta possuir a mobilidade de um dispositivo portá-

til como o *laptop*, por exemplo, pois o usuário continuaria dependente apenas daquela "caixa" para desenvolver seus trabalhos [38]. A computação ubíqua vai muito além. Seus dispositivos precisam ser capazes de reconhecer o ambiente onde se encontram e adaptar seus serviços de acordo com o contexto deste ambiente; eles precisam ser capazes de se auto-configurar para continuar suprindo as necessidades do usuário independente de toda e qualquer transição que possa ocorrer.

2.1 Características

O desenvolvimento dos sistemas ubíquos possui, pelo menos, três princípios básicos, os quais podem ser descritos como segue [3]:

- **Diversidade**: Ao contrário do que se pode imaginar, a diversidade nos propõe uma nova visão, a dos computadores de propósito específico. Este termo vem, então, da diversidade de dispositivos, não de aplicações. Ela nos diz que, diferente do que normalmente vemos atualmente, os dispositivos devem deixar de atuar por propósitos gerais, buscando exercer funcionalidades específicas. Um *palmtop*, por exemplo, pode ser melhor aplicado para que sejam efetuadas anotações rápidas do que na navegação web, deixando tal funcionalidade a cargo de outro dispositivo mais apropriado;

- **Conectividade**: Como temos agora diversos dispositivos exercendo, cada um, uma tarefa em especial, torna-se necessário uma expansão da conectividade, possibilitando a geração da cooperação e, além disso, como a computação ubíqua traz o conceito de conectividade sem fronteiras, essa expansão deve permitir, também, que o usuário permaneça conectado mesmo estando em trânsito por diversas redes heterogêneas;

- **Descentralização**: Tudo isso nos remete, então, à descentralização. A partir do momento que temos um ambiente cooperativo e distribuído, não podemos permanecer "presos" a um determinado servidor ou roteador, por exemplo. É interessante que o fluxo de informações seja propagado de hospedeiro a hospedeiro, independente das características particulares dos dispositivos, podendo, sim, existir servidores. Porém, a comunicação com eles ocorre apenas quando são necessárias atualizações de informação. Essa descentralização vem acompanhada da questão de sincronização. É importante levarmos em

consideração que, uma vez que o usuário pode efetuar determinadas operações de vários dispositivos diferentes, a replicação das informações é extremamente necessária para que se mantenha a consistência dos dados, evitando que estes estejam atualizados em determinados dispositivos e em outros não.

Quando estamos falando da possibilidade de existência de ambientes computacionais totalmente ubíquos, imediatamente imaginamos suas vantagens. Pensar em constante cooperação transparente de dispositivos em prol do nosso "bem-estar" é, certamente, algo que desperta bastante interesse. Não obstante, como pode ocorrer com quaisquer outras tecnologias, alguns problemas podem ser apontados para discussão, sendo eles [1]:

- **Segurança**: Em meio ao requerido nível de conectividade, como prevenir tentativas de acesso mal intencionadas? A segurança é um ponto de grande relevância. Deve-se pensar, para isso, na implantação de sistemas como *firewalls* embarcados para que se tenha uma forma de proteção contra, por exemplo, roubo de informações;

- **Privacidade**: A conectividade põe em dúvida, também, a privacidade. Muitos dos dispositivos espalhados por todo(s) o(s) ambiente(s) deverão manter bases de dados com informações de seus usuários, informações estas que, muitas vezes, são de âmbito particular e não devem ser acessíveis a qualquer pessoa;

- **Expansibilidade**: Aqui, uma questão pode ser colocada: como fazer para permitir que configurações de *hardware* e *software* diferentes interajam entre si? Se formos analisar, de uma forma geral, a computação ubíqua propõe que todos os tipos de dispositivos possam se comunicar e interagir, independente de arquitetura ou função específica;

- **Complexidade**: Por sua vez, o problema da complexidade está voltado para a forma com a qual os dispositivos irão interagir com os usuários. Como tudo estará acontecendo automaticamente, é bem comum que o sistema torne-se confuso devido a sobrecarga de informação. Se o sistema todo não for desenvolvido de forma a conseguir selecionar quais das informações são ou não relevantes, todas poderão ser repassadas ao utilizador, mesmo que ele não tenha idéia alguma de como utilizar tais dados.

Mark Weiser, em 1995, já discutia a problemática da complexidade, propondo, portanto, o que foi denominado como ***Calm Technology*** (Tecnologia Calma) [40]. A tecnologia calma baseia-se principalmente em dois conceitos: centro e periférico. Um exemplo exposto por Weiser traduz com bastante precisão os fundamentos do que ele defende: quando você está dirigindo seu carro, normalmente fica prestando atenção na pista, no rádio, ou, talvez, na conversa que esta tendo com um passageiro, mas dificilmente dispensa atenção com o barulho do motor. Você só irá fazê-lo se sua atenção for requisitada pelo veículo quando este liberar um aviso sonoro. Aqui os conceitos de centro e periférico estão bem visíveis. O centro da atenção do motorista é a pista. Ele só irá dirigí-lo ao periférico quando algo realmente importante acontecer.

Muitas vezes, algumas pessoas costumam utilizar computação móvel e computação pervasiva como sinônimo para a ubíqua, mas, na realidade, essas formas de computação são conceitualmente diferentes [3]. Podemos dizer que a computação ubíqua une as praticidades encontradas na móvel com a transparência e adaptabilidade da pervasiva. Observando a Figura 2.1 [3] podemos ter uma visão melhor dessa relação. A computação ubíqua localiza-se na intersecção dessas tecnologias, reunindo características relevantes de cada uma delas, buscando permitir que o computador deixe de ser visto apenas uma máquina estática sobre uma mesa.

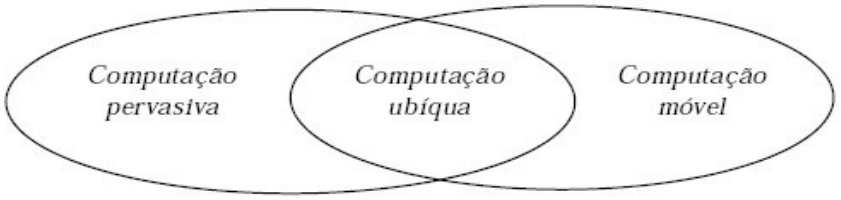


Figura 2.1: Relação entre Computação Ubíqua, Móvel e Pervasiva

Abaixo, temos os conceitos referentes a computação pervasiva e móvel, para que possamos distinguir com maior clareza suas peculiaridades.

2.1.1 Computação Móvel

A computação móvel pode ser vista como um novo paradigma, uma nova abordagem dos sistemas computacionais. Segundo Mateus e Loureiro [22], ela "*surge como uma quarta revolução da computação, antecedida pelos grandes centros de processamento de dados na década*

de sessenta, o surgimento dos terminais nos anos setenta e as redes de computadores na década de oitenta".

De uma forma geral, a idéia é fornecer recursos computacionais comparáveis as estações de trabalho convencionais, no entanto, com suporte a mobilidade, ou seja, prover processamento e troca de informações via rede com a utilização de dispositivos portáveis [8]. Podendo ser considerada uma área da comunicação sem fio, seus dispositivos são caracterizados, principalmente, pelo tamanho reduzido, uso de baterias e, obviamente, presença de uma tecnologia de acesso a redes sem fio, possibilitando, assim, que seus usuários os utilizem em qualquer lugar e a qualquer momento.

Um cenário capaz de exemplificar a utilização da computação móvel é o dia-a-dia de um representante de vendas. Este deve possuir acesso remoto a base de dados de sua empresa independente de sua localização, podendo, inclusive, efetuar operações sobre ela, como o acréscimo de um novo pedido ou uma consulta por determinado produto. Seu trabalho tornaria-se inviável ou, no mínimo, extremamente custoso se fosse necessário, a cada nova visita, conectar cabos a rede elétrica e a rede de computadores.

Normalmente acompanhado de seu *palmtop*, o representante apenas o retira de sua pasta e seleciona os produtos solicitados para nova ordem de compra. Caso exista possibilidade de comunicação, as informações são enviadas diretamente ao servidor da empresa atualizando os dados, do contrário, as modificações da base são armazenadas localmente para que sejam transmitidas assim que a rede esteja disponível.

A comunicação sem fio já vem sendo utilizada a muito tempo nos sinais de rádio e televisão. No entanto, a associação com a portabilidade e mobilidade é mais recente. A tecnologia que pode ser considerada como ponto de partida foi lançada em 1979 no Japão, sendo sucedida pela instalação das redes de telefonia celular em Chicago e Baltimore no ano de 1983. Atualmente, os sistemas de comunicação de voz, móveis e particulares, representam o principal nicho de comunicação sem fio. Isso devido, principalmente, mais ao baixo custo e fácil adaptação a rede pública de telefonia do que as suas características tecnológicas [22].

2.1.2 Computação Pervasiva

A computação pervasiva traz uma visão completamente diferente dos sistemas computacionais. Atualmente estamos acostumados a ver os computadores simplesmente como máquinas capazes de executar programas em ambientes virtuais, as quais são acessadas somente no momento da necessidade e deixadas logo em seguida.

Esta nova visão propõe que os dispositivos possam atuar como um portal para o espaço das aplicações e não apenas como um repositório de softwares que o usuário deve gerenciar. As aplicações, por sua vez, tornam-se um meio por onde o usuário efetua suas tarefas e não um *software* escrito para explorar o poder computacional. Além disso, um ambiente computacional seria um espaço físico de informações aprimorado, não somente um ambiente virtual existente para armazenar e executar os softwares [35].

De uma forma geral, ela traz o conceito de uma computação transparente, inteligente e adaptável. Os computadores permanecem embutidos no ambiente. "*Nessa concepção, o computador tem a capacidade de obter informação do ambiente no qual está embarcado e utilizá-la para dinamicamente construir modelos computacionais*" [3], ou seja, existe a necessidade de prover transparência associada a possibilidade de efetuar reconfigurações mediante a modificações no ambiente ou nas necessidades dos usuários.

A busca pela pervasividade computacional impõe uma série de mudanças não apenas na nossa forma de pensar, mas também no desenvolvimento das aplicações e dispositivos. Segundo Saha e Mukherjee [35], a computação pervasiva se inclui em muitas das pesquisas relacionadas a computação móvel, porém, gerando novas questões para serem tratadas. As principais características e desafios da computação pervasiva são [35]:

- **Escalabilidade**: representa a capacidade de manipular porções crescentes de trabalho de forma uniforme ou estar preparado para o seu crescimento. É bem provável que, com os ambientes pervasivos, ocorra um aumento do número de dispositivos, usuários, aplicações e interações como nunca antes visto. A medida que a quantidade de ambientes aumentar, o aumento do número de dispositivos conectados a esses ambientes e das interações entre homem e máquina é praticamente inevitável;

- **Heterogeneidade**: assumindo que implementações uniformes de ambientes inteligentes

ainda não estão disponíveis, é necessário buscar meios para não deixar que essa heterogeneidade seja percebida pelos usuários. Por exemplo, um ambiente inteligente desenvolvido para um grande laboratório certamente será muito diferente de um desenvolvido para um supermercado. Dessa forma, a utilização de *middlewares* torna-se inevitável para que os dispositivos possam migrar por esses ambientes sem que as diferenças sejam sentidas pelo usuário;

- **Integração**: apesar de já existirem inúmeros esforços para o desenvolvimento de componentes para ambientes pervasivos, a questão de integração entre eles ainda é um objeto de pesquisa. Por exemplo, os dispositivos necessitam de respostas rápidas dos servidores para conseguir suprir as necessidades dos usuários, no entanto, estes servidores certamente possuirão milhares de dispositivos concorrentes para gerenciar. A integração desses servidores buscando cooperação é algo inevitável para que se possa alcançar a disponibilização de serviços de forma eficiente. Observemos, também, o problema de heterogeneidade encontrado aqui;

- **Invisibilidade**: a invisibilidade pode ser vista como um requisito fundamental dos sistemas que não necessitam de muita intervenção humana. Tais intervenções podem ocorrer e são de extrema importância no processo de aprendizado do ambiente, no entanto, estes devem possuir estratégias de auto-ajuste capazes de reduzir ainda mais a necessidade da utilização da "mão humana". Sendo capaz de efetuar auto-ajuste, ou seja, reduzindo a necessidade de intervenção, o sistema pode passar despercebido pelos usuários, alcançando um nível de invisibilidade (transparência) tamanha que sua existência realmente deixa de interferir no dia-a-dia dos usuários. Voltando ao comentário de Weiser, você finalmente poderia deixar de se preocupar com a ferramenta e passaria a focar-se unicamente na tarefa;

- **Percepção**: a questão da percepção está diretamente ligada com a capacidade de reconhecimento do contexto atual do ambiente ao qual o sistema está embarcado. Atualmente as aplicações e dispositivos não possuem tal capacidade. No entanto, a computação pervasiva faz disso um pré-requisito, tornando o desenvolvimento dos ambientes mais complexo que, por exemplo, a localização de um dispositivo móvel. Essa é vista como uma

tarefa reativa, já que irá responder a acontecimentos. A computação pervasiva, por sua vez, é proativa, ou seja, deve ser capaz de tomar decisões antes que determinados fatos possam ocorrer, gerando a complexidade anteriormente citada;

- **Inteligência**: a inteligência de um sistema está ligada ao poder de gerenciar as mudanças de contexto ocorridas. Como tais sistemas, agora, são capazes de perceber alterações no ambiente, eles devem estar capacitados para utilizar essas percepções de forma eficiente.

2.2 Exemplos de Ambientes Ubíquos

O objetivo desta sessão é apresentar alguns trabalhos relacionados ao desenvolvimento de ambientes ubíquos. Abaixo temos uma breve descrição de três projetos desenvolvidos na área, onde é possível verificar algumas aplicações que serviram como base para o presente trabalho.

2.2.1 Projeto UbiqMuseum

No projeto UbiqMuseum [5] foi desenvolvido um ambiente heterogêneo, utilizando rede Ethernet, Wireless IEEE 802.11b e Bluetooth (promovendo o acesso aos dispositivos móveis). A idéia do projeto é a criação de um ambiente capaz de fornecer informações sobre as obras disponíveis no museu diretamente aos dispositivos móveis dos visitantes.

A arquitetura proposta (vide Figura 2.2) é formada por um servidor que detém uma base de dados com informações das obras, pelos MIPs (*Museum Information Points* – Pontos de Informação do Museu) e pelos dispositivos móveis dos visitantes. As informações são transmitidas do servidor até os MIPs através da infra-estrutura de rede citada anteriormente e estes, por sua vez, retransmitem tais informações por Bluetooth aos usuários.

Conforme o cliente passeia em meio as obras, seu dispositivo efetua buscas constantes por novos MIPs; quando um é encontrado, inicia-se então a busca por novas informações; caso esta exista, o cliente é informado e questionado sobre o interesse em visualizá-la.

Diferente dos muitos outros existentes, o projeto UbiqMuseum possui uma implementação experimental, não limitando-se apenas a simulação.

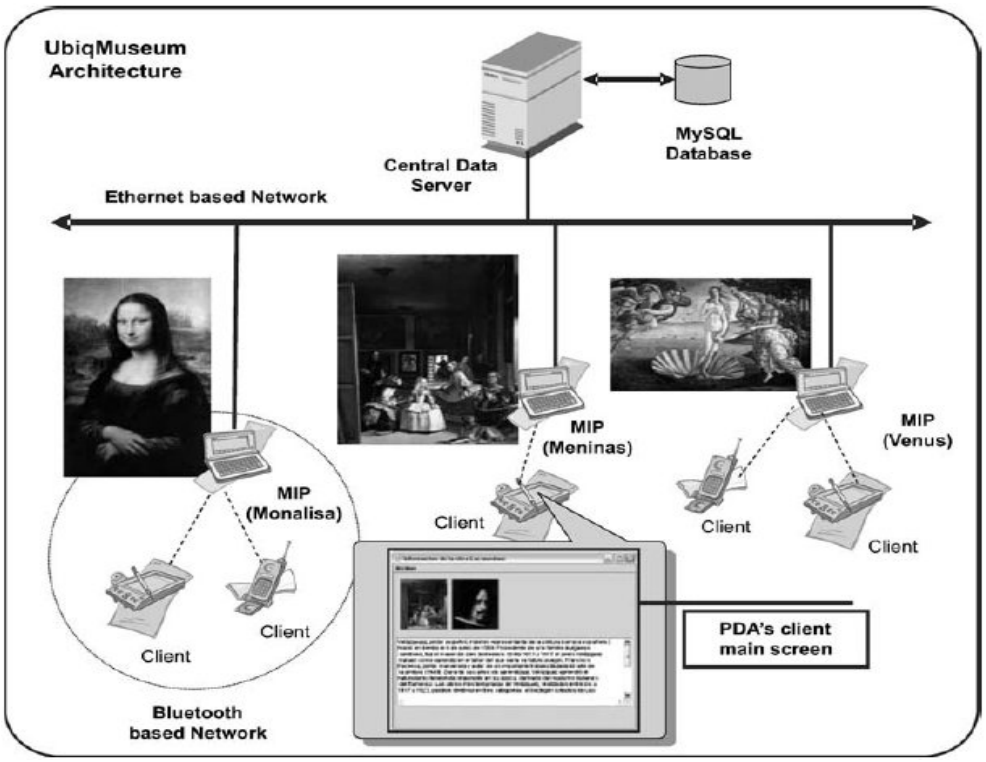


Figura 2.2: Arquitetura do UbiqMuseum

2.2.2 Projeto Imity

O Imity [6] sugere a criação de um espaço voltado para o "encontro" de seus usuários, onde serão possíveis interações e trocas de experiências, como ocorre nos atuais sites de relacionamentos, porém, sendo constituído por celulares com a tecnologia Bluetooth.

Em linhas gerais, o Imity cria um conceito de identidade móvel, fornecendo a possibilidade de uma"socialização digital" em um mundo ubíquo. A forma encontrada para a criação deste espaço de relacionamento é a utilização da comunicação dos clientes com um servidor web. Cada cliente (celular) possui uma identificação exclusiva dentre os demais e tem como função principal executar constantes buscas por outros dispositivos via Bluetooth e registrá-las no servidor.

Para que possam ocorrer as interações é criado um histórico de cada dispositivo. A criação deste histórico possibilita o desenvolvimento de um perfil que conterá dados sobre o "dia-a-dia do dispositivo", sendo este disponibilizado. Desse modo, torna-se possível conhecer mais sobre cada um dos usuários e, também, possibilita que relações fora do ambiente virtual (Imity)

possam ocorrer entre aqueles que encontraram outras pessoas com experiências semelhantes ou do seu interesse.

2.2.3 Projeto Buttons

O projeto Buttons [32] exemplifica bem a vasta gama de aplicações da computação ubíqua. É interessante sua descrição por propor a idéia da computação a qualquer instante.

O projeto Buttons consiste na criação de uma máquina fotográfica que, como colocado pela própria desenvolvedora, "tira outras fotos". Esta câmera nada mais é do que uma caixa selada que contém em seu interior um aparelho celular. A questão é: como tirar fotos com uma caixa selada?

O Buttons não possui lentes fotográficas e acaba, na verdade, funcionando mais como um relógio do que, efetivamente, como uma câmera. Quando seu botão é pressionado inicia-se um busca na internet (como no conhecido site de compartilhamento de imagens Flickr [9]) por outras fotos tiradas naquele exato momento. Essa busca pode demorar minutos ou, até mesmo, horas, tendo em vista que é necessário aguardar até que seja compartilhada uma foto sob as condições citadas anteriormente. Ao encontrar a foto, a imagem é imediatamente exibida no display do dispositivo.

O ponto interessante nesse projeto é o fato de toda computação ocorrer de forma transparente. Supondo que houvesse a possibilidade de localizar fotos que, além do mesmo instante, fosse tirada do mesmo local e sua exibição fosse em tempo mais hábil, o usuário do Buttons acabaria por ter a sensação de que foi exatamente aquela foto que ele acabou de tirar, mascarando toda computação e processo de busca existente.

Capítulo 3

Tecnologia Bluetooth

Bluetooth é um padrão para comunicação sem fio que utiliza tecnologia de transmissão via rádio de curto alcance, fornecendo baixo custo e baixo consumo de energia [27].

Os dispositivos desta tecnologia possuem uma velocidade de comunicação de 1 Mbps a 3 Mbps (representando, respectivamente, as versões 1.2 e 2.0 + *Enhanced Data Rate*) e são divididos em três classes que indicam a faixa de alcance conseguida, como mostra a Tabela 3.1 [4]. A classe mais comumente utilizada é a de número dois, encontrada nos dispositivos móveis; enquanto que os dispositivos referentes a classe três foram desenvolvidos primeiramente para casos de uso voltados à indústria.

Tabela 3.1: Classes da Tecnologia Bluetooth

Classes	Alcances	Potência Máxima de Saída (Pmax)
Classe 3	1 - 3 metros	1 mW (0 dBm)
Classe 2	10 - 33 metros	2,5 mW (4 dBm)
Classe 1	100 - 300 metros	100 mW (20 dBm)

Inicialmente, a tecnologia desenvolvida pela empresa Ericsson em 1994 visava apenas a eliminação de cabos dos periféricos criados para seus celulares. No entanto, com o passar dos anos, expandiu-se para inúmeros dispositivos como os PDAs, computadores pessoais (PCs), teclados, mouses, pontos de acesso de rede, fones de ouvido, dentre inúmeros outros. Sua especificação foi desenvolvida pela *Bluetooth Special Interest Group* (Bluetooth SIG), no qual o trabalho de seus membros (como Toshiba, Intel, IBM e Nokia) possibilitou a criação de um padrão aberto que assegurou uma rápida aceitação e grande compatibilidade no mercado. Hoje o Bluetooth é suportado por mais de 2100 companhias ao redor do mundo [27].

Com essa enorme aceitação e mediante a possibilidade de prover comunicação entre dis-

positivos, a *Wireless Personal Area Network* (WPAN), baseada na especificação do Bluetooth, tornou-se um padrão da IEEE sob a denominação de 802.15 WPANs [27], a qual pode ser encontrada na especificação IEEE 802.15 [12]. A especificação do Bluetooth define uma quantidade limitada de dispositivos que podem estar participando simultaneamente de uma comunicação. Tal limitação gera o que chamamos de Piconet (formada por oito dispositivos). No entanto, essas Piconets podem efetuar associações formando as Scatternets. Ambas serão descritas com maiores detalhes na Seção 3.3. Através destas topologias, os dispositivos se capacitam a efetuar comunicações tanto ponto a ponto, ou seja, comunicações diretas, quanto indiretas, baseadas na utilização de saltos (*hops*) [27].

3.1 A Pilha de Protocolos Bluetooth

Com o auxílio da Figura 3.1 [27] podemos observar a organização da pilha de protocolos do Bluetooth.

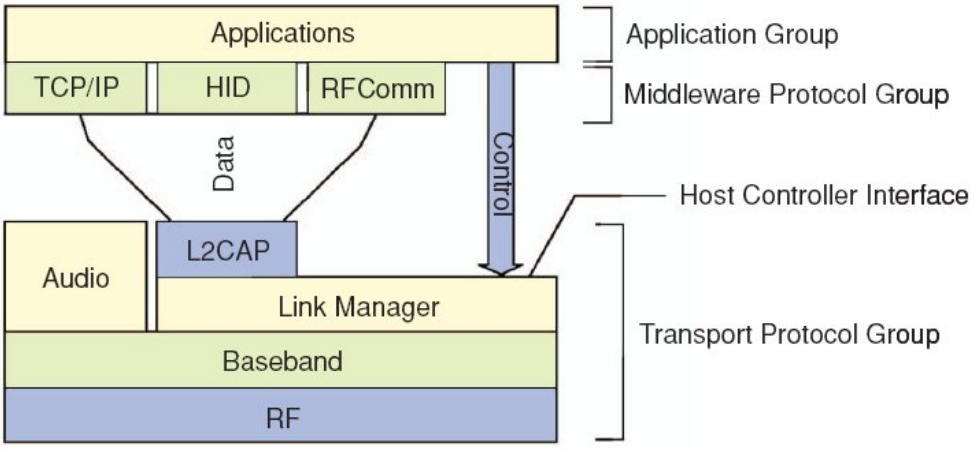


Figura 3.1: Pilha de Protocolos Bluetooth

A especificação divide a pilha em três camadas bem definidas, sendo elas, o grupo de protocolos de transporte, grupo de protocolos de *middleware* e grupo de aplicações. O grupo de protocolos de transporte é responsável pela localização de outros dispositivos e por todo gerenciamento das ligações com as camadas de níveis mais altos e com as aplicações. Fazem parte do grupo de transporte as camadas [4]:

- **Radio**: Bluetooth opera sob a banda não licenciada 2,4 GHz ISM (*Industrial, Scientific,*

Medical), onde seu *transceiver* [1] implementa um esquema de saltos de freqüência (*frequency hops*) para evitar interferências e perdas de força. Sua faixa de utilização é de 2400 a 2483,5 MHz e seus 79 canais são ordenados de 0 a 78, possuindo um espaçamento de 1 MHz começando em 2402 Mhz;

- **Baseband**: Parte do sistema Bluetooth que especifica ou implementa o acesso ao meio e os procedimentos da "camada física" entre os dispositivos Bluetooth. Aqui são definidas as regras que os dispositivos mestres ou escravos de uma Piconet precisam assumir e, também, a seqüência de saltos de freqüência utilizada por eles. Esta camada suporta dois tipos de ligações, as síncronas (Synchronous Connection-Oriented – SCO) e as assíncronas (*Assynchronous Connection-Less* - ACL). As conexões síncronas são caracterizadas por uma rápida e constante transmissão de dados. Estabelecendo uma conexão síncrona o dispositivo reserva um espaço de tempo para seu uso e os pacotes transmitidos são tratados antes de quaisquer pacotes transferidos por conexão assíncrona. As assíncronas, por sua vez, podem enviar pacotes de 1, 3 ou 5 espaços de tempo, no entanto, sem efetuar reserva prévia para tal operação;

- **Link Manager**: Implementa o *Link Manager Protocol* (Protocolo de Gerenciamento de Ligações – LMP), o qual é utilizado para controlar e negociar todo e quaisquer aspectos operacionais entre dois dispositivos. Isto inclui, por exemplo, a configuração e controle das ligações e transportes lógicos e o controle das ligações físicas;

- **L2CAP**: Fornece a interface entre as camadas mais altas e as camadas de transporte. L2CAP suporta a multiplexação de protocolos de mais alto nível (como o *Radio Frequency Communications Protocol* – RFCOMM), segmentação de pacotes e transferência das informações de qualidade de serviço. Este protocolo baseia-se no conceito de canais, onde cada um possui um identificador único, o CID. A Figura 3.2 [4] mostra a utilização dos CIDs. Os canais de dados orientados a conexão representam a conexão entre dois dispositivos, onde é verificada a existência de um CID em cada extremidade dos canais. Os canais não-orientados a conexão representam conexões com restrição de fluxo de dados para uma única direção. Este tipo de ligação é utilizada para formação de "grupos" de

[1] Termo normalmente utilizado em dispositivos *wireless* que descreve a combinação de transmissor e receptor em um mesmo módulo.

canais, no qual o CID na fonte pode representar um ou mais dispositivos. Os canais de sinalização são exemplos de canais reservados. Estes são usados para criar e estabelecer canais de dados orientados a conexão e para negociar alterações nas características tanto dos canais orientados a conexão quanto dos não-orientados;

- **HCI**: Provê uma interface de comandos para as camadas de banda básica e gerenciador de ligações e acesso a parâmetros de configuração. Não é uma camada obrigatória na especificação. Seu objetivo é de permitir comunicação transparente entre os dispositivos e a utilização de protocolos e aplicativos de níveis mais altos.

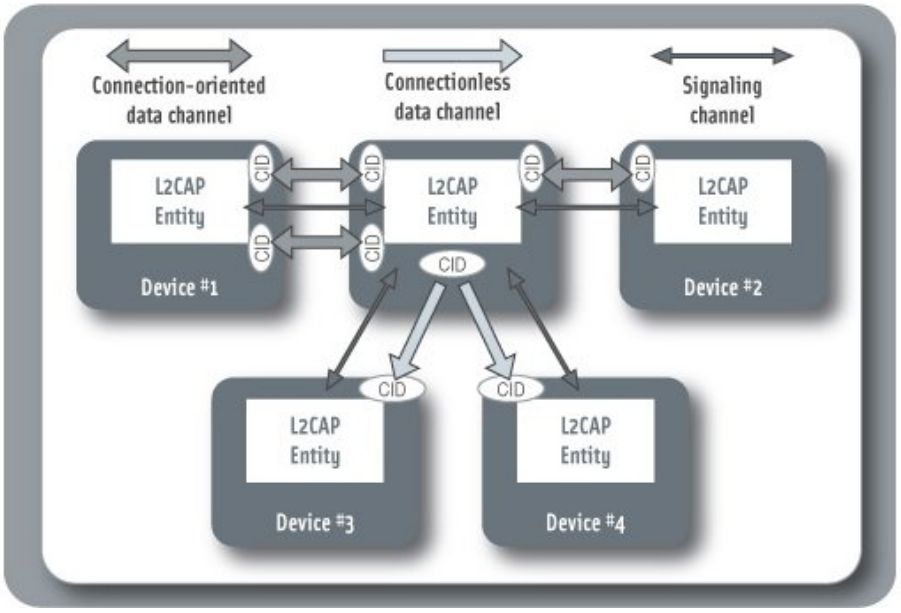


Figura 3.2: Exemplo de utilização dos CIDs

O grupo de *middlewares* inclui protocolos desenvolvidos por terceiros, protocolos padrões da indústria e os desenvolvidos pelo Bluetooth SIG. A existência destes protocolos permite que tanto as novas quanto as aplicações já existentes possam operar sobre conexões Bluetooth. Os protocolos do padrão industrial implementados abrangem o Protocolo Ponto-a-Ponto (PPP), Protocolo da Internet (IP), Protocolo de Controle de Transmissão (TCP), Protocolo de Aplicações Sem-fio (WAP) e o Protocolo de Troca de Objetos (OBEX). Os protocolos desenvolvidos pelo Bluetooth SIG compreendem o RFCOMM que permite que aplicações mais antigas possam operar sobre o Bluetooth, o Protocolo de Controle Sinalizado de Telefonia Baseado em

Pacotes (TCS) que coordena as operações telefônicas e o Protocolo de Descoberta de Serviços (SDP) que permite aos dispositivos obter informações sobre serviços disponibilizados pelos outros dispositivos.

O grupo de aplicações consiste nas aplicações habilitadas à utilização do Bluetooth.

3.2 Segurança

As tecnologias sem fio apresentam um desafio especial na questão segurança, pois sua comunicação pode ser facilmente interceptada.

Observando este problema, a SIG criou seu grupo de especialistas em segurança (*Security Expert Group*) formado pelos engenheiros de suas companhias membro, os quais ficam responsáveis por analisar e sugerir soluções para os problemas de segurança encontrados na especificação do Bluetooth.

Atualmente um sistema Bluetooth baseia-se em três componentes de segurança, sendo eles autenticação, criptografia e autorização. Em adição, também são propostos três modos de segurança, proporcionando, assim, níveis de confiabilidade diferentes [4].

3.2.1 Modos de Segurança

Os modos de segurança fazem parte do Perfil Genérico de Acesso (*Generic Access Profile* – GAP) [2]. Todos os dispositivos Bluetooth realmente qualificados possuem uma implementação do GAP, portanto, todos apresentam um modo de segurança implementado. Os níveis modos de segurança definidos pelo GAP são [14]:

1. Sem segurança: Neste modo os dispositivos nunca iniciam qualquer procedimento de segurança. Possuindo um suporte opcional à autenticação;

2. Segurança ao nível dos serviços: Modo utilizado pela maioria dos dispositivos. Atuando no nível dos serviços, são eles que decidem quando um procedimento de segurança é ou não necessário. Como a inicialização das atividades neste modo acontecem nos níveis mais altos da pilha de protocolos do Bluetooth, os desenvolvedores ficam habilitados a decidir se a segurança será implantada em seus serviços;

[2]Base para todos os perfis em um sistema Bluetooth. Define as funcionalidades básicas do Bluetooth, como a criação de ligações L2CAP, por exemplo [14]

3. Segurança ao nível das ligações: Neste modo os procedimentos de segurança são iniciados durante o processo de estabelecimento das ligações. Caso ocorra falha na segurança, a ligação também falhará. Aqui, as configurações de segurança vêm definidas de fábrica, não permitindo que o desenvolvedor de um serviço possa decidir algo sobre ela.

3.2.2 Autenticação – Emparelhamento (*Pairing*) e União (*Bonding*)

A autenticação em uma comunicação Bluetooth consiste do emparelhamento e união. A união corresponde a autenticação em si e depende do compartilhamento de uma chave de autenticação. Caso ela ainda não tenha sido trocada entre os dispositivos, o emparelhamento entra em ação.

O emparelhamento é todo processo de construção da chave de autenticação. Faz parte do processo a criação de uma chave de inicialização que é formada por uma entrada do usuário, um número aleatório e o endereço do *hardware* Bluetooth, e uma chave de autenticação que baseia-se na união de um número aleatório com o endereço do hardware Bluetooth dos dois dispositivos. A chave de inicialização é utilizada para criptografar a troca da chave de autenticação e, após isso, é descartada.

Após o processo de emparelhamento ambos dispositivos possuem a chave armazenada e podem se comunicar normalmente. Só haverá um novo emparelhamento se este for solicitado pelos usuários ou se a chave armazenada for apagada, do contrário, futuras comunicações serão efetuadas utilizando-a.

3.2.3 Criptografia (*Encryption*)

Sempre que dois dispositivos já estão devidamente autenticados, uma requisição de criptografia pode ser feita. Antes que a criptografia possa ser iniciada, os dispositivos precisam decidir qual o modo e o tamanho da chave que irão utilizar.

Em uma comunicação padrão entre dois dispositivos o modo de criptografia de pacotes ponto-a-ponto (*point-to-point packets encryption*) é o mais comum. Existe também o modo sem criptografia (*no encryption*), o qual será escolhido caso um dos dispositivos não possua suporte ao serviço. Um terceiro modo é o de criptografia de pacotes ponto-a-ponto e broadcast (*point-to-point and broadcast packets encryption*).

3.2.4 Autorização

Refere-se à permissão que dado dispositivo pode fornecer a outro remoto, para que este acesse um serviço em particular. Observe que, para um dispositivo conseguir autorização, ele deve previamente estar autenticado.

3.3 Comunicação

Um transceiver Bluetooth é um dispositivo FHSS (*Frequency-Hopping Spread-Spectrum*) que usa a banda de freqüência não licenciada de 2,4 GHz ISM. Na maioria dos países são disponibilizados 79 canais para utilização, no entanto, alguns possuem apenas 23 canais [27], sendo cada canal de 1 MHz.

Quando um dispositivo se conecta a outro, incia-se o processo de saltos de freqüência (mudanças de freqüência), que ocorrem a uma taxa de 1600 vezes por segundo. No caso do dispositivo estar em modo de busca (*inquiry mode*) ou em modo de convite (*paging mode*) essa taxa de mudanças aumenta para 3200 vezes por segundo.

A especificação do Bluetooth usa duas formas de comunicação entre os dispositivos a TDD (*Time Division Duplexing*) e a TDMA (*Time Division Multiple Access*). A comunicação é dividida em pequenos espaços de tempo, onde cada um possui 625 microssegundos . Este espaço de tempo representa, também, o tamanho de um pacote. Na camada *baseband* um pacote consiste de um código de acesso, um cabeçalho e o dado em si.

O código de acesso é formado pelo endereço da Piconet (utilizado para possibilitar a comunicação entre Piconets), contendo normalmente 72 bits de comprimento. O cabeçalho contém o dado de controle de ligação (*link control data*), informando o endereço para um dispositivo escravo atualmente ativo. Normalmente seu tamanho é de 18 bits. O dado pode conter de 0 a 2745 bits (Figura 3.3 [27]).

Figura 3.3: Formato de um pacote

Para as comunicações SCO os pacotes necessitam possuir exatamente o tamanho de um

espaço de tempo, espaço este reservado para transmissão. Já no caso das comunicações ACL, os pacotes podem possuir 1, 3 ou 5 espaços de tempo, no entanto sem reserva prévia. Os pacotes transmitidos de forma síncrona possuem preferência sobre os assíncronos, ou seja, os pacotes transmitidos são tratados antes de quaisquer outros que foram enviados de forma assíncrona.

Todas as comunicações ocorrem entre um mestre e um escravo utilizando TDD. Este mestre é responsável por coordenar a troca de informações através de um esquema de eleição (*polling-based package transmission*). Neste esquema o mestre verifica cada escravo, "perguntando" se este possui dados a transmitir. O escravo só estará apto a efetuar transmissão se for requisitado pelo mestre, sendo que esta transmissão deve ser executada exatamente no espaço de tempo seguinte a visita.

O mestre transmite apenas nos espaços de tempo de numeração par, enquanto os escravos transmitem nos ímpares. Cada espaço de tempo utiliza uma freqüência diferente, a qual é selecionada através dos saltos.

3.3.1 Redes Bluetooth

O protocolo permite a geração de comunicação entre os dispositivos utilizando conexões sem fio. Esta seção descreverá as características dessas comunicações, incluindo as topologias básicas das redes Bluetooth.

A criação de uma rede Bluetooth baseia-se no conceito de canais já discutido. Para que os dispositivos estejam localizados em uma mesma rede, eles necessitam estar compartilhando um mesmo canal. Um ou mais dispositivos utilizando o mesmo canal físico forma o que chamamos de Piconet (Figura 3.4 [19]).

Essas Piconets são formadas por até oito nós ativos, onde um é eleito como mestre e os demais permanecem como escravos. Esta limitação é imposta apenas para os nós que se encontram ativos. Dessa forma, é possível a existência de um número bem maior de nós em cada Piconet, sendo que estes devem estar em estado estacionado (*parked state*). O número máximo de nós estacionados na Piconet é de 255, os quais, segundo a SIG, possuem um endereçamento direto (o *parked slave address*) ou podem ser endereçados através do seu *Bluetooth device address* (o endereço de *hardware* do Bluetooth).

Além dos modos ativo e estacionado, um dispositivo Bluetooth pode estar em outros dife-

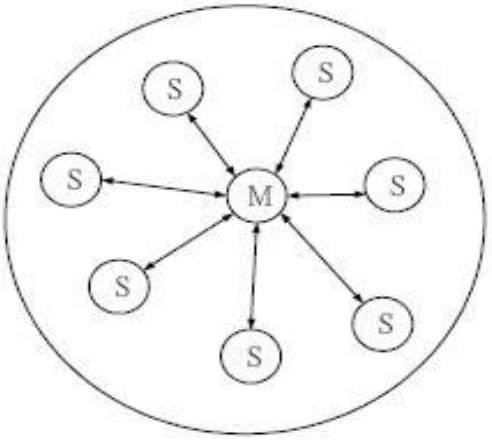


Figura 3.4: Uma Piconet formada por um mestre e sete escravos

rentes estados, como exemplifica a Figura 3.5 [4]

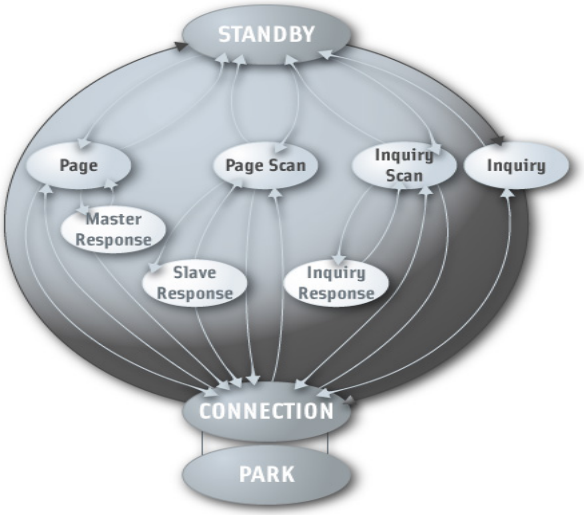


Figura 3.5: Possíveis estados do Bluetooth

- **Standby** (modo de espera): quando o dispositivo está ligado, porém, sem estar participando de quaisquer Piconets;
- **Inquiry** (modo de busca): o Bluetooth entra neste estado quando envia requisições para localização de outros dispositivos, com os quais irá se conectar;
- **Page** (modo de convite): estado utilizado por mestres para verificar a existência de outros dispositivos que desejam ingressar à Piconet;

- **Conected** (modo conectado): quando a comunicação se dá de forma correta entre o mestre e o novo dispositivo, este assume a situação de escravo, recebendo um endereço de dispositivo ativo (*active address*) e entrando em modo conectado.

A partir do momento que o novo escravo encontra-se no modo conectado, pode passar a transmitir dados sempre que for "visitado" pelo mestre, entrando no estado de transmissão (*transmission state*). Ao finalizar, volta a seu estado conectado. Outro estado possível é o de descanso (*sniff state*). Nele o dispositivo inicia um processo de economia de energia, onde continua apto a efetuar transferências. Diferente do modo bloqueado (*hold*) que também visa economia de energia, porém permanecendo como inativo neste período.

Dispositivos Bluetooth podem, também, servir como pontes unindo as Piconets. Esses dispositivos podem ser escravos em todas as redes a qual pertence ou mestre em uma delas e escravo nas demais. Pertencendo a mais de uma Piconet, eles devem sincronizar sua presença em todas, no entanto, não podem efetuar operações em mais do que uma simultaneamente [11]. Essa interconexão de Piconets forma as chamadas Scatternets (Figura 3.6 [11]). As Scatternets vem a suprir a limitação do número de dispositivos ativos imposta pelas Piconets, uma vez que podem existir n dispositivos permitindo esta junção.

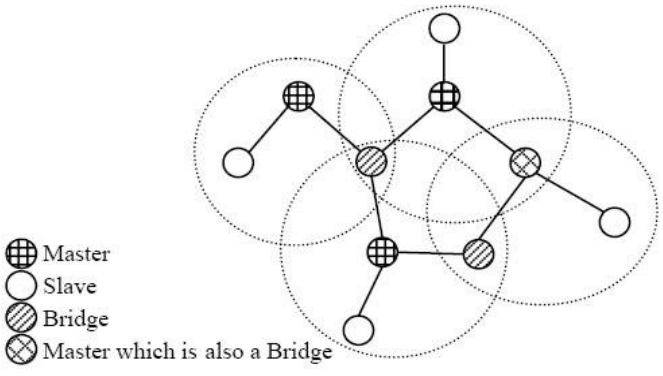


Figura 3.6: Um exemplo de Scatternet

A possibilidade de criação destas redes pode ser vista como ponto chave e principal inspiração para o desenvolvimento do presente trabalho, já que, com tal característica, a subutilização do Bluetooth pode ser reduzida.

Capítulo 4

Plataforma J2ME

O objetivo principal da plataforma Java [23] é conseguir englobar a maior variedade de aplicações e executar em diferentes tipos de dispositivos, desde os com maior grau de limitação até grandes servidores corporativos. Para tal, a plataforma Java possui três versões principais, sendo elas: *Java 2 Standard Edition* (J2SE), mais voltada para utilização em *desktops*; *Java 2 Enterprise Edition* (J2EE), uma plataforma mais robusta, podendo ser utilizada em aplicações do tipo multi-usuário corporativas; e a *Java 2 Micro Edition* (J2ME), desenvolvida para dispositivos pequenos e com poder computacional limitado, e outras duas versões mais específicas, a *Java Card*, voltada para dispositivos embarcados com limitação de processamento e armazenamento (como *smart cards*[1]) e a *Java FX*, uma plataforma de desenvolvimento de aplicações multimídia em *desktops* e dispositivos móveis. Este capítulo trata especificamente da plataforma J2ME, sendo baseado no texto do livro *Beginning J2ME – From Novice to Professional* [15].

J2ME disponibiliza um subconjunto de componentes do J2SE, como máquinas virtuais mais reduzidas e uma gama de APIs mais "enxutas". No entanto, não se pode considerar que J2ME é simplesmente um fragmento de *software* ou de especificação. Na realidade deve-se dizer apenas que trata-se de Java para pequenos dispositivos. Esses pequenos dispositivos compreendem desde *pagers*, telefones celulares, PDAs, até os *set-top boxes*[2].

A J2ME está dividida em **configurações**, **perfis** e **APIs opcionais**, onde cada uma provê informações específicas sobre as diferentes famílias de dispositivos. As configurações, por exem-

[1]Como os *chips* encontrados nos celulares GSM. Sua principal diferença para, por exemplo, os cartões de crédito das agências bancárias, é que um *smart card* possui capacidade de processamento, apresentando, inclusive, memória.

[2]Nome dado aos conversores que são tipicamente conectados a um televisor e a alguma fonte de sinal, transformando este sinal em um formato que possa ser apresentado na tela.

plo, são projetadas para categorias específicas de dispositivos, baseando-se nas suas limitações de memória e poder de processamento. Ela especifica, portanto, uma Máquina Virtual Java (JVM) que pode ser facilmente portada para dispositivos que suportam tal configuração. Define também qual subconjunto de APIs da J2SE será utilizado, assim como quais APIs adicionais que podem ser disponíveis.

Os perfis são mais específicos que as configurações. Eles baseiam-se nas configurações e provêem APIs adicionais, possibilitando a criação de interface com o usuário e armazenamento, por exemplo.

As APIs opcionais disponibilizam funcionalidades adicionais especificas que podem ser incluídas em um perfil ou configuração em particular. Este conjunto formado pela configuração, perfil e APIs opcionais formam o que é denominado pilha (*stack*).

Atualmente existe uma série de configurações e perfis, porém, os mais relevantes para os desenvolvedores J2ME podem ser visualizados na Figura 4.1 [15]. Tais componentes serão descritos nas próximas seções.

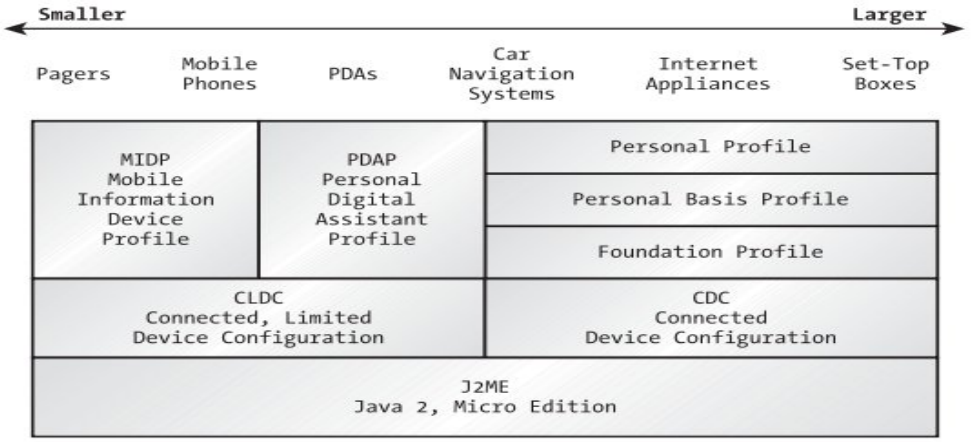


Figura 4.1: Perfis e Configurações J2ME mais comuns

4.1 Configurações

Descrevendo de forma mais pontual, uma configuração especifica uma JVM e um conjunto de APIs para uma família específica de dispositivo. Atualmente elas são duas: Configuração para Dispositivo Conectado (*Connected Device Configuration* – CDC) e a Configuração para

Dispositivos Limitados e Conectados (*Connected, Limited Device Configuration* – CLDC).

Geralmente os perfis e configurações são descritos com base na capacidade de memória. Normalmente um limite mínimo de memória é especificado. E, exatamente por este motivo, existe esta divisão entre CDC e CLDC.

4.1.1 Connected Device Configuration

Um dispositivo conectado possui, no mínimo, 512 KB de memória apenas de leitura (*Read-Only Memory* – ROM) e 256 KB de memória de acesso aleatório (*Random Access Memory* – RAM). Os CDCs são projetados para dispositivos como *set-top boxes*, sistemas de navegação automobilístico e alguns modelos de PDAs. Esta configuração especifica que uma JVM completa (como definida na *Java Virtual Machine Specification, 2nd Edition* [18]) precisa ser suportada.

4.1.2 Connected, Limited Device Configuration

CLDC é a configuração que mais traz interesse para o desenvolvimento deste trabalho, uma vez que engloba os telefones celulares, *pagers*, PDAs e alguns outros dispositivos de capacidade similar. O foco da CLDC é a utilização de dispositivos menores do que os verificados na CDC.

Com relação a capacidade de memória, esta configuração é projetada para dispositivos que possuem cerca de 160 KB a 512 KB de memória total, incluindo um mínimo de 160 KB de ROM e 32 KB de RAM. Se formos comparar estes requisitos com os da J2SE que necessita de um mínimo de 10 MB, fica mais fácil apreciar o esforço empreendido pela J2ME.

As implementações da CLDC estão fundadas sobre uma pequena JVM chamada KVM. Seu nome vem do fato de que esta JVM tem seu tamanho medido em KB e não MB, como uma máquina virtual tradicional. Devido a sua limitação de tamanho, uma KVM não pode fazer tudo o que uma máquina virtual tradicional faz no universo J2SE, como, por exemplo permitir a adição de métodos nativos em tempo de execução.

4.2 Perfis

O perfil é uma camada disposta logo acima das configurações (Figura 4.1), adicionando as APIs e especificações necessárias para o desenvolvimento de aplicações para uma família

específica de dispositivos. Como podemos verificar na Figura 4.1, existe uma quantidade razoavelmente grande de perfis já desenvolvidos.

O Perfil Base (*Fundation Profile*) é uma especificação para dispositivos que podem suportar um robusto ambiente de redes J2ME. Ele não suporta, por exemplo, interfaceamento com o usuário, no entanto, outros perfis podem ser dispostos sobre o Perfil Base para prover outras funcionalidades.

Dispostos logo acima do Perfil Base temos o Perfil Base Pessoal (*Personal Basis Profile*) e o Perfil Pessoal (*Personal Profile*). A combinação formada por CDC + Perfil Base + Perfil Base Pessoal + Perfil Pessoal é um projeto para a nova geração de Ambientes Pessoais para Aplicações Java (*PersonalJava Application Environment*), como descrito em [24].

O Perfil PDA, o qual é construído sobre a CLDC, é projetado para dispositivos como *palm tops* com um mínimo de 512 KB de memória (distribuídos entre ROM e RAM) e um máximo de 16 MB. Ele está localizado entre o Perfil Dispositivo Móvel de Informação (*Mobile Information Device Profile* – MIDP) e o Perfil Pessoal.

4.2.1 Mobile Information Device Profile

O presente perfil engloba dispositivos como telefones celulares e *pagers* mais avançados, tornado-se o principal perfil para este trabalho. O MIDP apresenta como principais características exigir dispositivos com:

- um mínimo de 256 KB de memória ROM apenas para sua implementação, ou seja, este valor é adicionado ao já requerido pelo CLDC;
- um mínimo de 128 KB de RAM para a pilha de tempo de execução Java;
- um mínimo de 8 KB de memória de escrita não volátil para dados persistentes;
- uma tela de no mínimo 96x54 pixels;
- suporte a teclados ou tela sensível ao toque; e
- conexão de rede de dois caminhos, possivelmente intermitente.

Este perfil está disponível atualmente em três versões: MIDP 1.0, MIDP 2.0 e MIDP 2.1. Grande parte dos dispositivos mais recentes suportam o MIDP 2.0. Se comparado à versão

anterior, o MIDP 2.0 apresenta, dentre outras, a inclusão de funcionalidades como suporte à multimídia, uma nova API de interface para jogos e suporte para conexões HTTP. É importante citar, também, que MIDP 2.0 é totalmente compatível com MIDP 1.0.

4.3 MIDlets

Aplicações MIDP são costumeiramente chamadas de MIDlets, nome sugerido para dar seqüência a série iniciada por **applets** e **servlets**. Escrever um código para MIDlet é relativamente fácil para quem já possui algum conhecimento prévio como programador Java, pois, antes de qualquer coisa, a linguagem de programação em questão é propriamente a Java.

De uma forma geral, os MIDlets podem ser comparados aos applets J2SE. A existência de um MIDlet baseia-se em quatro diferentes estados: carregado (*loaded*), ativo (*active*), pausado (*paused*) e destruído (*destroyed*). A Figura 4.2 [16] apresenta uma visão do ciclo de vida de um MIDlet. Após um MIDlet ser "carregado" em um dispositivo e seu construtor ser chamado, ele encontra-se em seu estado carregado. Este estado sempre é utilizado antes que o gerenciador de aplicações inicie efetivamente o MIDlet através da chamada os método startApp(). Após a chamada deste método o aplicativo permanece no estado ativo até que o gerenciador chame o método pauseApp() ou destroyApp(), onde pauseApp() coloca o MIDlet em modo de espera, parando animações e outros recursos para reduzir o consumo de bateria ou minimizar eventuais conflitos com outras aplicações que possam estar em execução, e destroyApp() encerra o aplicativo.

4.4 Desenvolvimento de Aplicativos Utilizando Bluetooth

Visando a criação de aplicações baseadas na utilização da tecnologia Bluetooth, esforços de integrantes de diversas empresas, como Nokia, Sun Microsystems, IBM, dentre outras, resultaram no desenvolvimento da JSR-82 (baseada na versão 1.1 da especificação do Bluetooth), um conjunto de APIs para ser utilizado juntamente com a plataforma de desenvolvimento J2ME [26].

O objetivo principal da JSR é ocultar toda complexidade da pilha de protocolos Bluetooth, permitindo que o desenvolvedor mantenha seu foco voltado para aplicação e não para os detalhes de baixo nível da tecnologia.

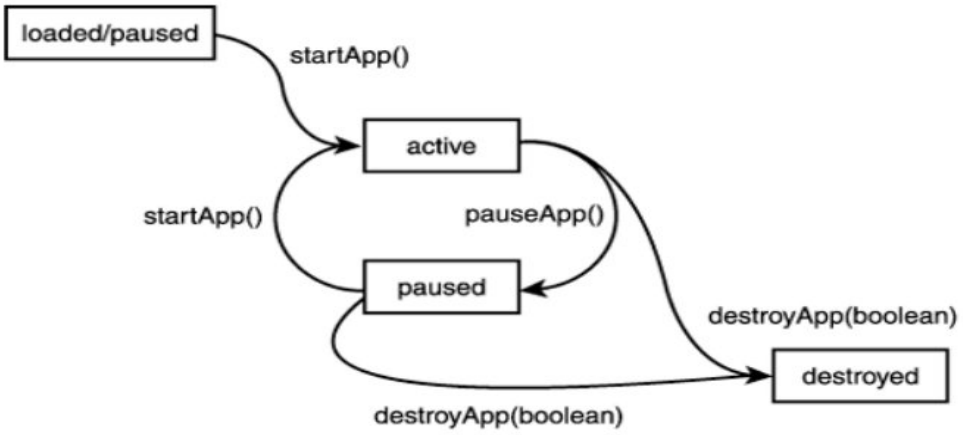


Figura 4.2: Ciclo de vida de um MIDlet

JSR-82 consiste de dois pacotes: a API Bluetooth e a API OBEX, sendo o último totalmente independente, podendo ser utilizado separadamente do primeiro. É importante mencionar que as APIs Java para o Bluetooth não implementam sua especificação, e sim mecanismos para acessar e controlar dispositivos dotados da tecnologia [21].

Para suportar a JSR-82 alguns requisitos de sistema são especificados. Exemplos de requisitos pertencentes a esta especificação são: a necessidade do dispositivo estar qualificado de acordo com o programa de qualificação do Bluetooth; suporte do sistema às camadas de comunicação da pilha de protocolos Bluetooth; concessão de acesso ao SDP, RFCOMM e L2CAP para a API; e implementação de um Centro de Controle Bluetooth (*Bluetooth Control Center* – BCC).

O BCC é responsável por permitir que o usuário ou o próprio dispositivo possa definir valores de certos parâmetros da pilha Bluetooth. Os dispositivos que suportam esta API permitem que múltiplos programas sejam executados de forma concorrente. Sendo assim, o BCC irá prevenir que uma aplicação possa danificar o funcionamento de outra. O centro de controle pode ser uma aplicação nativa, uma aplicação em uma API separada ou simplesmente um grupo de configurações que foram especificadas pelo fabricante e não podem ser alteradas pelo usuário. BCC não está definida na especificação da JSR, mas, sem dúvida, é uma parte importante da sua segurança.

A funcionalidades fornecidas pela JSR-82 estão divididas em três categorias [26]:

- ***Discovery*** (descoberta) - implementa três serviços:
  - *Device Discovery* (Descoberta de Dispositivos): responsáveis por disparar o processo de busca por dispositivos. Para efetuar uma busca por dispositivos a aplicação necessita possuir objeto ouvinte (*listener*) que será notificado sempre que um novo dispositivo for encontrado;
  - *Service Discovery* (Descoberta de Serviços): este serviço é representado pela classe DiscoveryAgent, a qual fornece métodos para busca de serviços em dispositivos que estão desenvolvendo papel de servidores Bluetooth e para inicialização da transação de descoberta de serviços;
  - *Service Registration* (Registro de Serviços): o registro de serviços é um processo colaborativo entre o servidor de aplicações, a implementação da API e a pilha de protocolos Bluetooth. Registrar um novo serviço nada mais é, na verdade, do que disponibilizá-lo. Quando um serviço é registrado, o dispositivo passa a trabalhar como um servidor Bluetooth e fica aguardando que outros dispositivos conectem-se a ele.

- ***Communication*** (comunicação): Esta JSR fornece todo o acesso necessário para permitir o estabelecimento das conexões e para utilizar estas conexões na comunicação Bluetooth entre as aplicações;

- ***Device Management*** (gerenciamento do dispositivo): Aqui está toda parte de possibilita ao dispositivo local responder às requisições de um remoto, sendo representada pelas classes que permitem acesso ao GAP. Dentre outros atributos, é possível obter informações do dispositivo remoto e controlar as funcionalidades padrões do dispositivo local. Toda parte de segurança do Bluetooth também é controlada nesta categoria.

O uso da JSR-82 é de grande importância para que consigamos realizar um desenvolvimento rápido das aplicações que utilizam Bluetooth. Da mesma forma que um *socket* funciona como uma interface de ligação entre a camada de aplicação e a de transporte dentro de uma máquina [17], o conjunto de APIs fornecidos pela JSR proporcionam ao usuário a possibilidade de trabalhar as questões de gerenciamento da tecnologia em alto nível, preocupando-se apenas com o cumprimento dos requisitos especificados para a aplicação.

4.5 Exemplos de aplicações J2ME

São apresentados, nesta Seção, exemplos de códigos-fonte de aplicações J2ME visando facilitar o entendimento da construção de uma aplicação J2ME. Serão apresentados um exemplo de MIDlet, um segundo exemplo utilizando a API JSR-82 para efetuar o processo de descoberta do Bluetooth e um terceiro exemplo mostrando a comunicação entre dispositivos com a JSR-82.

4.5.1 MIDlet Hello World

```
1 public class HelloWorld extends MIDlet implements CommandListener{
2
3         private List titleScreen;
4         private Display myDisplay;
5         private Command exitCommand = new Command("Sair", Command.EXIT, 1);
6
7         public HelloWorld() {
8                 myDisplay = Display.getDisplay(this);
9
10                titleScreen = new List("Teste de implementação", List.IMPLICIT);
11                titleScreen.append("Hello World!!!", null);
12                titleScreen.addCommand(exitCommand);
13                titleScreen.setCommandListener(this);
14        }
15
16        protected void destroyApp(boolean arg0) throws MIDletStateChangeException {
17                titleScreen.deleteAll();
18                titleScreen = null;
19                myDisplay = null;
20        }
21
22        protected void pauseApp() {
23        }
24
25        protected void startApp() throws MIDletStateChangeException {
26                myDisplay.setCurrent(titleScreen);
27        }
28
29        public void commandAction(Command c, Displayable d) {
30                try {
31                        if(c == exitCommand){
32                                this.destroyApp(true);
33                                notifyDestroyed();
34                        }
35                } catch (MIDletStateChangeException e) {
36                        e.printStackTrace();
37                }
38        }
39 }
```

Figura 4.3: MIDlet Hello World

Este é um exemplo clássico de aplicação, no qual o objetivo é apenas exibir uma mensagem na tela do dispositivo. Nele podemos avaliar os principais aspectos que envolvem o desenvolvimento de um MIDlet.

Neste exemplo (Figura 4.3), através dos atributos titleScreen, myDisplay e exitCommand, é possível criar a interface com o usuário, a qual exibirá a mensagem de *Hello World* e possibilitará ao usuário encerrar a aplicação assim que desejado.

Os estados, como podemos observar, são tratados como métodos. Ao executar o método startApp(), a tela criada pela aplicação assume a exibição no dispositivo, exibindo os valores atribuídos pelo construtor da classe. O tratamento do comando exitCommand é feito através da implementação interface CommandListener, a qual insere o método commandAction, responsável por definir as ações que serão executadas pelos comandos.

No caso especial deste exemplo, apenas um comando de encerrar o aplicativo foi criado. Quando este é executado o método destroyApp() é chamado, finalizando, assim, a execução do MIDlet.

A aparência deste MIDlet fica semelhante à ilustrada na Figura 4.4, no entanto existe uma grande dependência quanto ao dispositivo que está sendo utilizado.

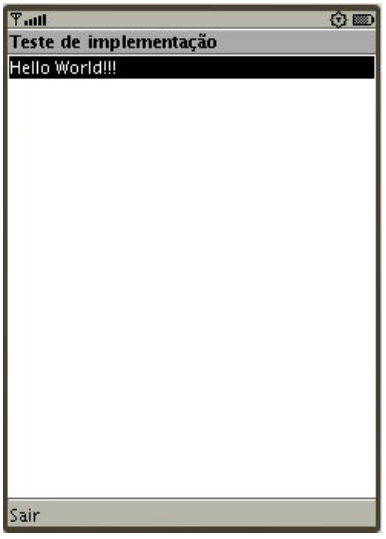


Figura 4.4: Exemplo de execução do MIDlet

4.5.2 Processo de Descoberta do Bluetooth

Os dispositivos que possuem a tecnologia Bluetooth necessitam efetuar um processo de descoberta para poder saber quais são os demais dispositivos que estão em sua área de cobertura. Este processo consiste no envio de diversas mensagens de descoberta e no aguardo de respostas

de outros dispositivos. Temos na Figura 4.5 um exemplo de busca por dispositivos utilizando a JSR-82.

```
1 private synchronized int findDevices() {
2         try {
3                 // limpando lista de dispositivos
4                 btDevicesFound.removeAllElements();
5                 isBTSearchComplete = false;
6                 LocalDevice local = LocalDevice.getLocalDevice();
7                 DiscoveryAgent discoveryAgent = local.getDiscoveryAgent();
8                 // buscando novos dispositivos
9                 discoveryAgent.startInquiry(DiscoveryAgent.GIAC, this);
10                while ((!isBTSearchComplete)) {
11                        // aguardando um tempo pre-determinado
12                        synchronized (this) {
13                                this.wait(BtOperations.BLUETOOTH_TIMEOUT);
14                        }
15                        // checando se a busca terminou
16                        if (!isBTSearchComplete) {
17                                // se a busca ainda não encerrou deve ser cancelada
18                                discoveryAgent.cancelInquiry(this);
19                        }
20                }
21        } catch (Exception e) {
22        }
23                return btDevicesFound.size();
24        }
25
26        public void deviceDiscovered(RemoteDevice remoteDevice, DeviceClass deviceClass) {
27
28                System.out.println("Dispositivo encontrado!!!");
29
30                btDevicesFound.addElement(remoteDevice);
31        }
32
33        public void inquiryCompleted(int arg0) {
34                isBTSearchComplete = true;
35                // notifica a thread principal de que a busca esta completa
36                synchronized (this) {
37                        this.notify();
38                }
39        }
40 }
```

Figura 4.5: Processo de Descoberta do Bluetooth

A base principal deste trecho de código está no comando discoveryAgent.startInquiry(DiscoveryAgent.GIAC, this) e nos métodos deviceDiscovered e inquiryCompleted. Estes métodos são gerados automaticamente através da implementação da interface DiscoveryListener, a qual é responsável por toda busca por dispositivos ou serviços.

Quando o método startInquiry é invocado para realizar a busca de dispositivos, ocorre a mudança de estado do Bluetooth para seu estado de busca (*inquiry state*). Esta execução é assistida pelos métodos deviceDiscovered e inquiryCompleted. O método deviceDiscoreved informa ao aplicativo qual ação deverá ser executada quando um dispositivo remoto for encontrado. Neste caso os dispositivos estão sendo armazenados em um Vector. Já o método inquiryCompleted

informa a ação referente ao término da busca.

Ao referenciar a interface DiscoveryListener estamos, além dos métodos já citados, criando mais dois: servicesDiscovered e serviceSearchComplete. Estes dizem respeito à busca por serviço. O processo em si é análogo ao verificado para os dispositivos, exceto pelo fato de que um dado dispositivo remoto deve ser passado para que, então, seus serviços (ou um em especial) sejam descobertos.

4.5.3 Comunicação via Bluetooth

A comunicação através da tecnologia Bluetooth pode ser executada de diferentes formas e por diferentes protocolos. Temos a possibilidade de efetuar comunicação via IP, OBEX ou RFCOMM, por exemplo. O exemplo que será comentado nesta Seção utiliza trechos de códigos retirados da implementação do protocolo BtJAdhoc, utilizando comunicação via RFCOMM.

O trecho da Figura 4.6 ilustra como a criação e disponibilização de um serviço é feita. Quando desejamos criar um serviço, temos que "assinar" um número de identificação para ele, o que está sendo feito na linha cinco do trecho de código. Após essa assinatura ter sido concluída, podemos abrir um canal de comunicação com base no número de identificação criado e no protocolo que se deseja utilizar. O protocolo RFCOMM é referenciado no código por btspp (*Bluetooth Serial Port Protocol*). A URL que define este novo serviço é formada, basicamente, pelo tipo de protocolo a ser utilizado, o número do serviço criado e um nome que poderá ser atribuído ao serviço.

O restante do procedimento é muito semelhante ao utilizado para, por exemplo, a criação de *socket* de comunicação TCP. O serviço é disponibilizado pelo método open(url) e o servidor passa a ficar monitorando o canal a espera de novas conexões através da chamada ao método acceptAndOpen(). Após isto, basta tratar adequadamente, de acordo com as necessidades do usuário, o recebimento de algum pacote.

O processo de envio de pacotes, através do mesmo protocolo RFCOMM, pode ser compreendido através do trecho na Figura 4.7.

Como pode ser observado o processo é muito simples e, inclusive, também muito semelhante ao utilizado nos *sockets* TCP. O envio de informações baseia-se apenas em abrir um

```
1 public class Btpcap {
2
3         private ConfigInfo cfgInfo;
4
5         private static final UUID AODV_SERVICE = new UUID(0x100A);
6
7         public synchronized void loopPacket(ConfigInfo cfg, BtpcapHandler handler){
8
9                 cfgInfo = cfg;
10
11                try {
12
13                        LocalDevice.getLocalDevice().setDiscoverable(DiscoveryAgent.GIAC);
14
15                        String url = "btspp://localhost:" + AODV_SERVICE.toString()
16                                + ";name=AODV_Protocol";
17
18                        StreamConnectionNotifier service =
19                                (StreamConnectionNotifier) Connector.open(url);
20
21                        System.out.println("Abriu conexao");
22
23                        while(true){
24                                StreamConnection con =
25                                        (StreamConnection) service.acceptAndOpen();
26
27                                System.out.println("Conexão Efetuada");
28
29                                BtListener btListener = new BtListener(con, cfgInfo, handler);
30                                btListener.start();
31                        }
32                } catch (IOException ex) {
33                        System.out.println("Btpcap - Impossivel iniciar serviço AODV");
34                }
35        }
36 }
```

Figura 4.6: Disponibilizando um serviço

```
1 con = (StreamConnection)
2         Connector.open(urlServidor);
3
4 os = con.openOutputStream();
5
6 os.write(aodvDgram.toString().getBytes());
```

Figura 4.7: Envio com o protocolo RFCOMM

canal de comunicação com o servidor desejado[3] e no envio propriamente dito através da utilização do método write(). A informação é recebida, então, pelo método read() no lado servidor e devidamente processada com base nas necessidades do usuário.

[3]Quando a busca por serviços é efetuada em um dispositivo, o dispositivo local pode armazenar informações como a URL do serviço desejado

Capítulo 5

Redes Ad-Hoc

5.1 Características Gerais das Redes Sem Fio Tradicionais

Nos últimos anos tem sido possível perceber que a proliferação da utilização das redes sem fio vem acontecendo de forma impressionante. Atualmente a quantidade de serviços disponíveis e usuários dos sistemas que baseiam-se neste conceito cresce consideravelmente. Segundo Anjum e Mouchtaris [2], hoje são registrados mais de um bilhão de assinantes de serviços *wireless* (sem fio) tanto para comunicação por voz, quanto para transferência de dados ao redor do mundo.

A busca pela mobilidade, sem dúvida, é uma forte impulsionadora deste crescimento. No escritório ou até mesmo dentro de casa, a praticidade que a ausência de cabos pode proporcionar é algo fascinante. A possibilidade de utilizar serviços de telefonia ou consultar seus *e-mails* sem estar preso a um determinado local torna o dia-a-dia muito mais dinâmico.

Ainda segundo Anjum e Mouchtaris [2], existem razões bem definidas para essa grande aceitação e proliferação da tecnologia *wireless*. Estas razões estão voltadas para um assunto que afeta a todos, sem exceção: o custo. Os equipamentos pertencentes a tecnologia sem fio, assim como a sua instalação, possuem custos significantemente mais baixos que o verificado nos sistemas cabeados. Outro ponto importante é que os sistemas sem fio possibilitam a disponibilização de serviços de voz e dados sobre a mesma rede e com alguma mobilidade, tornando muito atraente o fato de poder utilizar estes serviços a qualquer momento e de qualquer lugar.

A estrutura básica de uma rede sem fio apresentada por Kurose e Ross [17] pode ser visualizada na Figura 5.1. Os principais elementos são:

- **Hospedeiros sem fio**: Da mesma forma que nas redes tradicionais, os hospedeiros são

os equipamentos pertencentes a borda do sistema, ou seja, são aqueles que executam as aplicações. Exemplos de hospedeiros são PCs, PDAs, telefones celulares, dentre outros. Eles podem, ou não, ser dispositivos móveis;

- **Estações-base**: Diferente dos hospedeiros, as estações-base não possuem uma analogia com as redes tradicionais. Elas são responsáveis por toda comunicação efetuada de e para qualquer hospedeiro que a ela esteja associado. Ao afirmar que um hospedeiro está associado a determinada estação-base significa que este está dentro da área de cobertura desta estação e a utiliza para a transmissão de dados entre ele e a rede maior. Exemplos de estações-base são as torres de telefonia celular e os pontos de acesso em uma LAN 802.11 [1];

- **Enlaces sem fio**: Os hospedeiros efetuam comunicações entre eles e com as estações-base através dos enlaces de comunicação sem fio. São os enlaces que determinam as taxas de transferência e a distância que um sinal pode alcançar, dependendo da tecnologia de enlace utilizada;

- **Infra-estrutura de rede**: Representa a rede maior com a qual um hospedeiro pode querer efetuar comunicação.

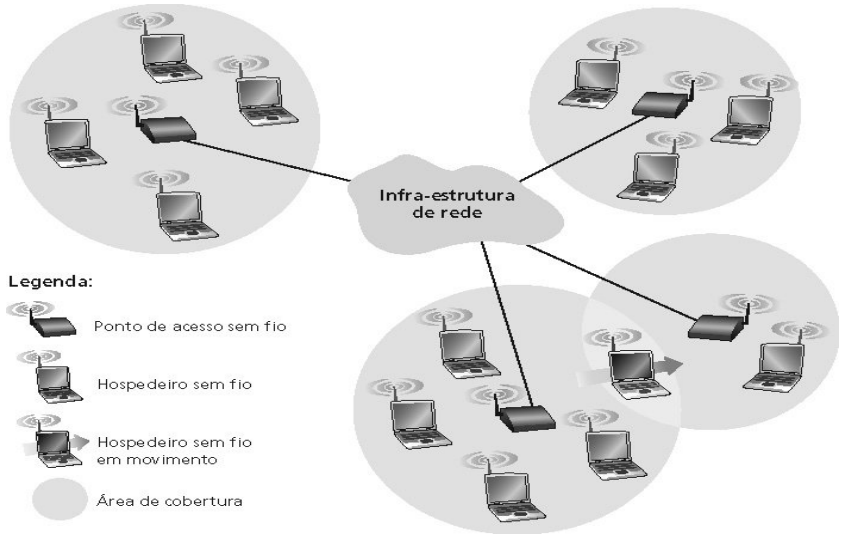


Figura 5.1: Elementos de uma rede sem fio

[1] Padrão da IEEE para redes sem fio locais

5.2 Características das Redes Ad-Hoc

A forma de operação das redes sem fio tradicionais é conhecida como infra-estruturada, na qual a estação-base é quem fornece tal infra-estrutura.

Tecendo um comparativo entre uma rede sem fio tradicional e uma Ad-Hoc, a principal e fundamental diferença está exatamente na infra-estrutura. Quando se trata de redes Ad-Hoc, essa infra-estrutura deixa de existir (Figura 5.2 [2]), fazendo com que os próprios hospedeiros fiquem responsáveis pela atribuição e tradução de endereços, roteamento, dentre os outros vários serviços necessários para o funcionamento de uma rede [17].

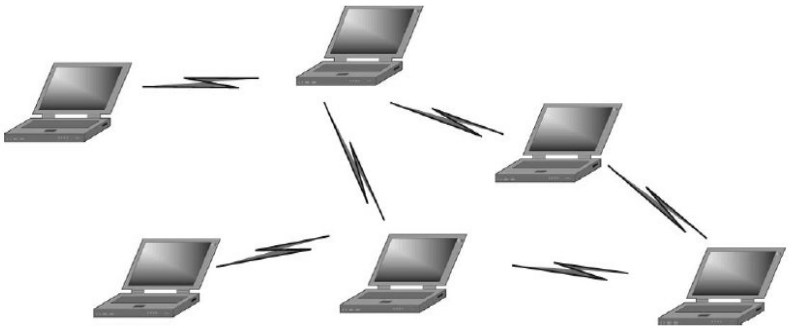


Figura 5.2: Exemplo de Rede Ad-Hoc

Nessa nova arquitetura os hospedeiros, não possuindo os pontos de acesso para auxiliar na comunicação, necessitam fazê-la através de seus vizinhos. Este conceito nos traz, visivelmente, a possibilidade de total mobilidade dos dispositivos, uma vez que estes já não ficam presos a área de cobertura de determinada estação-base. Para que se possa efetuar comunicação com outros dispositivos que não estão diretamente no alcance da fonte, o uso dos protocolos de roteamento Ad-Hoc permitem que uma busca seja efetuada através dos próprios hospedeiros para localização e definição de uma rota, permitindo, posteriormente, a comunicação. As redes que suportam essa arquitetura são também conhecidas como Redes Móveis Ad-Hoc (MANET - *Mobile Ad-Hoc Networks*) [2].

As MANETs podem estar presentes em diversos nichos de aplicação. Os primeiros esforços foram desenvolvidos pelo departamento de defesa norte americano (DoD). Devido as dificuldades de implantar comunicação nos campos de batalha (pela existência de obstáculos, distância

entre os nós e vários outros fatores), a utilização de uma rede estruturada apresentava-se totalmente inviável. Por esse motivo, o DoD passou a incentivar inúmeras pesquisas na área, o que culminou no desenvolvimento de vários artigos que motivaram a criação de outras aplicações.

Uma outra aplicação que atualmente está sendo muito difundida é a rede de sensores sem fio. Nessas redes, cada nó é equipado com diversos tipos de sensores, tais como acústico, de temperatura e pressão. Esses nós podem ser organizados em grupos, onde são utilizados para detecção de determinados eventos no local. Segundo Loureiro et al. [20], a rede de sensores pode ser considerada como uma classe especial de MANET, pois, além da comunicação entre os nós, ela deve propiciar um grau elevado de cooperação para alcançar os objetivos de sua instalação. Outro exemplo de extrema importância apresentado por Anjum e Mouchtaris [2] é a possibilidade de manter comunicação em casos de grandes desastres e situações de emergência, pois, normalmente, a infra-estrutura de comunicação nos locais onde o problema ocorreu é danificada, reduzindo ou, até mesmo, acabando com qualquer possibilidade de solicitar algum tipo de auxílio.

5.3 Roteamento

Ao abordar o tema de estratégias de roteamento para redes Ad-Hoc, estamos falando de um desafio a parte. Os protocolos tradicionais tornam-se ineficientes neste caso, pois a característica principal do modelo é permitir que os dispositivos possam estar em constante movimentação sem que percam conectividade. Desse modo a busca por soluções mais eficientes tem sido objeto de estudo já a algum tempo.

No campo das redes Ad-Hoc, os protocolos são divididos em dois grupos: os **proativos** e os **reativos**. Os protocolos proativos são aqueles que detém uma ou mais tabelas de roteamento em cada dispositivo, nas quais são armazenadas as rotas já consultadas. Essas rotas são constantemente atualizadas. Sempre que um dispositivo encontra alguma alteração, ele propaga essa atualização para os demais, mantendo as tabelas de roteamento sempre consistentes [13]. Por outro lado, os reativos são caracterizados por protocolos sob demanda, ou seja, eles não mantém armazenadas diversas rotas sofrendo constante alteração, essas são formadas apenas no momento em que são necessárias e posteriormente armazenadas. Verificações periódicas são efetuadas para certificar de que aquelas rotas ainda permanecem válidas, caso contrário são

excluídas da tabela [13]. Existem também os protocolos ditos **híbridos**. Estes possuem características de ambas abordagens, podendo utilizar estratégias proativas para comunicação entre as redes e reativas no“intra-rede”, por exemplo.

Para os fins do trabalho, verificou-se que a utilização do conceito de proatividade não seria muito interessante. Devido às propriedades dos dispositivos-alvo, armazenar n tabelas de roteamento poderia gerar consumo desnecessário dos recursos naturalmente limitados. Portanto, a partir daqui, estarão sendo tratados apenas os protocolos reativos, em especial o AODV [30]. Um fator importante da proposta está centrado na adaptação do protocolo para o funcionamento sobre o Bluetooth, por esse motivo, na próxima seção, apresentamos uma breve discussão sobre as implicações dessa adaptação.

5.4 Roteamento Ad-Hoc Sobre a Tecnologia Bluetooth

Em *Lunar over bluetooth* [34] o autor apresenta o processo de adaptação de um protocolo de roteamento para ser utilizado com a tecnologia Bluetooth. O protocolo escolhido foi o LUNAR (*Lightweight Underlay Network Ad-Hoc Routing*) [29]. Este é um protocolo extremamente simples, desenvolvido com o objetivo de suportar um cenário de dez a dezesseis nós. LUNAR é considerado um protocolo híbrido, já que novas rotas são descobertas apenas no momento da necessidade (sob-demanda). Quando estas são criadas, permanecem armazenadas nas tabelas de roteamento, onde entra a proatividade. A partir do momento em que a rota está armazenada, constantes verificações de validade são efetuadas e, segundo a necessidade, informações de atualização são propagadas para os demais dispositivos.

No LUNAR o autor coloca três pontos cruciais observados na adaptação. Estas observações são importantes por pertencerem exclusivamente a forma de funcionamento do Bluetooth, relevando suas características ligadas a formação de redes.

5.4.1 Descoberta de novos vizinhos

Diferente do padrão 802.11 que pode utilizar *broadcast* para tal verificação, o Bluetooth necessita efetuar constantes execuções de sua operação de descoberta. Quando em estado de busca, as demais operações do rádio são interrompidas, o que compromete a propagação de dados, tendo em vista que, se um dispositivo pertencente a determinada rota entrar em processo de

busca, todo fluxo de dados que necessita dele será temporariamente interrompido. Se o dispositivo em questão for constantemente requisitado, esses bloqueios necessários para o processo de busca pode tornar o desempenho da rede muito baixo, sendo gasto um tempo consideravelmente alto para propagação de pequenos pacotes, por exemplo.

5.4.2 Formação das Piconets e Scatternets

A dificuldade desta questão está em como fazê-la de forma eficiente. Existem diversas propostas de protocolos para a formação das Scatternets, no entanto, são poucas as implementações e, muitas vezes, a questão da eficiência não é colocada de forma muito clara. Devido a essa problemática de localização de fontes para basear o desenvolvimento, o autor optou por uma estratégia simples que não visa exatamente a eficiência: "localizar e conectar".

5.4.3 Roteamento entre e dentro das Piconets

Como colocado por Rensfelt [34], a especificação do Bluetooth peca um pouco com relação às informações da forma como o tráfego deve ser transmitido entre os nós. Uma solução viável é a utilização do conceito de IP, o qual pode ser facilmente mapeado para o endereço de hardware do Bluetooth, muito semelhante aos endereços MAC comumente utilizados nas LANs, não apresentando muita complicação no desenvolvimento.

Capítulo 6

Ad-Hoc On-demand Distance Vector (AODV)

O objetivo do capítulo é descrever com maiores detalhes o funcionamento do protocolo AODV, baseando-se na RFC 3561 [30].

O algoritmo do AODV permite um roteamento dinâmico e de múltiplos saltos entre todos os nós participantes da rede Ad-Hoc. Desenvolvido para suportar de dezenas a milhares de dispositivos, ele proporciona uma transmissão veloz de dados aos destinatários desejados, sem, para isso, precisar manter rotas armazenadas, com exceção das que estão atualmente em uso. Sendo um protocolo imune a ciclos, evita o problema de pacotes sendo propagados "infinitamente". Outra característica importante é a rápida resposta dos nós às quebras de rotas ou mudanças na topologia da rede.

Se comparado a outros protocolos utilizados nas redes tradicionais, AODV apresenta três tipos especiais de mensagens, chamadas *Route Requests* (RREQs), *Route Replies* (RREPs) e *Route Errors* (RERRs). Se um nó fonte já possui uma ou mais rotas armazenadas para o destino, então o AODV atua de forma passiva, no entanto, caso a rota necessária ainda não exista, o protocolo torna-se ativo. Ele dispara, por *broadcast*, mensagens do tipo RREQ para encontrar uma rota válida. Existem duas formas para chegar a uma rota válida. A primeira é quando a mensagem atinge o destino e a segunda é quando um nó intermediário já possui a rota armazenada, não sendo necessário continuar a propagação. Para ativar a rota encontrada, uma mensagem do tipo RREP é enviada de volta ao nó de origem através de *unicast*. Cada nó na rede, ao receber uma mensagem RREQ, armazena em sua tabela de roteamento o caminho de volta ao nó de origem, possibilitando, assim, o envio da mensagem de resposta RREP.

Na existência de uma rota ativa, todo nó sempre observa o status da ligação com o nó referente ao próximo salto. Caso ocorra uma quebra nessa rota, a mensagem RERR é enviada para informar todos os outros nós que ocorreu uma quebra de ligação. Para possibilitar o uso deste mecanismo, cada nó possui uma lista de precursores (*precursor list*), que contém o endereço IP de todos os vizinhos que, eventualmente, poderiam usar aquele nó com status negativo como próximo salto para um destino específico.

6.1 Tabela de Roteamento

Como o AODV é um protocolo de roteamento, é natural que possua uma tabela para registrar suas rotas. Existe a necessidade de armazenar, mesmo que temporariamente, essas informações para que seja possível, por exemplo, saber o caminho de volta ao nó que originou uma mensagem de RREQ. Sua tabela de rotamento possui os seguintes campos, conforme Figura 6.1 [36]:

- *Destination IP Address* (Endereço IP de Destino);
- *Destination Sequence Number* (Número de Seqüência do Destino - DSN);
- *Valid DSN Flag* (Indicador de Validade do DSN);
- Outros estados e indicadores de roteamento (como válido, inválido, reparável, sendo reparado);
- *Network Interface* (Interface de Rede);
- *Hop Count* (Contagem de Saltos): número de saltos necessários para alcançar o destino;
- *Next Hop* (Próximo Salto): endereço do próximo nó na rota;
- *List of Precursors* (Lista de Precursores): endereços dos dispositivos vizinhos que poderão vir a utilizar a rota;
- *Lifetime* (Tempo de Vida): tempo para que a rota expire ou seja apagada.

Destination IP Address	DSN	Valid DSN flag	Other flags			Network Interface	Hop Count	Next Hop	List of Precursors	Lifetime
123.456.789.012	12	0	0	1	0	loO	5	456.789.012.345	...	4
321.654.987.210	45	1	1	0	1	leO	9	456.789.012.345	...	6
456.789.012.345	13	0	1	0	0	loO	3	123.456.789.012	...	7

Figura 6.1: Exemplo de uma tabela de roteamento

6.2 Formato das Mensagens

Aqui estão dispostos os formatos das três principais mensagens do protocolo e, também, da mensagem especial RREP-ACK (mensagem de reconhecimento do RREP), incluindo o significado de cada um dos seus campos . As figuras utilizadas nesta sessão foram retiradas de *AODV enhanced by Smart Antennas* [36].

6.2.1 RREQ

Type	J	R	G	D	U	Reserved	Hop Count
RREQ ID							
Destination IP Address							
Destination Sequence Number							
Originator IP Address							
Originator Sequence Number							

Figura 6.2: Formato da mensagem RREQ

- J – *Join flag* (Indicador de ligação): reservado para multicast;
- R – *Repair flag* (Indicador de reparo): reservado para multicast;
- G - *Gratuitous RREP flag* (Indicador de RREP desnecessário): indica se um RREP desnecessário será enviado por unicast ao destinatário especificado no campo Endereço IP de Destino;
- D – *Destination only flag* (Indicador de apenas o destinatário): indica que apenas o destinatário pode responder a esta RREQ;

- U - *Unknown sequence number* (Número de seqüência desconhecido): indica que o número de seqüência é desconhecido;
- *Reserved* (Reservado): se enviado como 0 é ignorado na recepção;
- *Hop Count* (Contagem de saltos): número de saltos a partir do nó fonte;
- *RREQ ID* (Identificador do RREQ): um identificador único associado a cada RREQ enviado;
- *Destination IP Address* (Endereço IP de destino): endereço de destino para o qual a rota é desejada;
- *Destination Sequence Number* (Número de seqüência do destino): o último número de seqüência recebido pela origem para qualquer rota para o destino;
- *Originator IP Address* (Endereço IP da origem): endereço IP do nó que originou a requisição da rota;
- *Originator Sequence Number* (Número de seqüência da origem): o número de seqüência atual a ser utilizado na entrada da rota que aponta para a origem.

6.2.2 RREP

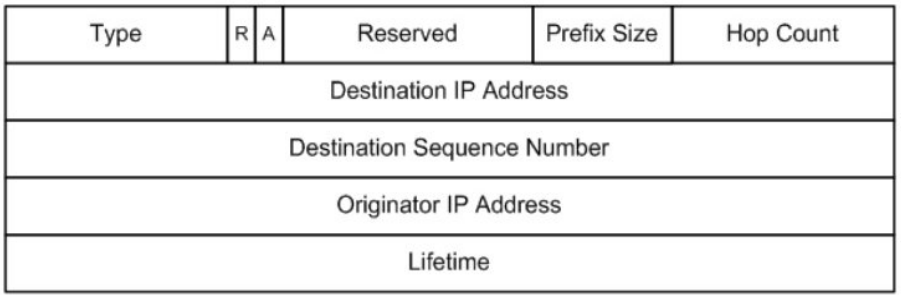


Figura 6.3: Formato da mensagem RREP

- R – *Repair flag* (Indicador de reparo): reservado para multicast;
- A – *Acknowledgment required* (Reconhecimento requerido): quando diferente de 0, indica que o receptor da mensagem RREP necessita enviar uma mensagem de RREP-ACK;

- *Reserved* (Reservado): se enviado como 0 é ignorado na recepção;
- *Prefix Size* (Tamanho do prefixo): se diferente de 0 especifica que o próximo salto indicado pode ser usado por qualquer nó com o mesmo prefixo de roteamento para o destino requisitado;
- *Hop Count* (Contagem de saltos): o número de saltos da origem até o destino;
- *Destination IP Address* (Endereço IP de destino): endereço IP do nó para o qual a rota foi solicitada;
- *Destination Sequence Number* (Número de seqüência do destino): o número de seqüência do destino associado a rota;
- *Originator IP Address* (Endereço IP da origem): endereço IP do nó que originou a mensagem RREQ para a rota em questão;
- *Lifetime* (Tempo de vida): tempo, em milissegundos, que a rota será considerada válida para o nó que recebeu a mensagem RREP.

6.2.3 RERR

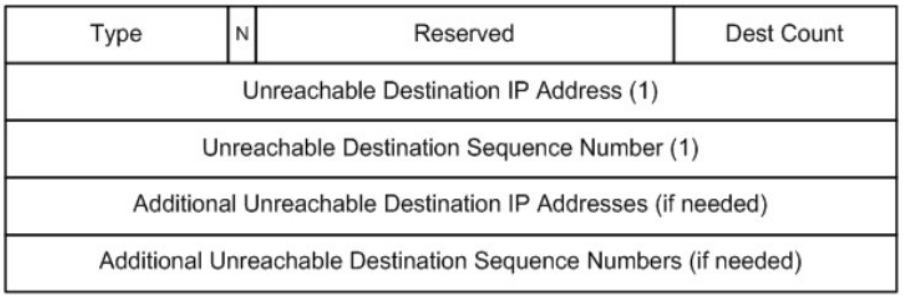


Figura 6.4: Formato da mensagem RERR

- N - *No delete flag* (Indicador para não-deleção): utilizado quando um nó está executando a manutenção de uma ligação, representando que os demais nós não devem apagar a rota;
- *Reserved* (Reservado): se enviado como 0 é ignorado na recepção;

- *Dest Count* (Contagem de destinos): número de destinos inalcançáveis; deve ser de pelo menos um;
- *Unreachable Destination IP Address* (Endereço IP do destino inalcançável): endereço do destino que se tornou inalcançável devido a uma quebra de ligação;
- *Unreachable Destination Sequence Number* (Número de seqüência do destino inalcançável): número de seqüência na tabela de roteamento para o destino informado no campo anterior.

6.2.4 RREP-ACK

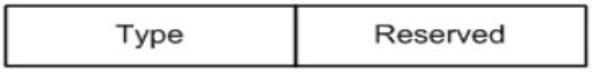


Figura 6.5: Formato da mensagem RREP-ACK

Este tipo de mensagem precisa ser enviada como resposta à RREP sempre que o indicador "A" estiver sendo utilizado. Esta ação é normalmente executada quando há risco de existir uma ligação unidirecional impedindo a conclusão do ciclo de descoberta de rota. Possui apenas um campo a destacar:

- *Reserved* (Reservado): se enviado como 0 é ignorado na recepção.

6.3 Exemplo de Funcionamento do Protocolo

O objetivo desta seção é tornar claro a forma com a qual as rotas são formadas através do envio de mensagens. Para tal, será utilizado um exemplo exposto por Solheid [36].

Neste exemplo, a rede consiste em quatorze nós, onde o nó de número 1 é a origem e o 14 o destino, como mostra a Figura 6.6. A formação da rota se dá como se segue:

6.3.1 *Route Request* – RREQ

O processo se inicia com o nó de origem (nó 1) enviando um RREQ para cada um dos seus vizinhos. Para isso, ele incrementa o número de seqüência e cria a mensagem de requisição com o endereço de destino do nó 14 (Figura 6.7).

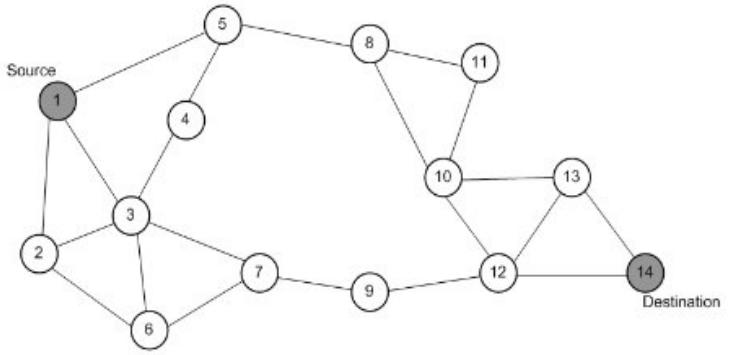


Figura 6.6: Rede utilizada no exemplo

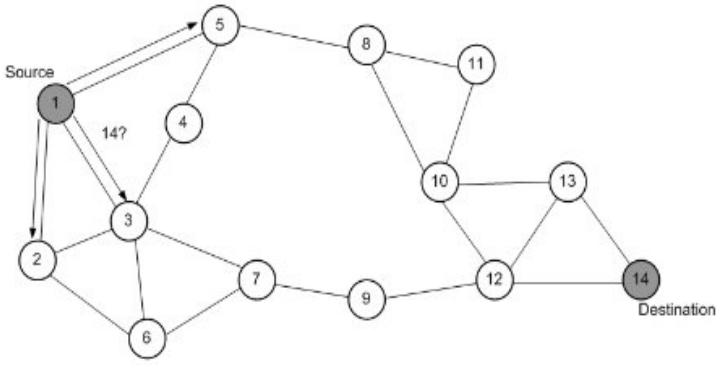


Figura 6.7: Inicio do processo de criação da rota

Ao receber a mensagem, porém, antes de propagá-la, os nós 2, 3 e 5 definem a rota de retorno ao nó de origem e atualizam suas tabelas de roteamento. Ao receber a requisição, o número de seqüência atual armazenado no nó é comparado ao DSN (Destination Sequence Number) da mensagem RREQ. Se o número de seqüência for maior, a mensagem deve ser descartada. Do contrário, o número de seqüência do nó deve ser atualizado com o DSN (Figura 6.8).

Agora, os nós 2, 3 e 5 repassam a requisição a seus vizinhos, lembrando que a contagem de saltos deve ser atualizada. Um problema verificado no *broadcast* é a possibilidade de existir replicação de mensagens para a origem. Diante disso, o algoritmo do protocolo propõe o seguinte mecanismo: quando um nó recebe novamente uma mensagem, ele simplesmente a descarta. Assim, a propriedade de ausência de ciclos é garantida (Figura 6.9).

O mesmo ocorre com os nós 4, 6, 7 e 8. Eles recebem a mensagem, definem a rota de retorno à origem e dão seqüência a propagação (Figura 6.10).

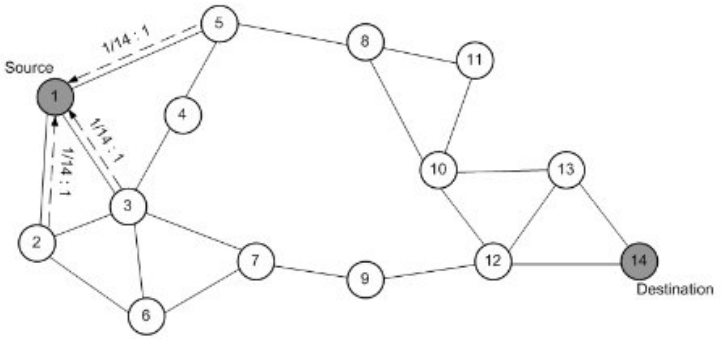


Figura 6.8: Definindo o rota de retorno ao nó de origem

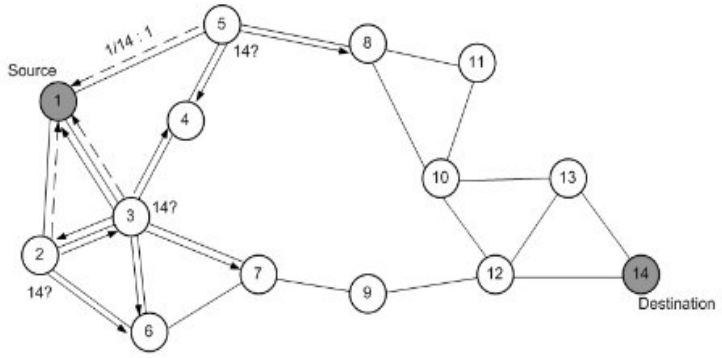


Figura 6.9: Nós 2, 3 e 5 propagando a requisição

A propagação continua avançando até encontrar o nó de destino.

Finalmente o nó 14 é alcançado e, como podemos ver, com duas rotas encontradas (Figura 6.12).

6.3.2 *Route Reply* – RREP

Assim que o nó 14 recebe a mensagem de requisição, responde com uma RREP para o nó 1 (Figura 6.13).

Antes de enviar a resposta, o nó 14 incrementa o número de seqüência e, também, seu DSN. A RREP é sempre enviada na forma de *unicast*, uma vez que todos os nós da rota já conhecem os "próximos saltos" necessários. Como pode ocorrer que mais de uma rota seja encontrada, o algoritmo fica responsável por escolher a mais curta, verificando, para isso, o número de saltos efetuados em cada uma.

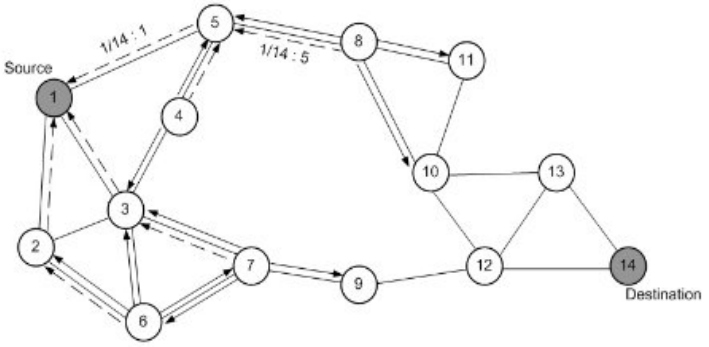


Figura 6.10: Nós 4, 6, 7 e 8 definindo a rota de retorno à fonte e repassando a requisição

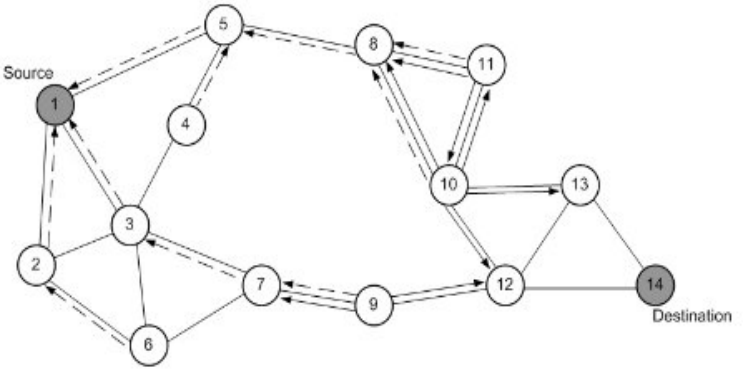


Figura 6.11: Rede após os nós 12 e 13 receberem a requisição

Ao receber a RREP, o nó 13 atualiza novamente o DSN e sua tabela roteamento, inserindo agora a rota utilizada para o destino, ou seja, para o nó 14, repassando-a posteriormente (Figura 6.14).

A mensagem de resposta normalmente é muito rápida, já que a utilização de *unicast* não costuma encontrar os mesmo problemas que o *broadcast*, como as colisões.

Finalmente a RREP alcança o nó de origem. Quando o nó um recebe a mensagem ele adota automaticamente seu DSN.

6.3.3 *Route Error* – RERR

Vamos supor que depois da rota formada, ocorreu um erro entre os nós 8 e 10 (Figura 6.17). O nó 8 percebe essa quebra de ligação porque, ao tentar enviar algum pacote ao nó 10, ocorre uma falha.

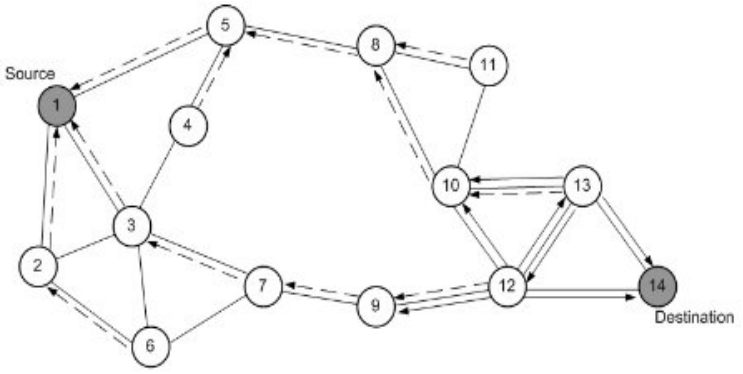


Figura 6.12: RREQ chega até ao nó de destino

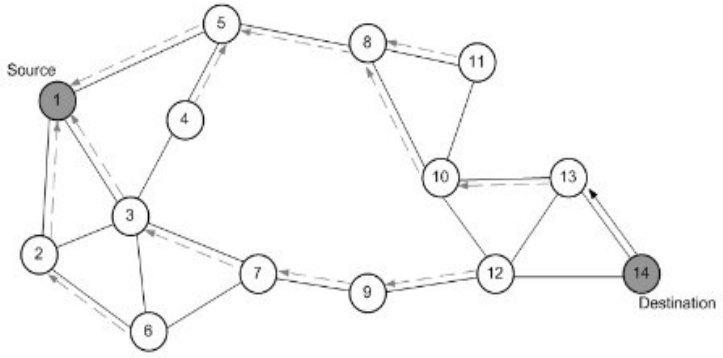


Figura 6.13: Início do envio da mensagem de resposta à origem

Como resultado da falha o nó oito incrementa seu número de seqüência e gera uma mensagem RERR, a qual precisa ser enviada ao nó fonte da rota (que neste caso é o nó 1). Ao receber essa mensagem de erro a origem é obrigada a iniciar um novo processo de descoberta de rota. Note que, após os passos descritos acima, quando a mensagem RREQ alcançar o nó 10 ou o 13, estes já podem responder ao nó de origem com uma RREP, tendo em vista que ambos já possuem uma rota válida para 14.

Com este simples exemplo já é possível ter uma noção do funcionamento básico do protocolo AODV. Diante da mobilidade dos aparelhos celulares pode-se perceber que, a partir da adaptação do protocolo ao Bluetooth, as ferramentas necessárias na comunicação e propagação de pacotes entre os dispositivos estarão disponíveis para nosso uso, proporcionando um ambiente propício à criação da ubiqüidade.

A característica do Bluetooth de não fornecer nativamente a possibilidade de efetuar

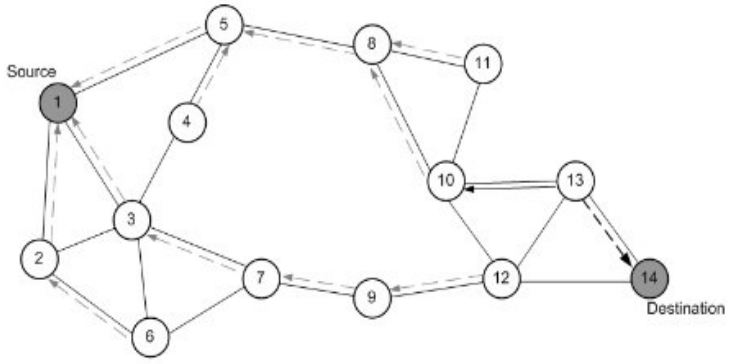


Figura 6.14: Nó 13 definindo a rota ao destino e repassando a mensagem de resposta

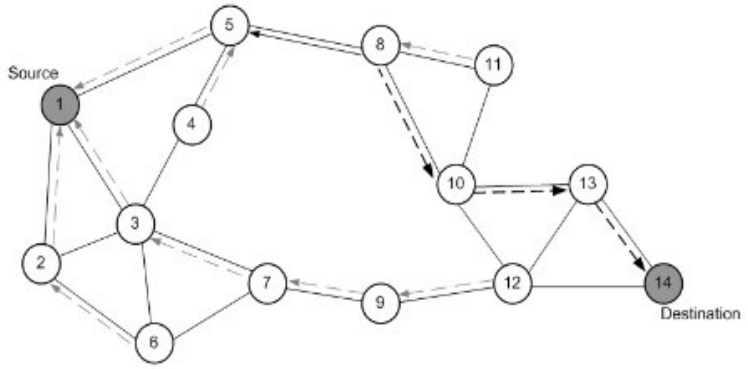


Figura 6.15: Continuação da propagação da RREP

`broadcast` aparece aqui como uma limitação. Neste caso, como mencionado na Seção 4.4.1, serão necessárias várias execuções de busca por dispositivos vizinhos para, aí sim, conseguir enviar as diversas mensagens. No entanto, tratando esta questão, teremos os atributos suficientes para alcançar nossos objetivos.

6.4 AODV Sobre o Bluetooth

Para realização deste trabalho foi utilizada uma implementação do protocolo AODV denominada J-Adhoc [37] como base para a adaptação. J-Adhoc é distribuída sob licença GPL (GNU General Public License), sendo escrita em Java e baseada na RFC- 3561 [30].

Uma vez que o objetivo principal do trabalho é a implantação do protocolo junto a aparelhos celulares, o J-Adhoc foi utilizado como base para implementação na tecnologia J2ME. No entanto, toda parte de comunicação utilizada pelo J-Adhoc necessitou ser alterada para que fosse

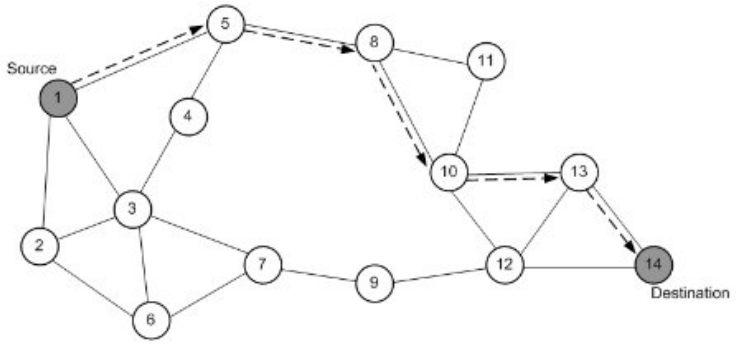


Figura 6.16: Definição de rota concluída com RREP alcançando a origem

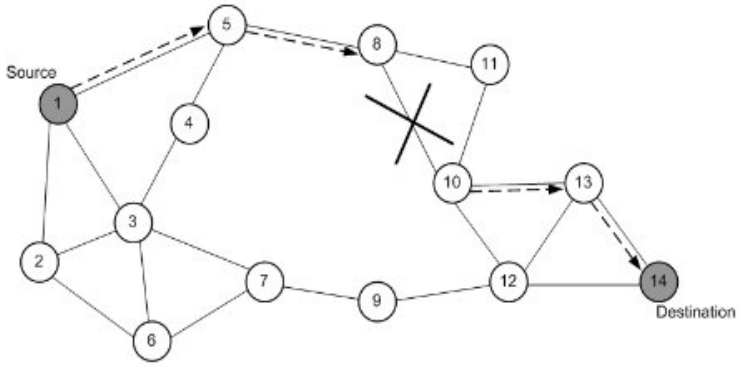


Figura 6.17: Exemplo do surgimento de um erro entre os nós 8 e 10

possível a utilização do Bluetooth como "interface de rede", gerando uma nova implementação chamada BtJAdhoc (Bluetooth Java Ad-Hoc). O diagrama de classes mostrado na Figura 6.18 apresenta uma visão geral da implementação do BtJAdhoc. De uma forma geral, a estruturação referente a organização das classes nos pacotes foi mantida, as alterações principais ocorreram nas classes responsáveis pelo processo de envio e recebimento de pacotes, as quais estão implementadas no pacote Btpcap (vide Figura 6.19).

6.4.1 Adaptação

Buscando não modificar as características básicas do algoritmo do protocolo AODV, as alterações aplicadas foram direcionadas principalmente às questões de comunicação, deixando o gerenciamento interno praticamente inalterado.

Começando a discussão pelo monitoramento do tráfego da rede, J-Adhoc utiliza a biblioteca

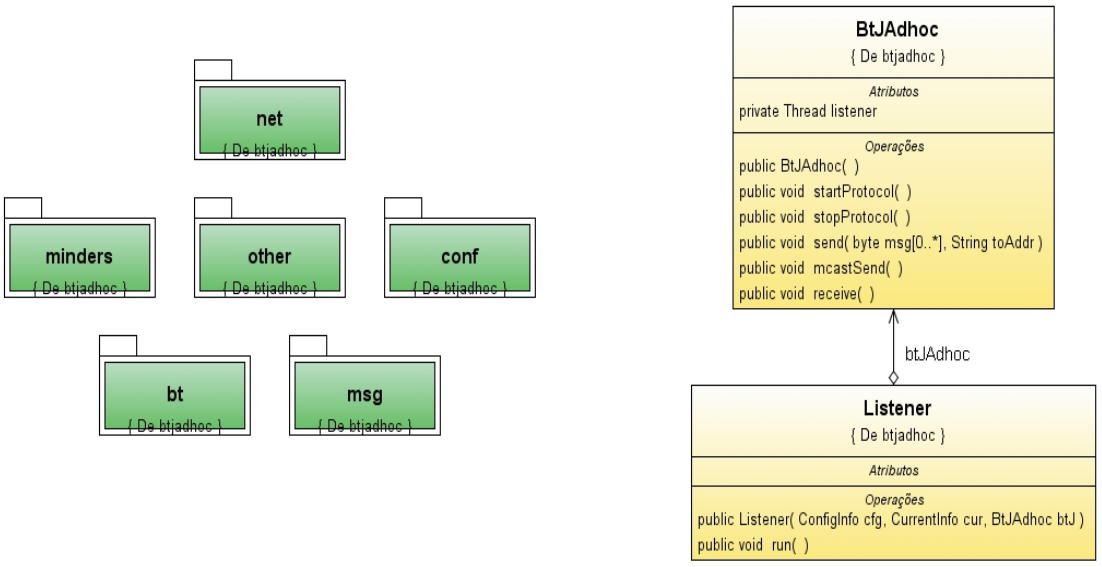


Figura 6.18: Diagrama de Classes do Pacote BtJAdhoc

Java Jpcap [10] (também regida pela licença GPL), a qual permite às aplicações enviar e capturar pacotes através das interfaces de rede, manipulando-os posteriormente em Java. Essa foi a parte que mais sofreu alterações. Uma nova biblioteca com as funcionalidades do Jpcap foi desenvolvida para o Bluetooth. Esta nova biblioteca chamada de Btpcap manteve apenas algumas poucas características da implementação original. Através do diagrama de classes da Figura 6.19 podemos visualizar a organização desta nova biblioteca. A biblioteca Jpcap foi desenvolvida para suportar a captura e envio de pacotes dos tipos Ethernet, IPv4, IPv6, ARP/RARP, TCP, UDP e ICMPv4. Para a implementação no Bluetooth, toda essa variedade de pacotes pode ser descartada, sendo mantidas apenas algumas informações importantes para o algoritmo, como endereço de origem, endereço de destino, tamanho do pacote, dentre outras, com as quais foi possível criar uma nova gama de tipos de pacotes, sendo eles:

- **BtAODVDgram**: para criação deste tipo de pacote foram utilizadas algumas informações existentes nos datagramas UDP. Através desses pacotes é que nossa implementação do protocolo envia as mensagens de controle do AODV;
- **BtPacket**: semelhante ao antigo IPPacket existente na implementação da classe Jpcap. Com o auxílio deste pacote é que o Btpcap manipula os dados que devem ser transmitidos entre os hospedeiros, ou seja, as informações propriamente ditas.

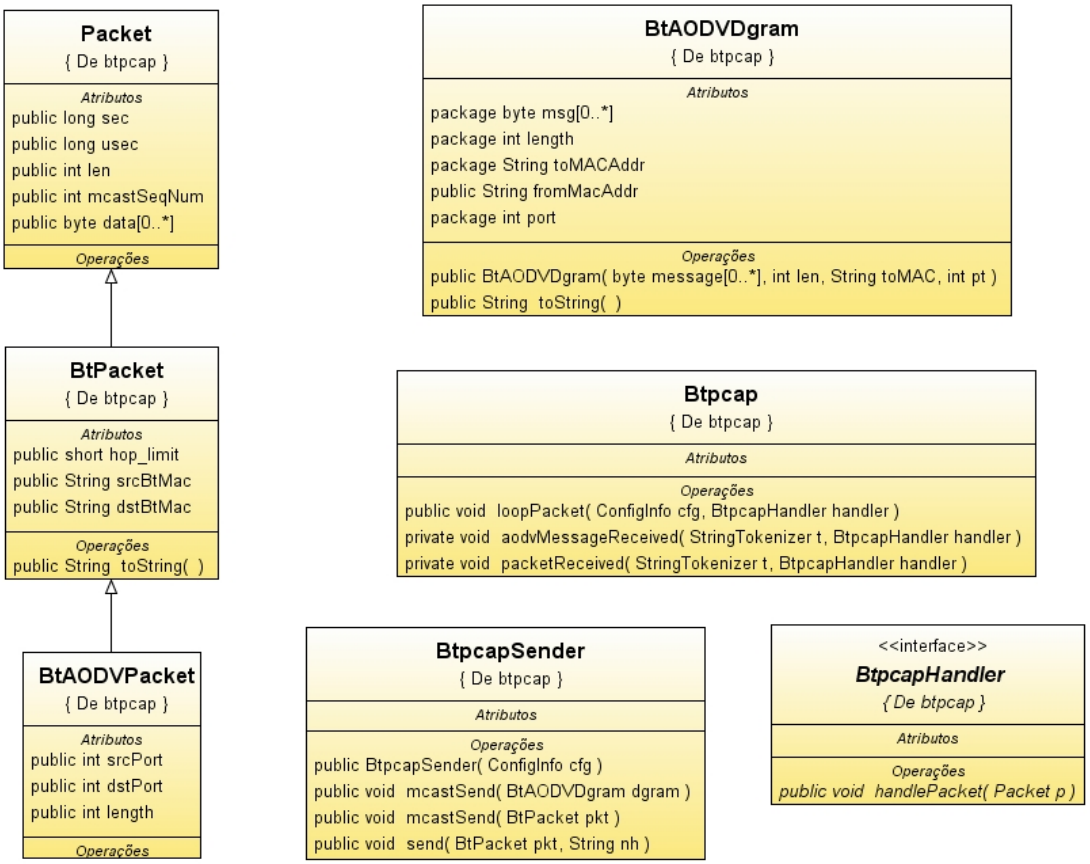


Figura 6.19: Diagrama de Classes da Biblioteca Btpcap

Outro ponto importante alterado na biblioteca original foi a forma utilizada para captura dos pacotes propriamente dita. Antes, tal tarefa era efetuada através de chamadas a funções do sistema operacional, necessitando do acesso à interface de rede. Para utilização do Bluetooth, esta estratégia também teve que ser totalmente repensada.

No Bluetooth serviços podem ser "registrados" e disponibilizados para que os demais dispositivos no alcance possam utilizá-los de alguma forma. Sendo assim, a classe responsável pela captura de pacotes (Btpcap) nada mais é do que um servidor. Quando o protocolo entra em funcionamento um serviço é disponibilizado (AODV-Protocol) e um canal de comunicação é aberto. O diagrama de seqüência da Figura 6.20 descreve o comportamento do protocolo ao receber um novo pacote.

A classe Btpcap possui o método loopPacket, o qual é responsável por ficar monitorando o canal de comunicação criado pelo serviço a espera de novas conexões e o conseqüente rece-

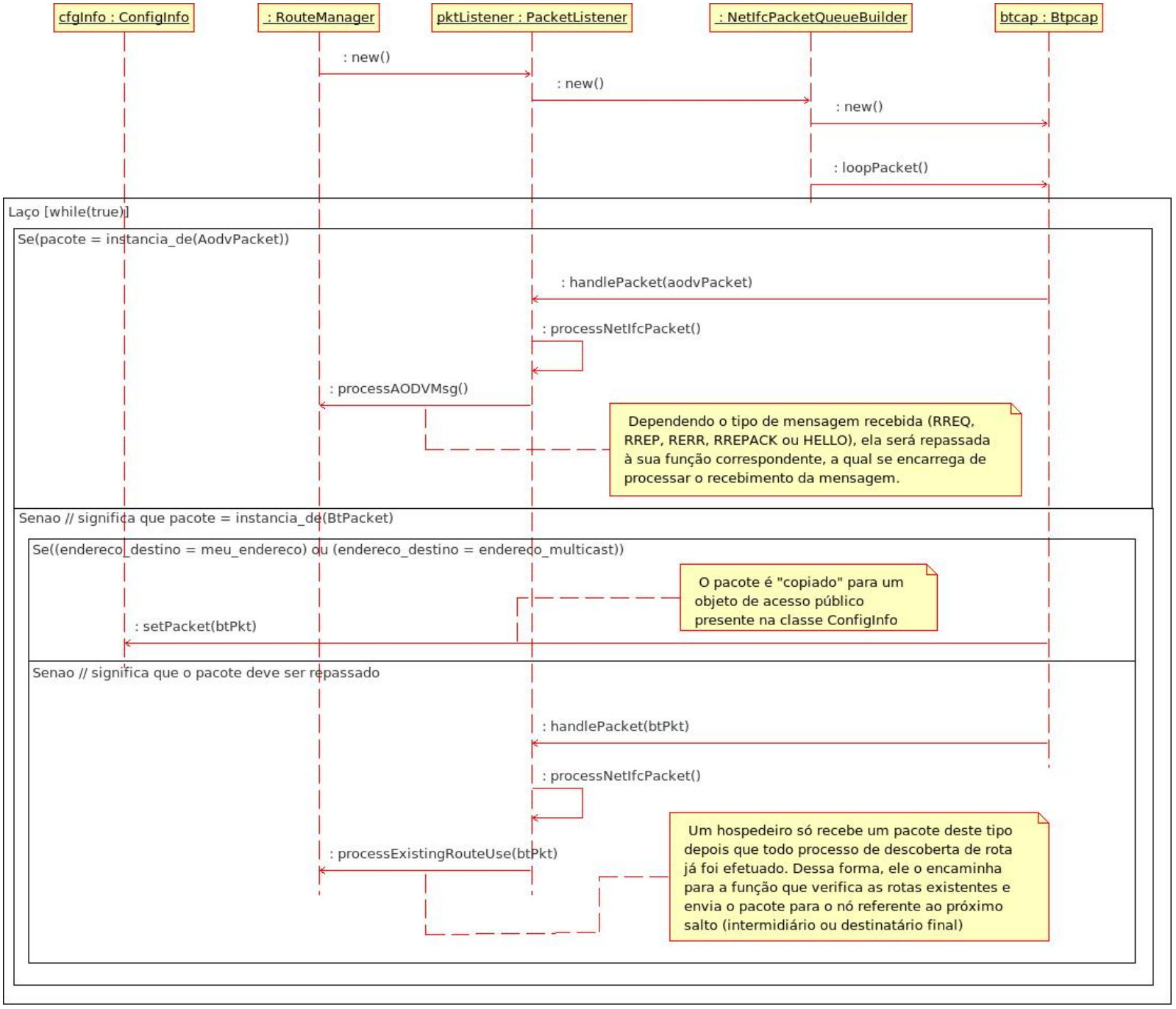


Figura 6.20: Diagrama de Seqüência do Recebimento de Pacotes

bimento de pacotes. Sempre que um dispositivo conecta-se ao serviço e envia algum pacote, a classe verifica qual o tipo de pacote recebido. Caso seja um pacote de controle AODV, o mesmo será repassado para a classe PacketListener, sendo processado (para verificar o tipo de mensagem, como RREQ ou RERR, ou questões relativas ao endereçamento) e posteriormente ficando sob responsabilidade da classe RouteManager, a qual se encarrega da interpretação e do processamento final do pacote.

No entanto, caso o pacote seja uma instância da classe BtPacket, existem duas possibilidades para o seu tratamento. Na primeira possibilidade, o pacote possui, como destino, o endereço do dispositivo local ou de *multicast*[1]. Quando esta alternativa se verifica o pacote é "copiado" para um objeto de acesso público presente na classe ConfigInfo. A escolha desta classe como hospedeira do pacote recebido foi devido a ela estar visível por, praticamente, todas as demais classes existentes no protocolo, facilitando um possível acesso aos pacotes recebidos. A segunda possibilidade é o endereço de destino ser diferente do endereço local ou *multicast*, significando que o dispositivo local está servindo como ponte para a propagação do pacote. Desse modo, o pacote, a exemplo das mensagens de controle, é repassado para a classe PacketListener, a qual efetuará um pré-processamento, e, na seqüência repassado para a classe RouteManager. Esta, através do seu método processExistingRouteUse, consegue verificar exatamente "para quem" o pacote deve ser encaminhado, dando seqüência ao processo de propagação.

Com relação ao envio, a estratégia também teve que ser totalmente revista. Devido ao fato do Bluetooth não implementar *broadcast*, um grande gargalo é gerado no momento do envio das mensagens de controle ou, até mesmo, de outro tipo de pacote com necessidade de propagação *multicast*.

A estratégia adotada foi realizar uma descoberta para determinar quais são os dispositivos que estão no seu alcance. O processo de descoberta de vizinhos do Bluetooth verificado durante os testes foi de cerca de dez segundos. Procurando reduzir o *overhead* gerado por esse processo, optamos por manter armazenado durante um determinado período de tempo os dispositivos encontrados na última busca, ao invés de efetuar uma nova busca a cada envio. Após

[1] O "pseudo-*broadcas*t" implementado para o Bluetooth trabalha, na realidade, sob a forma de *multicast*. Um endereçamento foi criado para o grupo, evitando que pacotes sejam propagados para outros dispositivos que não possuem interesse no recebimento. Mesmo existindo a necessidade de possuir o serviço AODV-Protocol sendo executado no dispositivo, a idéia de um endereçamento para o grupo foi mantida pensando no caso de algo semelhante tornar-se nativo ao Bluetooth.

os dispositivos no alcance terem sido encontrados, uma verificação é efetuada em cada um para manter armazenado apenas os que possuem o serviço do protocolo AODV executando.

A Figura 6.21 apresenta o diagrama de seqüência referente ao envio de mensagens. Quando este é do tipo *multicast*, desconsiderando os pacotes de controle, existe um incremento no número de seqüência de *multicast*, utilizado para que os dispositivos descartem pacotes já recebidos. Após o incremento o pacote é gerado na classe BtJAdhoc e repassado para o método mcastSend do objeto pktSender pertencente a classe RouteManager. Este método, por sua vez, repassa o pacote para a classe BtpcapSender (da biblioteca Btpcap) a qual utiliza o Bluetooth para concluir o processo de envio.

O envio do tipo *unicast* diferencia-se do *multicast* pelo fato da necessidade da descoberta de rota. Sendo um protocolo sob-demanda, a rota é gerada apenas no momento da necessidade de sua utilização. Dessa forma o pacote gerado na classe BtJAdhoc é repassado para a classe RouteManager, onde é verificado se já existe uma rota válida para o nó destino ou se será necessária sua criação. No caso da necessidade de descoberta de uma rota, mensagens RREQ são propagadas e o protocolo fica aguardando a chegada de uma resposta do tipo RREP. Quando isto ocorre, o pacote pode, finalmente, ser enviado ao destino com o auxílio do método send da classe BtpcapSender.

Em relação ao núcleo do protocolo J-Adhoc, algumas pequenas alterações também foram necessárias. Uma delas está relacionada ao endereçamento. A implementação original trabalha com endereços IP, o qual poderia ser mantido, no entanto, optou-se por trabalhar diretamente com o endereço do Bluetooth (com as mesmas características dos endereços MAC das interfaces de rede tradicionais). Esse endereço é único em cada dispositivo, anulando a necessidade de associar um dado endereço a cada dispositivo.

Existem também funcionalidades que foram totalmente extintas, como a atualização da tabela de roteamento do sistema operacional, por exemplo, ou a propagação das mensagens de Hello, utilizadas para avisar os nós vizinhos e para o controle das quebras de ligação. Com a existência do processo de busca necessário em todos os dispositivos que utilizam Bluetooth, o envio do Hello não é mais necessário.

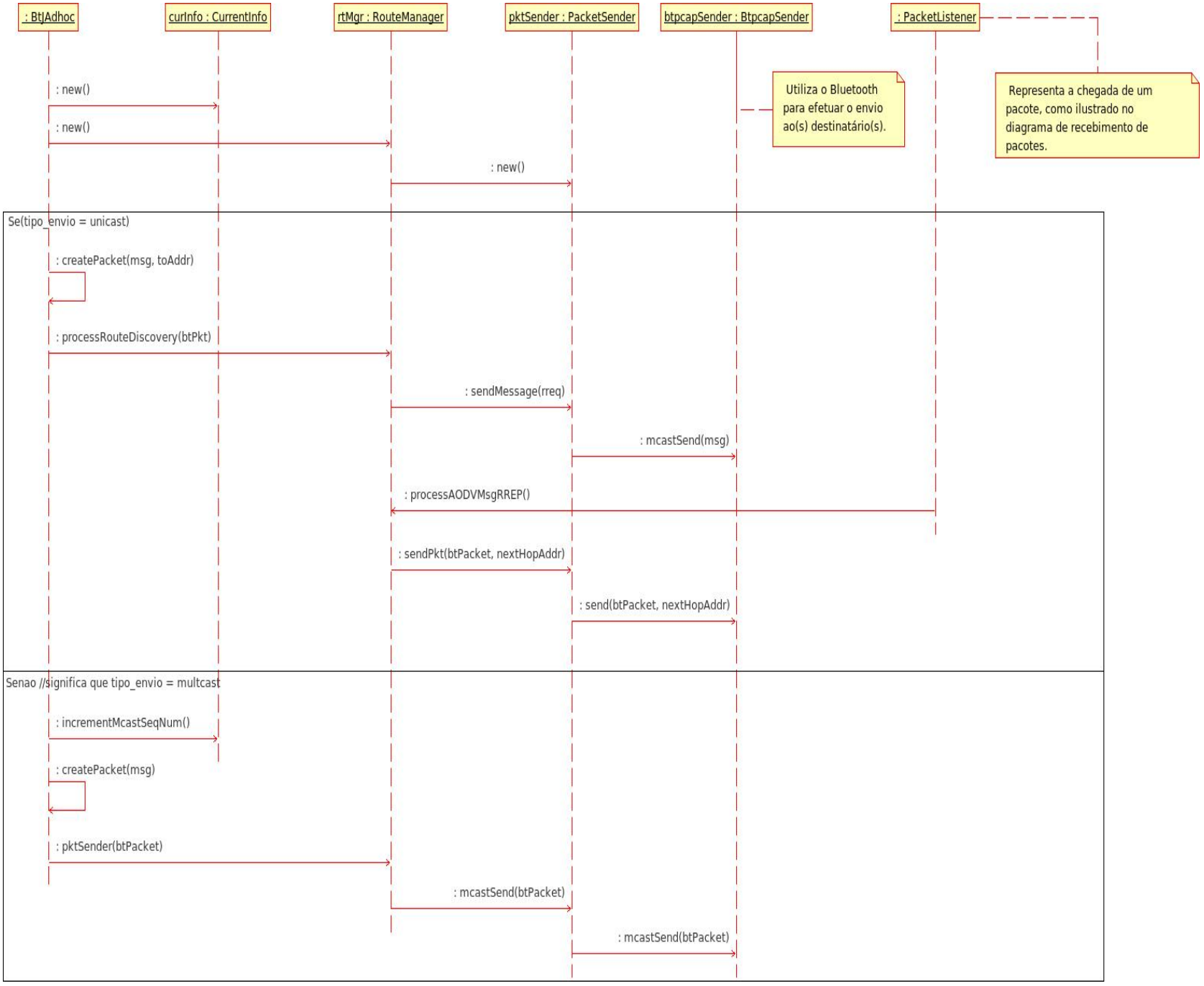


Figura 6.21: Diagrama de Seqüência do Envio de Pacotes

6.4.2 Dificuldades Encontradas

A tentativa de adaptar um robusto protocolo de roteamento impôs inúmeros complicadores relacionados tanto ao Bluetooth quanto ao desenvolvimento utilizando J2ME.

Primeiramente, algumas estruturas originalmente usadas na implementação do J- Adhoc, como as LinkedLists ou a interface Map (substituídas por Vectors) e a classe InetAddress da java.net (substituída por endereços representados em Strings), não puderam ser mantidas, visto que a J2ME não as suportava.

Ainda com relação a J2ME, o envio de mensagens de qualquer tipo também foi comprometido, uma vez que esta tecnologia não implementa serialização, não permitindo que instâncias de classes pudessem ser enviadas diretamente. Dessa forma, para transmissão de um pacote que é representado por um objeto Java, é necessário transformá-lo em um *array* de *bytes*. Ao ser recebido, esse *array* é então processado e os campos atribuídos a uma nova instância da classe no dispositivo de destino.

As limitações nativas dos dispositivos Bluetooth também impuseram algumas dificuldades na implementação. Como exemplo, temos os bloqueios efetuados a cada atividade do Bluetooth. Nele, apenas uma determinada operação pode ser efetuada por vez, ou seja, ele só pode buscar dispositivos, transmitir, ou receber informações em um determinado instante. Essa característica gera um grande atraso na execução das funcionalidades do protocolo e também no próprio gerenciamento das Piconets e Scatternets, tendo em vista que muitos conflitos podem acontecer durante as atividades.

Da mesma forma, o processo de busca do Bluetooth também gerou alguns problemas na implementação, principalmente em relação ao tempo necessário para realizá-la. Considerando que dispositivos móveis podem entrar e deixar a área de cobertura de outros dispositivos com extrema rapidez, seria interessante que os dispositivos mantivessem a tabela de vizinhos a mais atualizada possível. Na implementação proposta neste trabalho optou-se por um compromisso entre tempo de envio e precisão, visto que a tabela de vizinhos não é atualizada a cada envio, porém a cada 30 segundos. Desta forma, evita-se que a cada envio um tempo adicional de 10 segundos seja consumido. Por outro lado, um dispositivo que acabou de entrar na área de cobertura e que poderia auxiliar no roteamento de pacotes não será detectado imediatamente.

Capítulo 7

Estudos de Caso

Este capítulo tem como objetivo descrever os estudos de caso escolhidos para a aplicação dos conceitos apurados. A partir desta descrição pretende-se criar uma visão geral dos requisitos necessários aos sistemas que serão implementados.

Para analisar a viabilidade da utilização do Bluetooth na construção de ambientes ubíquos, dois estudos foram idealizados. Com eles pretende-se obter resultados consistentes tanto no que diz respeito a eficiência da tecnologia mediante uma aplicação de rede real, quanto ao suporte que os dispositivos já disponibilizam para implantação de serviços mais específicos.

Os estudos de caso selecionados buscam verificar, também, as implicações no processo de adaptação de um protocolo de roteamento utilizado nas redes Ad-Hoc para dispositivos que suportam a tecnologia Bluetooth.

Ambos casos possuem uma lógica bastante simples. Esta propriedade nos permite direcionar os esforços aos quesitos realmente importantes, lembrando que o objetivo do trabalho é não se limitar a ambientes simulados. Acréscimos desnecessários de complexidade poderiam contribuir para um desenvolvimento excessivamente custoso, onde as principais barreiras estariam presentes nas limitações dos dispositivos, e não exatamente no tratamento dos cenários propostos.

7.1 Cenário 1 – Sistema Ubíquo de Propagação de Notícias

Uma grande rede de supermercados decide adotar uma nova, simples e barata estratégia de *marketing*. Para colocá-la em prática a equipe levanta que será necessária, apenas, a aquisição de um adaptador Bluetooth para cada loja, os quais seriam usados nos computadores já disponíveis.

Após a implantação do sistema, um cliente, ao adentrar uma das lojas com o Bluetooth de seu celular ou dispositivo móvel ativado, recebe uma notificação da disponibilidade de uma nova notícia e é questionado do interesse de sua visualização. No caso de uma resposta positiva, a notícia é imediatamente exibida. O cliente percebe, então, que nela estão disponíveis informações referentes as principais ofertas do dia.

Algum tempo depois, enquanto o cliente ainda está no interior do estabelecimento, outra notificação lhe é apresentada e a notícia aceita por ele. Nesta, porém, o usuário do dispositivo percebe que as informações passadas já não são mais daquelas promoções, e sim, que uma nova fornada de pães-de-queijo acabou de ficar pronta. Contente com a possibilidade de levar para casa alguns pães recém-preparados o cliente desloca-se até a seção da padaria e faz seu pedido.

Como grande parte dos clientes da rede de supermercados possui algum dispositivo dotado da tecnologia Bluetooth, foi observado que esta estratégia de "*marketing* ativo" poderá auxiliar no acréscimo do volume de vendas. A equipe percebeu que manter o cliente constantemente informado, independente da sua localização na loja, será a ferramenta ideal para despertar seu interesse na compra.

7.1.1 Objetivos

Tendo como base o cenário acima, este estudo de caso consiste na propagação de notícia através de aparelhos celulares que possuem a tecnologia Bluetooth.

Para alcançar o que foi exemplificado um computador ficará responsável por coletar algumas notícias e iniciar a propagação. Este início acontecerá mediante a formação de uma Piconet, ou seja, os primeiros pacotes com as notícias serão entregues aos dispositivos no alcance do computador.

Como provavelmente essa Piconet não será capaz de alcançar todos os interessados, cada um dos nós ficará responsável por retransmitir a notícia aos demais dispositivos que ingressarem no seu espaço de cobertura, prosseguindo sucessivamente até que todos possuam a notícia em questão. A forma encontrada para possibilitar a expansão dessa conectividade será a implantação do protocolo de roteamento AODV, o qual terá seu desenvolvido efetuado com o auxílio das APIs de desenvolvimento J2ME. A escolha deste protocolo foi devido a sua aplicação direta nas redes Ad-Hoc, elemento essencial para a implementação do estudo de caso.

Todo o processamento e comunicação dos dispositivos será realizado de forma transparente e silenciosa. O objetivo é fazer o usuário interagir com o processo somente no momento em que for questionado sobre o interesse de visualizar a notícia recebida. É neste ponto que o conceito de ubiqüidade é tratado. Permitir que o usuário possa receber a notícia em qualquer local dentro do ambiente utilizado e, também, que ele não precise focar sua atenção no dispositivo enquanto a mesma não for solicitada nos coloca de frente às principais características da computação ubíqua. Note que a questão de tecnologia calma também pode ser observada.

Este estudo de caso permite, além das questões principais da ubiqüidade, verificar também o comportamento da propagação de pacotes em uma rede Ad-Hoc e, principalmente, sobre a tecnologia Bluetooth. É importante verificar, por exemplo, se esta propagação não acabaria se tornando uma espécie *spam* sendo distribuído sem qualquer tipo de controle.

7.2 Cenário 2 – Sistema Ubíquo de Controle de Freqüência

Uma determinada universidade decide alterar a forma com a qual controla a freqüência de seus alunos depois de perceber que fraudes poderiam estar ocorrendo.

A idéia então foi de abandonar os antigos cartões usados nas cancelas e adquirir um sistema simples e automatizado para ficar responsável por este controle. A base do novo sistema agora é um computador com um adaptador Bluetooth que detém uma base de dados com informações tanto dos alunos quanto dos seus aparelhos celulares, já que todos possuem um que contém o Bluetooth.

No momento que o aluno chega à sala de aula, seu celular envia para os aparelhos que estiverem no alcance do Bluetooth as informações de identificação e de chegada, como horário, por exemplo. Essas informações possuem um destinatário único, o computador do professor. Por esse motivo, o restante dos dispositivos que fizerem parte da rota apenas irão retransmitir o pacote, sem necessidade alguma de intervenção do seu usuário.

Após este primeiro envio de informação ao computador, o celular do aluno torna a enviar mensagens de confirmação em períodos de tempos aleatórios. Essa constante troca de informações é necessária para que o sistema tenha certeza que o aluno continua na sala. A partir do momento que esta informação parar de chegar, o computador entenderá como se o aluno já tivesse deixado suas atividades e ido embora.

Ao final de mais uma aula, o professor se dirige ao seu laptop e envia a todos os alunos informações sobre um conjunto de exercícios que deverá ser entregue. Na seqüência, encaminha aos mesmos um recado sobre a alteração que ocorrerá nos seus horários de atendimento.

7.2.1 Objetivos

Baseando-se neste cenário, a idéia aqui é criar um sistema de chamadas totalmente ubíquo, não necessitando de qualquer intervenção dos usuários.

Após o aluno ter adentrado a sala de aula e seu aparelho celular ter encontrado um outro dispositivo no seu alcance, iniciará o processo de notificação da presença. Ao ingressar à sala, será definida uma rota do dispositivo do aluno até o computador do professor, rota esta gerada através do protocolo de roteamento AODV, o qual deverá ser implantado nos dispositivos através do desenvolvimento J2ME.

Como podemos perceber, neste caso, além da necessidade de uma vasta propagação de pacotes (envio dos recados do professor), existe também a necessidade de tratar o envio de mensagens a um destinatário específico. Para este caso de *unicast* serão efetuados apenas os saltos requeridos para alcançar o destino-alvo da comunicação. A geração dessa rota se fará necessária porque, provavelmente, o baixo alcance característico da tecnologia Bluetooth não permitirá uma comunicação direta entre o *laptop* do professor e o celular do aluno.

No laptop estará implementada uma base de dados contendo as informações pessoais do aluno e as do seu dispositivo, permitindo que os pacotes recebidos sejam corretamente associados. Todo processo deve acontecer de forma transparente, gerando, assim, o ambiente ubíquo desejado. Neste estudo de caso, principalmente, a transparência poderá ser verificada sem maiores problemas, uma vez que intervenção alguma dos usuários será necessária para que sua presença seja registrada.

O roteamento também terá grande importância para o bom funcionamento do sistema, já que a ineficiência na formação das rotas e na entrega das informações poderá acarretar no não registro da presença.

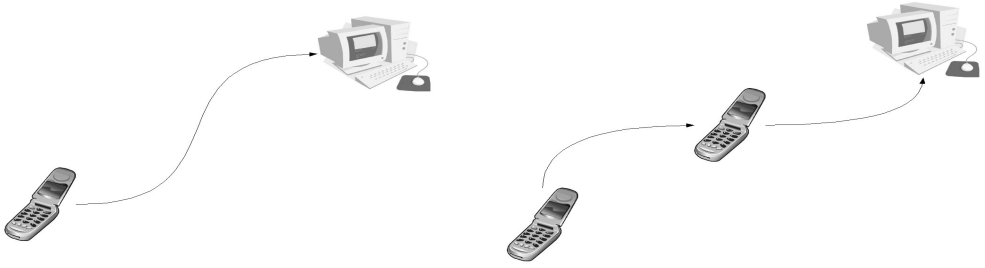
Assim como ocorre no cenário mencionado, constantes envios de mensagens de confirmação de presença continuarão sendo enviadas, o que representará para o sistema que o aluno ainda permanece em sala.

7.3 Testes Efetuados e Resultados Obtidos

Para realização dos testes foram usados três dispositivos, sendo eles dois aparelhos celulares Nokia [28] (um da Série 40 e outro da Série 60) e um computador pessoal utilizando um adaptador Bluetooth USB da marca Encore [7].

Os testes basearam-se na verificação do tempo gasto e na distância máxima suportada pelos dispositivos na utilização do protocolo para o envio de mensagens. O foco dos testes se manteve sobre o protocolo de roteamento visando verificar sua eficiência quanto a criação dos ambientes ubíquos propostos nos estudos de caso deste Capítulo. Através de uma análise feita na atuação do protocolo BtJAdhoc é possível ter uma noção do funcionamento dos sistemas desenvolvidos, no entanto, como não foi possível efetuar testes com um número grande de dispositivos, esta noção baseia-se apenas em conclusões que podem ser tiradas do comportamento observado em baixa escala.

Para obter os valores exibidos na Tabela 7.1, foram enviados pacotes de 126 *bytes* contendo apenas uma mensagem de texto, tanto diretamente entre dois dispositivos quaisquer, quanto entre esses mesmos dois dispositivos, porém com a necessidade de um intermediário para formação da ponte de comunicação (vide Figura 7.1), devido a distância máxima de alcance ter sido ultrapassada. Estes valores são conseqüência de dez execuções feitas para cada caso, onde o tempo médio representa a média aritmética entre os tempos obtidos.



(a) Direto (b) Com Intermediário

Figura 7.1: Tipos de envio utilizados

Tabela 7.1: Resultados Obtidos

Testes	Melhor Execução	Pior Execução	Tempo Médio
Envio Direto	23 seg.	8 min. e 30 seg.	6 min. e 52 seg.
Envio com Intermediário	3 min. 57 seg.	19 min. 23 seg.	14 min.

Essa enorme diferença obtida entre os melhores e piores tempos de execução são em decorrência das dificuldades que os dispositivos encontraram ao tentar transmitir seus pacotes de controle AODV para a descoberta da rota e, também, da dificuldade encontrada para a transmissão final. Os menores tempos de execução mostram que podem ocorrer comunicação sem muitas ocorrências de conflitos, no entanto, essas execuções ocorrem muito esporadicamente.

Como as tabelas de roteamento sempre iniciam vazias, todo processo de descoberta de rota teve que ser executado para os dois tipos de teste (com e sem necessidade de intermediário). É fácil observar o motivo pelo qual o tempo necessário para a comunicação aumenta nos casos referentes a transmissão com existência de dispositivos intermediários. Tendo em vista que a quantidade de conflitos pode aumentar e que, nos testes efetuados, só poderia existir apenas um caminho para o hospedeiro de destino, todo processo de descoberta de rota e envio da mensagem final teve que ficar aguardando até que os dispositivos conseguissem efetivar suas interações.

Em outros casos, tanto na comunicação direta quanto na comunicação com necessidade de saltos, a mensagem final simplesmente não foi entregue. Isso ocorreu devido a grande quantidade de conflitos não permitir que os pacotes de controle AODV ou o próprio pacote contendo a mensagem fossem transmitidos antes que seu tempo de vida expirasse, ou seja, a existência dos conflitos fez com que o processo fosse interrompido e os pacotes descartados. Este problema acontece com maior freqüência quando trata-se da necessidade de formação de pontes (representando cerca de 93% das tentativas de execução, num total de 10 tentativas), pois o "dispositivo do meio" deve conciliar as mensagens de controle recebidas pelos dois outros dispositivos com o envio das suas, além de ocorrer travamentos constantes no adaptador USB utilizado. Nas execuções feitas através do envio direto, a situação é exatamente oposta ao envio com intermediário, onde pode-ser observar a conclusão do envio em mais de 90% das tentativas.

Esses conflitos são um fator complicador para os objetivos deste trabalho, sendo gerados pelas limitações inerentes ao Bluetooth. Remetendo à Seção 3.3.1 e a Figura 3.5 nela apresentada, a existência dos estados de funcionamento do Bluetooth é a grande responsável por essa problemática. Devido à troca de estados necessária à cada ação do dispositivo, é extremamente freqüente a ocorrência de bloqueios de comunicação (aos quais chamamos de conflitos), ou seja, frequentemente um dispositivo fica incapacitado de receber alguma informação por estar atualmente em seu estado de busca ou tentando enviar algo.

Através dos testes foi possível observar, também, que a distância pode variar levando em consideração o dispositivo e o ambiente no qual se está realizando o experimento. Barreiras físicas, como paredes ou outros objetos, são fatores que auxiliam na redução desse alcance, fazendo com que a utilização dos saltos permitidos pelo protocolo de roteamento sejam realmente necessários. Com a utilização do protocolo desenvolvido conseguimos os alcances máximos apresentados na Tabela 7.2, sendo estes valores obtidos de forma empírica e em ambiente real. Não foi possível observar, porém, se esta distância influenciou consideravelmente nos tempos de execução, pois os conflitos ocorrem de maneira muito irregular, dificultando analisar o comportamento por outros pontos de vista.

Tabela 7.2: Distâncias com o Protocolo BtJAdhoc

Teste	Distância
Sem Intermediário	8.6 m
Com Intermediário	16,8 m

Os arquivos criados para implantação nos celulares (programas compilados) geraram em torno de 60 a 70 KB de espaço ocupado, sendo que o protocolo ocupa 64,5 KB e as aplicações 67 KB. Não foi possível verificar nos dispositivos a quantidade de memória ocupada durante suas execuções. Os dispositivos utilizados não apresentam tal informação.

7.3.1 Considerações

Inserir o Bluetooth em ambientes reais de redes permitiu um contato direto com suas limitações e suas possibilidades de utilização.

Em nosso sistema de controle de presença, por exemplo, a possível existência de grande fluxo de informações é clara, pois, supondo a necessidade de formação de pontes para propagação dos dados da chegada do aluno, fica fácil verificar que o número de mensagens de controle trocadas entre os dispositivos presentes no ambiente ubíquo é consideravelmente alto. Através então dos resultados obtidos com o protocolo, podemos concluir que existe a possibilidade de um aluno simplesmente não ter sua presença confirmada, mesmo estando com o Bluetooth ativado e com o sistema em execução, o que geraria grandes transtornos. Os testes realizados com o sistema basearam-se apenas nos três dispositivos, sendo assim, a relação exata entre as dimensões de uma sala de aula e a real necessidade da geração e pontes não foi verificada. No

entanto, supondo que realmente exista essa necessidade, e baseando-se nos experimentos efetuados, podemos inferir que Bluetooth ainda deixa a desejar quando a quantidade de comunicação é muito intensa.

Para propagação de pacotes via *multicast* ou *broadcast*, a implementação do BtJAdhoc desempenhou suas funcionalidades de forma satisfatória. Tendo em vista que o Bluetooth não implementa tais tipos de envio de forma nativa, o protocolo desenvolvido teve que ser capaz de controlar com perfeição os dados transmitidos, para que estes não permanecessem "inundando" os dispositivos infinitamente. Para o *desktop* que gerencia o envio de notícias, o sistema desenvolvido atribui um número de seqüência específico para cada nova notícia, permitindo que este número funcione como uma forma de controle para os demais dispositivos.

Nos celulares, a aplicação apresenta uma interface com o usuário. Sempre que uma nova notícia é recebida, o sistema internamente verifica o número de seqüência do pacote recebido e o compara com o número do último pacote. Caso este número seja maior que o anterior, o pacote é retransmitido e é exibida na tela do aparelho celular uma mensagem informando o usuário que existe uma nova notícia disponível. A aplicação dá ao usuário o direito de escolha entre visualizar ou não a notícia. Em caso afirmativo a notícia é exibida, do contrário simplesmente descartada. Nos testes realizados o sistema ubíquo de propagação de notícias executou sem maiores problemas, o que nos leva a crer que, em escalas maiores, esta afirmação pode ser repetir, uma vez que o tratamento das inundações foi feito e que mensagens repetidas não podem ser mantidas infinitamente na rede.

A necessidade de desenvolvimento de interfaces foi verificada a partir do momento que percebemos a existência de uma discrepância significativa entre o nível de limitação proveniente dos diferentes modelos de aparelhos celulares. Alguns modelos, como os da Série 40, impedem que um aplicativo permaneça em execução sem que este esteja sempre em primeiro plano, dificultando o pleno manuseio do aparelho; outros, como os da Série 60, já permitem a execução dos aplicativos sem a necessidade de uma interface sempre visível; existem ainda modelos, como o A1200 fabricado pela Motorola [25], que, mesmo impedindo a existência de um aplicativo sem interfaceamento com usuário, ainda permitem que o dispositivo seja utilizado sem problemas. Com base nessas constatações, decidiu-se por implementar um sistema provido de interface para padronizar sua utilização junto aos diversos dispositivos.

Algo que surgiu durante nossos testes e que não podemos afirmar ser um problema especificamente do *hardware* utilizado ou do *driver* de controle, foi o travamento dos serviços do adaptador Bluetooth USB quando a quantidade de atividade aumentava. Em grande parte das execuções de teste do protocolo e dos sistemas ubíquos (principalmente o de controle de freqüência) o adaptador simplesmente parava de funcionar devido a quantidade de informação que estava sendo trocada no momento.

Outro grande complicador observado no aparelho da Nokia Série 40 utilizado nos testes, foi o fato do dispositivo não permitir que as comunicações via Bluetooth possam ocorrer de forma "silenciosa". O aparelho solicita a intervenção do usuário antes de efetuar cada atividade de comunicação, seja ela de envio ou recepção de dados.

Capítulo 8

Conclusão

Este trabalho apresentou a adaptação de um protocolo de roteamento Ad-Hoc para o Bluetooth e sua utilização na criação de um ambiente computacional ubíquo real, mesmo considerando as inúmeras limitações existentes nas tecnologias utilizadas. Porém sua aplicação prática pode ser inviável devido, principalmente, aos problemas de atraso gerados pela busca de dispositivos e a baixa velocidade de comunicação.

É importante mencionar que nossas limitações tecnológicas impediram que a ubiqüidade ideal proposta e defendida por Mark Weiser [38] em muitos de seus trabalhos fosse alcançada. Da mesma forma que uma implementação voltada para os princípios da tecnologia calma. Para possibilitar a execução do protocolo e dos sistemas propostos pelo trabalho nos dispositivos móveis, foram necessárias efetuar suas instalações em cada dispositivo, necessitando ainda que uma interface com o usuário fosse disponibilizada, o que torna a intervenção do usuário muito grande para a proposta de Weiser.

Foi possível verificar uma discrepância significativa entre diferentes modelos de aparelhos celulares, o que dificulta consideravelmente o processo de desenvolvimento de uma aplicação genérica, principalmente se levarmos em conta a sofisticação de determinados tipos de aplicações, como é o caso do objeto-alvo deste trabalho.

Com base nos resultados, é possível perceber que se o ambiente ubíquo necessitar de grande dinamicidade, ou seja, caso ele necessite de constante troca de informações e grande agilidade nas mesmas, o Bluetooth ainda não é a alternativa mais indicada, principalmente no caso da necessidade de comunicação Ad-Hoc. Por outro lado, pode ser totalmente aceito em algum tipo de atividade que não exija respostas imediatas e onde o custo-benefício seja um fator preponderante, lembrando que o Bluetooth fornece comunicação via rádio de baixo custo e baixo

consumo de energia.

Outro fator que permite ao Bluetooth permanecer como uma boa opção é sua crescente adoção, podendo ser encontrado em telefones celulares, *notebooks*, PDAs, ou ser adquirido separadamente através dos adaptadores USB, permitindo que as aplicações que o utilizam possam atingir a um número muito grande de usuários.

Os trabalhos futuros compreendem no desenvolvimento de estratégias mais eficientes, as quais possibilitariam que as comunicações em meio a utilização do protocolo adaptado pudessem ocorrem com um menor número de conflitos de atividade do Bluetooth, tornando sua eficiência na comunicação e velocidade de troca de informações mais aceitáveis. Espera-se, também, a possibilidade de desenvolver testes com uma quantidade maior de celulares, colocando o protocolo adaptado e o sistemas frente a um ambiente mais complexo, possibilitando uma análise mais aprofundada.

Referências Bibliográficas

[1] ALMEIDA, A.; BARRENHO, R. **A Computação Ubíqua: Onde estamos... para onde vamos...** Consultado na Internet em: 11/07/2008. Disponível em: http://alunos.di.uevora.pt/ l19637/cub1/index.html.

[2] ANJUM, F.; MOUCHTARIS, P. **Security for Wireless Ad–hoc Networks**. John Wiley & Sons, 2006.

[3] ARAUJO, R. B. Computação ubíqua: Princípios, tecnologias e desafios. In: XXI SIMPOSIO BRASILEIRO DE REDES DE COMPUTADORES, 2003. **Proceedings...** Natal: [s.n.], 2003. p.45–115.

[4] BLUETOOTH. **Specs**. Consultado na Internet em: 11/02/2008. Disponível em: http://www.bluetooth.com.

[5] CANO, J.-C.; MANZONI, P.; TOH, C. K. Ubiqmuseum: A bluetooth and java based context-aware system for ubiquitous computing. **Wireless Personal Communications**, Hingham, v.38, n.2, p.187–202, 2006.

[6] DAHL, C. Practical ubiquity with mobile phones. In: X TECH CONFERENCE, 2007. **Proceedings...** Paris: [s.n.], 2007. p.1–3.

[7] ELECTRONICS, E. **Encore Electronics**. Consultado na Internet em: 02/11/2008. Disponível em: http://www.encore-usa.com.

[8] FIGUEIREDO, C. M. S.; NAKAMURA, E. Computação móvel: Novas oportunidades e novos desafios. **T&C Amazônia**, Manaus, v.1, n.2, p.16–28, Junho, 2003.

[9] FLICKR. **Flick: Gerenciamento e Compartilhamento**. Consultado na Internet em: 21/03/2008. Disponível em: http://www.flickr.com.

[10] FUJII, K. **Jpcap - a Java library for capturing and sending networks packets**. Consultado na Internet em: 02/09/2008. Disponível em: http://netresearch.ics.uci.edu/kfujii/jpcap/doc/.

[11] HAR-SHAI, L. et al. Inter-piconet scheduling in bluetooth scatternets. In: OPNETWORK, 2002. **Proceedings...** Washington: [s.n.], 2002. p.1–11.

[12] IEEE. **IEEE 802.15 - Wireless Personal Area Networks**. Consultado na Internet em: 22/07/2008. Disponível em: http://standards.ieee.org/getieee802/802.15.html.

[13] KARPIJOKI, V. Security in ad hoc networks. In: SEMINARS ON NETWORK SECURITY, 2000. **Proceedings...** Helsinki: [s.n.], 2000. p.1–16.

[14] KLINGSHEIM, A. N. **J2ME Bluetooth Programming**. Bergen: University of Bergen, Junho, 2004. Dissertação.

[15] KNUDSEN, J.; LI, S. **Beginning J2ME: From Novice To Professional (Beginning from Novice to Professional)**. Berkely, CA, USA: Apress, 2005.

[16] KROLL, M.; HAUSTEIN, S. **Java 2 Micro Edition (J2ME) Application Development**. Pearson Education, Junho, 2002.

[17] KUROSE, J. F.; ROSS, K. W. **Redes de Computadores e a Internet: Uma abordagem top-down**. Trad. 3 ed. ed. São Paulo: Addison Wesley, 2006.

[18] LINDHOLM, T.; YELLIN, F. **The Java(TM) Virtual Machine Specification (2nd Edition)**. Prentice Hall PTR, Abril, 1999.

[19] LIN, C.-H. **Bluetooth Scatternet Routing for Wireless Home Network**. Taiwan: National Dong Hua University, 2002. Dissertação.

[20] LOUREIRO, A. A. F. et al. Redes de sensores sem fio. In: XXI SIMPOSIO BRASILEIRO DE REDES DE COMPUTADORES, 2003. **Proceedings...** Natal: [s.n.], 2003. p.179–226.

[21] MAHMOUD, Q. H. **The Java APIs for Bluetooth Wireless Technology**. Consultado na Internet em: 23/07/2008. Disponível em: http://developers.sun.com/mobility/midp/articles/bluetooth2/index.html.

[22] MATEUS, G. R.; LOUREIRO, A. A. F. **Introdução à Computação Móvel**. Rio de Janeiro: 11ª Escola de Computação, 1998.

[23] MICROSYSTEMS, S. **Java**. Consultado na Internet em: 02/11/2008. Disponível em: http://www.java.com.

[24] MICROSYSTEMS, S. **PersonalJava**. Consultado na Internet em: 02/11/2008. Disponível em: http://java.sun.com/products/personaljava.

[25] MOTOROLA. **Motorola Worldwide**. Consultado na Internet em: 02/11/2008. Disponível em: http://www.motorola.com.

[26] Motorola, Austin. **Java APIs for Bluetooth Wireless Technology (JSR 82)**, 2005.

[27] MCDERMOTT-WELLS, P. What is bluetooth? **Potentials, IEEE**, [S.l.], v.23, n.5, p.33–35, 2005.

[28] NOKIA. **Nokia Connecting People**. Consultado na Internet em: 02/11/2008. Disponível em: http://www.nokia.com.

[29] OF BASEL, U. **LUNAR - Lightweight Underlay Network Ad hoc Routing**. Consultado na Internet em: 15/04/2008. Disponível em: http://cn.cs.unibas.ch/projects/lunar/.

[30] PERKINS, C.; BELDING-ROYER, E.; DAS, S. **Ad hoc On-Demand Distance Vector (AODV) Routing**. Consultado na Internet em: 11/04/2008. Disponível em: http://www.ietf.org/rfc/rfc3561.txt.

[31] PEREIRA, D. A. **JOGOS UBIQUOS COM BLUETOOTH**. Recife: Universidade Federal de Pernambuco, Fevereiro, 2006. Monografia.

[32] POHFLEPP, S. **Buttons: A Bind Camera**. Consultado na Internet em: 21/03/2008. Disponível em: http://www.blinksandbuttons.net/buttons_en.html.

[33] PRIESS, W. et al. Bluenets: Uma ferramenta para estudo de desempenho do bluetooth. In: IV WORKSHOP DE COMUNICAçãO SEM FIO E COMPUTAçãO MóVEL, 2005. **Proceedings...** São Paulo: [s.n.], 2005. p.161–171.

[34] RENSFELT, O. Lunar over bluetooth. Uppsala: Uppsala University, Março, 2004. Relatório técnico .

[35] SAHA, D.; MUKHERJEE, A. Pervasive computing: a paradigm for the 21st century. **IEEE Computer Society**, [S.l.], v.36, n.3, p.25–31, 2003.

[36] SOLHEID, A. **AODV enhanced by Smart Antennas**. Namur: University of Namur, Setembro, 2005. Dissertação.

[37] University of Bremen, Bremen. **JAdhoc System Design**, 2003.

[38] WEISER, M. The computer for the 21st century. **Scientific American**, [S.l.], v.265, n.3, p.66–75, 1991.

[39] WEISER, M. The world is not a desktop. **Interactions**, New York, NY, USA, v.1, n.1, p.7–8, Novembro, 1993.

[40] WEISER, M.; BROWN, J. S. **Designing Calm Technology**. Consultado na Internet em: 11/07/2008.