UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

SISTEMA GERENCIADOR DE PROJETOS UTILIZANDO TECNOLOGIA WIRELESS

JOSUÉ VLADIMIR GRANJENSE DE LIMA SARAIVA

ORIENTADOR: PAULO HENRIQUE PORTELA DE CARVALHO

PROJETO FINAL DE GRADUAÇÃO

BRASÍLIA/DF: JULHO – 2005

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

SISTEMA GERENCIADOR DE PROJETOS UTILIZANDO TECNOLOGIA WIRELESS

JOSUÉ VLADIMIR GRANJENSE DE LIMA SARAIVA

PROJETO FINAL DE GRADUAÇÃO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA, COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE ENGENHEIRO.

APROVADA POR:

-

PAULO HENRIQUE PORTELA DE CARVALHO, Doutor, UnB
(ORIENTADOR)

-

LEONARDO R. A. X. MENEZES, Ph. D., UnB
(EXAMINADOR)

-

ALEX HELDER C. DE OLIVEIRA, Engenheiro, UnB
(EXAMINADOR)

DATA: BRASÍLIA/DF, JULHO DE 2005

FICHA CATALOGRÁFICA

SARAIVA, JOSUÉ VLADIMIR GRANJENSE DE LIMA.
Sistema Gerenciador de Projetos Utilizando Tecnologia Wireless. [Distrito Federal] 2005.
84p., 297 mm (ENE/FT/UnB, Bacharel, Engenharia Elétrica, 2005).

Projeto Final de Graduação – Universidade de Brasília, Faculdade de Tecnologia. Departamento de Engenharia Elétrica.

1. Wireless

I. ENE/FT/UnB.
II. Título (Série)

REFERÊNCIA BIBLIOGRÁFICA

SARAIVA, JOSUÉ VLADIMIR GRANJENSE DE LIMA (2005). Sistema Gerenciador de Projetos Utilizando Tecnologia Wireless. (Projeto Final de Graduação), Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF. 84p.

CESSÃO DE DIREITOS

NOME DO AUTOR: Josué Vladimir Granjense de Lima Saraiva
TÍTULO DA DISSERTAÇÃO: Sistema Gerenciador de Projetos Utilizando Tecnologia Wireless.
GRAU/ANO: Bacharel/2005.

Josué Vladimir Granjense de Lima Saraiva
Condomínio Morada dos Nobres – Qd. 2C Cs. 22
CEP 73.070-028 – Sobradinho – DF – Brasil

AGRADECIMENTOS

Josué agradece primeiramente a Deus pelo amparo nos momentos mais difíceis que teve que enfrentar na vida e na conclusão desse trabalho. Aos seus pais, José e Joene, pelo imenso apoio, incentivo e sempre muito amor e dedicação. Ao professor orientador Paulo Portela, pela confiança, força e incentivo. Aos demais professores da UnB, que contribuíram para sua formação acadêmica. A sua namorada, Monique, e a todos os amigos e familiares.

DEDICATÓRIA

O Autor dedica este trabalho aos seus familiares e amigos.

RESUMO

Este Projeto implementa o sistema Gerenciador de Projetos, que é estruturado em um ambiente Web e utiliza serviços para uma comunicação absoluta através de diversas formas de mensagens como o SMS, o Email e o SMS Bluetooth. Tal sistema tem como objetivo proporcionar diversas formas de envio de mensagens utilizados em um único aplicativo. Como forma de validação dos diversos modos na entrega de mensagens foi criado um aplicativo que gerencia projetos e mensagens são entregues aos participantes do projeto. A arquitetura, o funcionamento, as comunicações estabelecidas entre os componentes, bem como ferramentas utilizadas no desenvolvimento do sistema, encontram-se descritos neste trabalho.

ABSTRACT

This Project implements the Gerenciador de Projetos(Project Manager) system wich is based in a WEB enviorement, and uses services for a full communication throught a wide message format like SMS, email and Bluetooth. That system has the objective of proving so many ways of sending message used in a single application. As form of validation of several ways of message delivery was created a application that manages the projetcs, and messages are delivered to project's participants. The arquitecture, the functioning, the estabilished communications between components, as well as the tools used in desenvolviment of the system was described in this work.

SUMÁRIO

ÍNDICE DE FIGURAS

LISTA DE ABREVIAÇÕES

Abreviação	Significado
AP	Access Point
API	Application Program Interface
AWT	Abstract Window Toolkit
CLDC	Connected Limited Device Configuration
DAO	Data Access Object
DB	Database
HTML	HyperText Markup Language

HTTP	HyperText Transfer Protocol
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
ISO	International Organization for Standardization
J2ME	Java 2 Platform, Micro Edition
J2SE	Java 2 Platform, Standard Edition
J2EE	Java 2 Platform, Enterprise Edition
JDBC	Java Database Connectivity
JSP	Java Server Pages
JVM	Java Virtual Machine
LAN	Local Area Network
MIDP	Mobile Information Device Profile
MVC	Model-View-Controller
PDA	Personal Digital Assistant
QoS	Quality of Service
RDBMS	Relational Database Management System
RMI	Remote Method Invocation
RMS	Record Management System
SAP	Service Access Point
SDK	Software Development Kit
SMS	Short Messaging Service
SQL	Structured Query Language
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
URL	Uniform Resource Locator
USB	Universal Serial Bus
VO	Value Object
WLAN	Wireless Local Area Network
WML	Wireless Markup Language
WPAN	Wireless Personal Area Network
XML	eXtensible Markup Language

1 – INTRODUÇÃO

1.1 – Motivações e Objetivo do Trabalho

Com o avanço da tecnologia nos dias atuais e a necessidade de uma comunicação rápida e segura, as empresas vêm buscando o máximo de mobilidade e conectividade com os funcionários, parceiros e clientes. Com o surgimento dos laptops, PDAs e celulares veio a mobilidade para que se tenha sempre à mão os nossos dados. Com a mobilidade veio a conectividade para estabelecer a conexão desses dispositivos a redes wireless e possibilitar conexões entre os próprios dispositivos.

A idéia desse projeto foi desenvolver um aplicativo com auxílio a processos focado em uma comunicação absoluta. Este projeto foi desenvolvido em duas partes. No Projeto Final I foi desenvolvido o Sistema Gerenciador de Mensagens e no Projeto Final II foi desenvolvido o Sistema Gerenciador de Projetos.

O Sistema Gerenciador de Mensagens consiste em um aplicativo que envia mensagens a pessoas de um determinado grupo previamente cadastrados, cujos tipos de mensagens são: Emails, SMS a celulares ou então um “SMS Bluetooth. O Sistema Gerenciador de Mensagens está dividido em quatro grandes partes: Access Point Bluetooth, Servidor, Desktop e Celular.

O Sistema Gerenciador de Projetos consiste em cadastro de projetos ou tarefas em um ambiente Web e as informações de tais projetos são enviadas através de Emails, SMS ao celulares ou então um “SMS Bluetooth” aos envolvidos no projeto.

A conectividade de um móvel (celulares, PDAs, etc) com PC´s é feita através de Bluetooth que é um protocolo de comunicação wireless estudado neste projeto de graduação.

1.2 – Tecnologia do Sistema Proposto

O ambiente desenvolvido no projeto utiliza ferramentas que oferecem suporte à implementação dos serviços mencionados, tais como, a linguagem Java – J2ME, J2SE, J2EE (JDBC, RMI, Servlet, JSP) –, Bluetooth, XML, - Design Patterns, Struts e *MySQL*, utilizando-se o sistema operacional *Windows*.

1.3 – Estrutura da Monografia

O texto está dividido em cinco capítulos. O Primeiro Capítulo, que é composto pela introdução, visa à contextualização do projeto no cenário atual, mostrando uma breve descrição do projeto e de seus objetivos bem como a motivação para sua concretização. O Capítulo Dois descreve as tecnologias utilizadas nas comunicações entre os elementos. O Capítulo Três contextualiza o Sistema Gerenciador de Mensagens, explicando as suas partes, citando exemplos e testes aplicados sobre o referido sistema. O Capítulo Quatro trata da arquitetura do ambiente Sistema Gerenciador de Projetos, mostrando a implementação do sistema e as diferenças deste com o Sistema Gerenciador de Mensagens. E por último há o Capítulo Cinco, que trata da conclusão do projeto, assim como das dificuldades que foram encontradas e das sugestões de projetos futuros.

2 - TECNOLOGIAS DE COMUNICAÇÃO UTILIZADAS

2.1 – BLUETOOTH

2.1.1 –Introdução

Bluetooth é um protocolo de comunicação wireless que usa uma tecnologia de rádio short-range que permite a conectividade wireless entre dispositivos móveis. Essa nova tecnologia foi sabiamente projetada uma vez que os três objetivos principais para o Bluetooth eram: tamanho pequeno, consumo de potência mínimo, e preço baixo. A tecnologia foi projetada para ser simples, e o seu alvo é transformar-se em um padrão de fato na conectividade wireless. O rádio de Bluetooth opera na faixa não-licenciada de 2.4 GHz. Em alguns países, esta faixa é reservada para o uso das forças armadas, mas estes países têm começado a liberar essa faixa para o uso geral. A taxa de dados bruta máxima é 1 Mbps. A escala de Bluetooth depende da classe da potência do rádio. Para a maioria dos dispositivos espera-se usar o rádio da classe 2 que fornece 0 de potência nominal da saída do dBm, tendo por resultado uma escala de até 10 metros em um ambiente livre. Esta escala é suficiente para aplicações de substituições de cabos. Quando uma escala mais longa for necessária (por exemplo, em pontos de acesso, APs), um rádio mais poderoso (classe 1) pode ser usada. O consumo de potência maior não é um problema se o dispositivo for uma parte de equipamento fixo. Com dispositivos móveis tais como telefones móveis, as relações de potência e consumo são cruciais e conseqüentemente a classe 2 é a única opção praticável.

2.1.1.1 - Bluetooth VS. Infravermelho

Há algum tempo a comunicação wireless entre dois computadores é possível. PDAs puderam fazer isso por anos usando a tecnologia infravermelha. Um inconveniente ao infravermelho é que os dispositivos envolvidos devem estar apenas alguns centímetros separados, e o mais importante, os transceptores infravermelhos devem ver-se "olho a olho". Se qualquer uma daquelas circunstâncias não for encontrada a transmissão falhará. O Bluetooth supera a primeira limitação tendo uma escala nominal de aproximadamente 10 metros (classe 2). Também supera a segunda limitação, pois trabalha como um rádio, sendo as transmissões omnidirecionais. Conseqüentemente, não há nenhum efeito da linha de sinal quando uma comunicação ocorre entre dois dispositivos Bluetooth.

2.1.1.2 - Bluetooth VS. 802.11b

Um dos mais famosos protocolos wireless é o 802.11b (o protocolo wireless) do IEEE. Bluetooth e 802.11b foram criados para realizar dois objetivos diferentes, embora ambas as tecnologias operassem na mesma faixa de freqüência: 2.4 GHz. O fato de ter ambas as tecnologias operando na mesma escala de freqüência não significa que irão interferir quando colocadas na escala, de acordo com um estudo da pesquisa de Forrester conduzido em 2001. O objetivo de Wireless LAN (802.11b) é conectar dois dispositivos relativamente grandes que têm lotes de potência em velocidades elevadas. Tipicamente, esta tecnologia é usada para conectar dois laptops dentro de 100 metros em 11 Mb/s. Esta tecnologia é também útil para os administradores da rede que querem estender sua LAN a lugares onde é caro ou inconveniente o funcionamento de cabos. Por outro lado, a tecnologia Bluetooth pretende conectar dispositivos menores como PDAs e telefones móveis dentro de uma escala de 10 metros em uma taxa de 1 Mb/s. As taxas de dados mais lentas e as escalas mais curtas permitem que Bluetooth seja uma tecnologia wireless de baixa potência. Comparado aos dispositivos 802.11b, alguns dispositivos de Bluetooth podem facilmente consumir 500 vezes menos potência, que pode fazer uma diferença enorme na vida da bateria de muitos dispositivos móveis. O Bluetooth pretende também ser usado como uma tecnologia de substituição de cabos. No caso de periféricos múltiplos conectados a um computador usando RS-232 ou USB, o Bluetooth é a solução ideal se for usada tecnologia wireless para estes dispositivos. É quase impossível conectar periféricos em um computador usando a tecnologia 802.11b (à exceção das impressoras). Bluetooth tem até mesmo uma potencialidade interna para uma comunicação áudio wireless.

Pergunta-se: Uma tecnologia pode substituir a outra? Dificilmente. Bluetooth nunca substituirá 802.11b pelo seguinte: As altas taxas de transferências e uma comunicação de longo alcance de dispositivos 802.11b que não são alcançadas por dispositivos Bluetooth. Por outro lado, 802.11b nunca substituirá o Bluetooth porque 802.11b não pode ser usado para se comunicar com periféricos. 802.11b requer demasiada potência para uma comunicação. Não é indicado para pequenas transferências de dados. Além de não ter sido projetado para uma comunicação de voz. Na arena wireless de comunicação, não há nenhuma tecnologia que é mais bem servida para cada aplicação possível. Bluetooth ou 802.11b podem ser usados para uma comunicação wireless entre computadores. Ambos têm seu lugar no mercado e podem atuar em seus nichos. Uns protocolos mais novos de

LAN wireless são o 802.11a e o 802.11g que promoverão a desobstrução da distinção entre Bluetooth e a LAN wireless porque estendem a limitação da largura de faixa 802.11b's a 54 Mb/s.

2.1.1.3 - Principais Dispositivos Bluetooth no Mercado Hoje

Existem vários dispositivos Bluetooth no mercado atualmente. Os principais são os periféricos dos computadores, celulares e palms (PDAs). As antenas adaptadoras Bluetooth comercializadas atualmente são em sua maioria da classe 1, isto é, de 100 metros e são conectadas geralmente em entradas USBs. A seguir temos alguns exemplos destes dispositivos:

Figura 2.1 – PDA Bluetooth.

Figura 2.2 – Celulares com Bluetooth.

Figura 2.3 – Adaptador Bluetooth USB (classe 1 – 100 metros).

2.1.1.4 – Ponto a Ponto e Multiponto

Um fator que distingue vários dispositivos de Bluetooth é sua potencialidade na conexão. Se um dispositivo Bluetooth pode somente suportar uma comunicação ponto-a-ponto, então ele pode somente se comunicar a um único dispositivo de Bluetooth a cada vez. A Figura 2.4 demonstra a comunicação ponto-a-ponto em Bluetooth.

Figura 2.4 – Comunicação ponto-a-ponto em Bluetooth.

Uma comunicação ponto-a-ponto não é necessariamente uma coisa ruim. Se você tiver um telefone Bluetooth, você necessita realmente somente uma conexão a seu telefone Bluetooth.

Por outro lado, um dispositivo multiponto pode comunicar-se ao mesmo tempo com até outros sete dispositivos. A figura 2.5 é um diagrama de um dispositivo multiponto que se comunica a outros dispositivos dentro da escala usando a tecnologia de Bluetooth. [1, 2]

Figura 2.5 – Diagrama de um dispositivo multiponto.

2.1.2 - Padrão IEEE 802.15.1

Apresentemos agora um resumo dos itens mais relevantes do padrão IEEE que descreve a tecnologia Bluetooth. Este nos serviu de embasamento teórico para o desenvolvimento do Sistema de Gerenciamento e Distribuição de Mensagens em Ambiente Corporativo Usando Tecnologia Wireless.

2.1.2.1 – A Tecnologia de Comunicação da WPAN

Os dispositivos eletrônicos pessoais estão ficando cada vez mais inteligentes e interativos. Muitos destes estão crescendo sua capacidade em dados. Esta capacidade lhes permite reter, usar, processar e comunicar uma grande quantidade de informação. Com uma base de dados PIM (Personal Information Management) estes dispositivos são capazes de se interconectarem e se sincronizarem.

Tradicionalmente, diversos tipos de cabos vêm sendo usados para interconectar dispositivos pessoais. Entretanto, muitos usuários não se dão muito bem com esses cabos. Os cabos podem ser perdidos ou estragados facilmente. Assim, nasce um desejo de se desenvolver uma solução wireless para a interconexão de dispositivos.

Os requerimentos gerais para uma solução sem fio estão relacionados com o layout dos dispositivos (*PDAs – Personal Digital Assistant*) que deve permanecer o mesmo apesar da nova tecnologia. Além disso, a solução wireless deveria ser voltada tanto para o mercado consumidor quanto para o mercado de negócios.

Interconectar dispositivos pessoais é diferente da interconexão de computadores. Uma WPAN pode ser vista como uma bolha de comunicação pessoal ao redor da pessoa. Esta bolha se move para onde a pessoa for e os dispositivos pessoais dentro dela se conectam e interagem uns com os outros.

Os dispositivos que estão se comunicando não precisam estar na “linha de visão” uns dos outros. Por esta razão, as WPANs empregam tecnologia de rádio freqüência (RF) para prover flexibilidade para comunicar os dispositivos escondidos. O padrão IEEE 802.15.1 apresenta a tecnologia RF de WPAN baseada na tecnologia wireless Bluetooth.

2.1.2.2 – Camada Física da Tecnologia Bluetooth

A figura abaixo indica a relação entre a pilha de protocolos Bluetooth com a camada física. A camada física é a primeira camada do modelo OSI de sete camadas e é responsável pelo transporte de bits entre sistemas adjacentes pelo ar. A descrição desta camada é a seguinte:

- Receber um stream de bits da subcamada MAC e transmiti-lo via ondas de rádio para uma estação associada.
- Receber ondas de rádio de uma estação associada e convertê-lo em um stream de bits que é passado para a subcamada MAC.

Isto reflete o escopo limitado da porção física de rádio da arquitetura IEEE 802.15.1. Bits e ondas de rádio são transmitidos, mas esta camada não faz nenhuma interpretação.

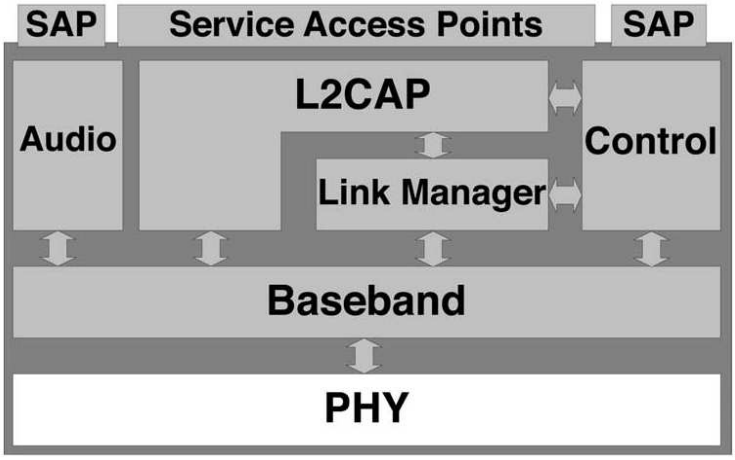

Figura 2.6 – Interface de relacionamento da camada física.

2.1.2.3 – Descrição Geral

Bluetooth é um link de radio de curto alcance com intuito de substituir o cabo na conexão de aparelhos portáteis ou fixos. As características principais são robustez, baixa complexidade, baixa potência e baixo custo.

A banda que o Bluetooth opera é a ISM de 2,4 GHz não licenciada. Para combater interferência e fading, aplica-se um transceiver com saltos de freqüência. A modulação usada é FM binária. A taxa de símbolos é 1 Msímbolos/s. O canal é dividido em slots de tempo de comprimento nominal 625 μs. TDD é usado para emular transmissão full-duplex. A informação é transmitida por pacotes. Cada pacote é transmitido numa freqüência diferente. Um pacote geralmente cobre um único slot, mas pode cobrir até 5 slots.

O protocolo Bluetooth usa uma combinação de comutação por circuito e por pacote. Os slots podem ser reservados para pacotes síncronos. Bluetooth suporta um canal assíncrono, até 3 canais de voz síncronos ou um canal que suporta simultaneamente transmissões assíncronas e síncronas. Cada canal de voz suporta um canal síncrono de 64kb/s em cada direção. O canal assíncrono pode suportar no máximo 723,2 Kbps assimétrico (57,6 Kbps na direção de volta) ou 433,9 Kbps simétrico.

O sistema Bluetooth consiste de uma unidade de rádio, uma unidade de controle de link e uma unidade de suporte de gerenciamento de link e funções de interface do terminal host.

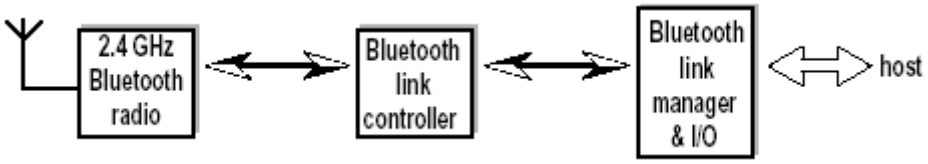

Figura 2.7 – Diagrama de blocos Bluetooth.

O sistema Bluetooth provê conexão ponto-a-ponto ou ponto-multiponto (figura 2.8) Em conexões ponto-multiponto, o canal é compartilhado entre várias unidades Bluetooth. Duas ou mais unidades dividindo o mesmo canal formam uma piconet. Uma unidade Bluetooth age como o master da piconet, enquanto as demais atuam como slaves. Uma piconet pode ter até 7 slaves. Mais slaves podem se manter sincronizados com o master entrando no estado park. Esses slaves em park não podem ser ativos no canal, apenas permanecem sincronizados. Tanto para os slaves ativos quanto para os em park, o acesso ao canal é controlado pelo master.

Várias piconets com áreas de cobertura sobrepostas formam uma scatternet. Cada piconet pode possuir apenas um master. Slaves podem participar de várias piconets numa multiplexação por divisão de tempo. Um master em uma piconet pode ser slave de outra. As piconets não podem ser sincronizadas na freqüência. Cada uma deve possuir uma seqüência de saltos diferentes.

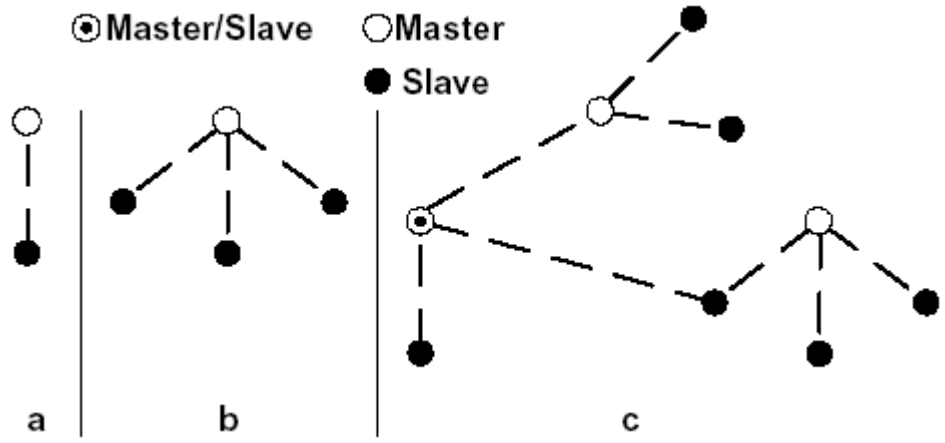

Figura 2.8 – Conexões entre dispositivos Bluetooth.

2.1.2.4 – Canal Físico

O canal é representado por uma seqüência pseudo-aleatória de saltos de freqüência, saltando pelos 79 ou 23 canais RF. A seqüência de salto é única para cada piconet e é determinada pelo endereço do dispositivo master. A fase dos saltos é determinada pelo clock Bluetooth do master. O canal é dividido em slots de tempo e cada um corresponde a uma freqüência da seqüência de saltos. Saltos consecutivos correspondem a diferentes freqüências. A taxa nominal de saltos é 1600 hops/s. Todas as unidades Bluetooth são sincronizadas com o clock e o salto do canal.

O canal é dividido em slots de tempo, cada um com 625 μs de comprimento. Os slots são numerados de acordo com o clock do master da piconet O intervalo de numeração dos slots é de 0 a 2^{27-1} e é periódica com período de 2^{27}.

Um esquema TDD é usado onde master e slave transmitem alternadamente (figura 2.9). O master deve começar sua transmissão em slots pares e o slave em ímpares apenas. O pacote deve começar a ser transmitido no começo do slot. Pacotes transmitidos pelo master ou pelo slave podem ocupar até 5 slots.

A freqüência durante a transmissão do pacote deve permanecer constante. Para um único pacote, a freqüência a ser usada é determinada pelo valor atual do clock Bluetooth. Para um pacote que ocupa vários slots, a freqüência é determinada pelo valor do clock no primeiro slot do pacote. A freqüência no primeiro slot depois de um pacote de vários slots é determinada pelo valor atual do clock Bluetooth. A figura 2.9 ilustra a definição de salto em pacotes com um slots e a figura 2.10 com vários slots.

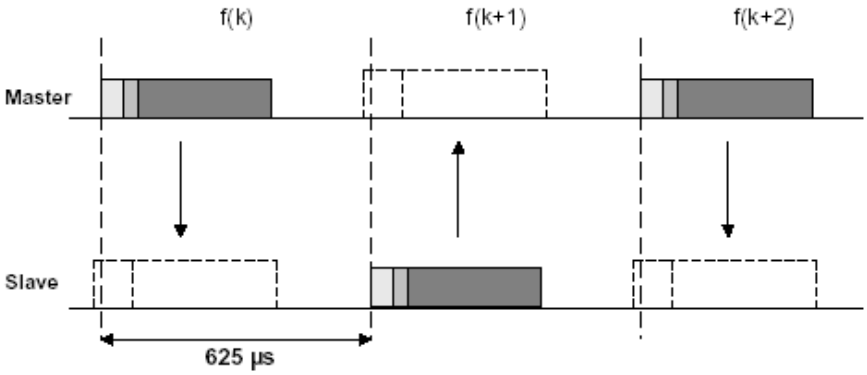

Figura 2.9 – TDD e Temporização.

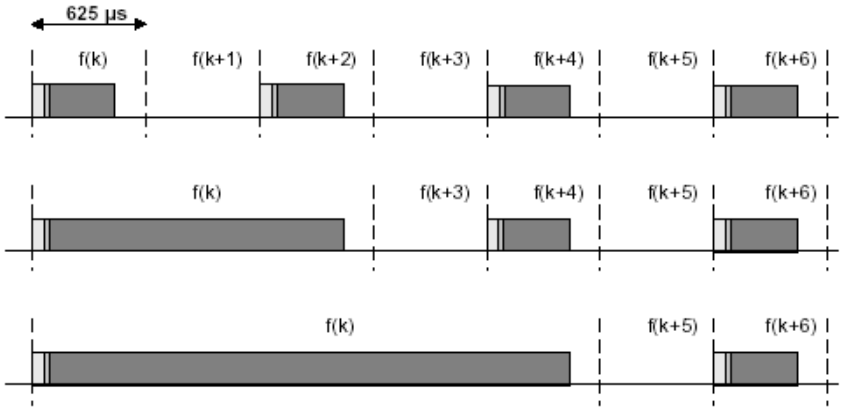

Figura 2.10 – Pacotes Multislot.

2.1.2.5 – Canal de Controle

Esta seção tem como objetivo descrever como funciona o estabelecimento de um canal em uma *piconet* e como os dispositivos Bluetooth podem entrar ou sair dessas redes, mostrando os diversos estados de operação das unidades Bluetooth para suportar tais funções. Além disso, discute-se a operação de várias *piconets* compartilhando uma mesma área, a chamada *scatternet*.

- **Definição mestre-escravo (*master-slave*)**

O canal em uma piconet é caracterizado pelo dispositivo mestre (*master*) da *piconet*. O dispositivo Bluetooth mestre determina a seqüência de saltos (*FH hopping sequence*), o código de acesso do canal, a fase da seqüência de saltos, a temporização, além de controlar o tráfego no canal.

Por definição, o dispositivo mestre é aquele que inicia a conexão para um ou mais dispositivos escravos (*slaves*). Estes dispositivos mestre e escravo são idênticos, ou seja, qualquer unidade Bluetooth pode se tornar o mestre de uma *piconet*.

- **Clock do Bluetooth**

Todos os dispositivos Bluetooth possuem um *clock* interno à uma freqüência de 3,2 kHz que determina a temporização do transceptor.

A temporização e a freqüência de saltos no canal em uma *piconet* é determinada pelo *clock* do Bluetooth da unidade mestre. Quando a *piconet* é estabelecida, o *clock* mestre é comunicado aos escravos, que por sua vez adicionam um *offset* aos seus *clocks* nativos para ficarem sincronizados com o *clock* mestre.

- **Visão Geral dos Estados**

A figura 2.11 mostra um diagrama de estados que ilustra os diferentes estados utilizados no controle de enlace do Bluetooth. Existem dois estados principais: STANDBY e CONNECTION, além de sete sub-estados que são utilizados para adicionar novos escravos à piconet: page, page scan, inquiry, inquiry scan, master response, slave response, e inquiry response. A mudança de um estado para outro pode ser feita através de comandos do gerenciador de enlace do Bluetooth (*link manager*) ou de sinais interno do controle de enlace (*link controller*).

- **Estado Standby**

Este é o estado padrão de uma unidade Bluetooth, que se apresenta em um modo de baixa potência.

O estado de Standby pode ser deixado para se fazer um *scan* por mensagens de *page* ou *inquiry*, ou para fazer um procedimento de *page* ou *inquiry*.

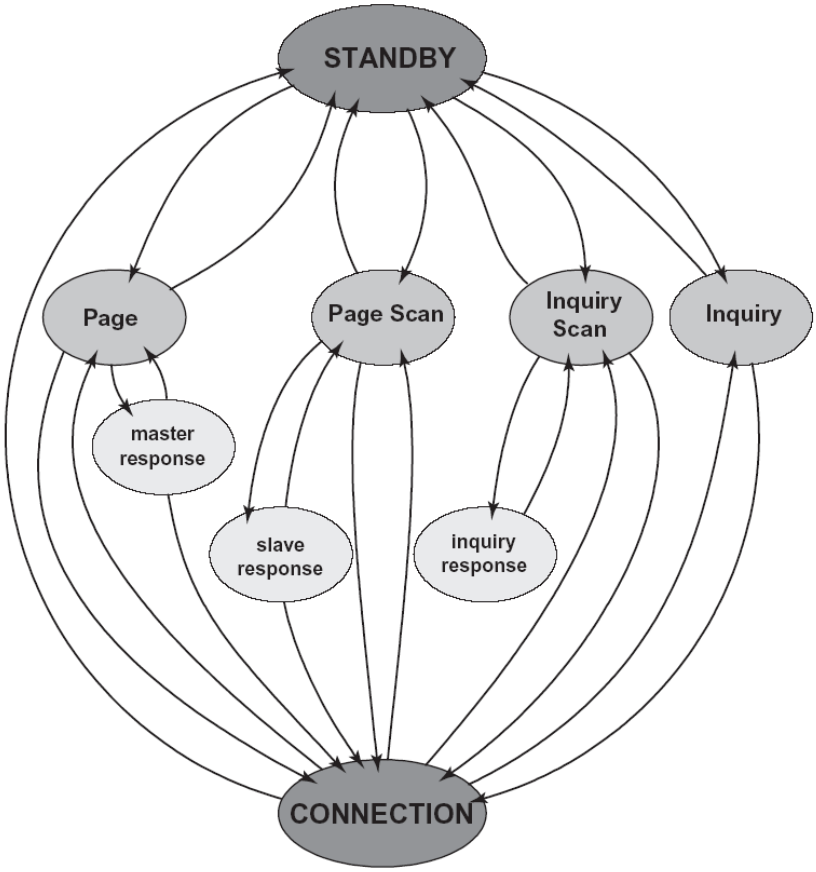

Figura 2.11 – Diagrama do Controle de Enlace do Bluetooth.

Procedimentos de Acesso

- **Geral**

Para estabelecer novas conexões são utilizados os procedimentos de *inquiry* e *paging*. O procedimento de *inquiry* habilita uma unidade para descobrir quais unidades estão na sua área de cobertura, quais os endereços de seus dispositivos e quais os seus clocks. Com o procedimento de *paging*, uma conexão pode, realmente, ser estabelecida. Apenas o endereço do dispositivo Bluetooth é necessário para configurar uma conexão. Uma unidade que estabelece uma conexão irá realizar o procedimento de *page* e, automaticamente, será o mestre da conexão.

- **Page scan**

No sub-estado de *page scan*, uma unidade escuta pelo seu próprio código de acesso durante o tempo da janela de *scan* (*scan window*). Durante esta janela, a unidade escuta em um único salto de freqüência. A janela de *scan* deve ser grande o suficiente para fazer o scan em 16 freqüências de *page*.

Pode-se chegar no sub-estado de *page scan* tanto pelo estado de *Standby* quanto pelo estado *Connection*. No estado de *Standby*, nenhuma conexão foi estabelecida

e, portanto, o dispositivo pode utilizar toda a capacidade para realizar o *page scan*. Antes de ir do estado *Connection* para o sub-estado de *page scan*, a unidade reserva capacidade para realizar o *scan*. Se necessário, a unidade pode colocar as conexões ACL no modo de HOLD ou até mesmo no modo de PARK. Já as conexões SCO não são interrompidas pelo *page scan.*

- **Page**

O sub-estado *page* é utilizado pelo mestre (fonte) para ativar e se conectar à um escravo (destino) que "acorda" periodicamente no sub-estado de *page scan*. O mestre tenta capturar o escravo transmitindo repetidamente o código de acesso do dispositivo (*device access code – DAC*) do escravo em diferentes canais. Como os clocks do Bluetooth do mestre e do escravo não estão sincronizados, o mestre não sabe quando exatamente o escravo "acorda" e nem mesmo a sua freqüência. Portanto, o mestre envia uma seqüência de DACs nas diferentes freqüências e escuta nos intervalos de envio até que receba uma resposta do escravo.

Pode-se chegar no sub-estado de *page scan* tanto pelo estado de *Standby* quanto pelo estado *Connection*. No estado de *Standby*, nenhuma conexão foi estabelecida e a unidade pode utilizar toda a capacidade para realizar o *page*. Antes de ir do estado *Connection* para o sub-estado de *page*, a unidade deve liberar tanta capacidade quanto for possível para fazer o *scan*. Para assegurar isso, é recomendado que as conexões ACL sejam colocadas em estado de HOLD ou PARK. Entretanto, as conexões SCO não devem ser interrompidas pelo *page*.

- **Procedimento de resposta ao page**

Quando uma mensagem de *page* é recebida com sucesso pelo escravo, há uma sincronização entre o mestre e o escravo. Ambos entram em uma rotina de resposta para trocar informações vitais para continuar com a configuração da conexão. É importante para a conexão da piconet que ambas as unidades Bluetooth utilizem o mesmo código de acesso ao canal, a mesma seqüência de saltos no canal, e que seus clocks estejam sincronizados. Estes parâmetros são derivados da unidade mestre, que é definida como sendo a unidade que começa a conexão (inicia o *paging*) e é mantida somente enquanto durar a piconet. O código de acesso e a seqüência de saltos do canal são derivados do endereço Bluetooth do dispositivo mestre. A temporização é determinada pelo clock do mestre. Um *offset* é adicionado ao clock nativo do escravo para sincronizar, temporariamente, o clock do escravo com o clock do mestre. No início, os parâmetros do mestre são transmitidos do mestre para o escravo.

- **Resposta do escravo (*Slave response*)**

Depois de ter recebido seu próprio código de acesso do dispositivo, o escravo transmite uma mensagem de resposta. Esta mensagem consiste, novamente, do código de acesso do dispositivo.

Depois de ter enviado a mensagem de resposta, o receptor do escravo é então ativado e fica esperando a chegada de um pacote FHS. Quando chega esse pacote FHS, o escravo retorna uma mensagem para reconhecer a recepção deste pacote e, finalmente, entra no estado *Connection*. A partir de então, o escravo utilizará o clock e o endereço Bluetooth do mestre para determinar a seqüência de saltos e o código de acesso do canal. O modo de conexão inicia com um pacote POLL transmitido pelo mestre. O escravo responde com qualquer tipo de pacote. Se o pacote POLL não for recebido pelo escravo, ou o pacote de resposta do escravo não for recebido pelo mestre, dentro de um dado número de slots após o reconhecimento do pacote FHS, o mestre e o escravo retornam aos sub-estrados de *page* e *page scan*, respectivamente.

- **Resposta do mestre (*Master response*)**

Quando o mestre recebe uma mensagem de resposta do escravo, este entra na rotina de resposta do mestre. O mestre transmite um pacote FHS contendo o clock do seu Bluetooth, o endereço Bluetooth do seu dispositivo, os bits de paridade BCH, e a classe do dispositivo. O pacote FHS contém toda a informação necessária para construir o código de acesso do canal.

Depois que o mestre enviou o pacote FHS, este espera por uma segunda resposta do escravo, que é a resposta de reconhecimento da recepção do pacote FHS. Se não for recebida nenhuma resposta, o mestre retransmite o pacote FHS com o clock atualizado. O mestre retransmite até que uma segunda resposta do escravo seja recebida, ou o *timeout* seja alcançado. Neste último caso, o mestre retorna para o sub-estado de *page*.

Se a resposta do escravo for recebida, o mestre finalmente entra no estado *Connection*. O mestre então envia um pacote POLL, o qual o escravo deve responder com qualquer tipo de pacote. Se o pacote POLL não for recebido pelo escravo ou o pacote de resposta não for recebido pelo mestre dentro de um dado número de slots, o mestre e o escravo retornam aos sub-estados de *page* e *page scan*, respectivamente.

Procedimentos de Inquiry

- **Geral**

No sistema Bluetooth, um procedimento de inquiry é utilizado em aplicações onde o endereço do dispositivo de destino é desconhecido pela fonte. Alternativamente, o procedimento de inquiry pode ser utilizado para descobrir outros dispositivos bluetooth que estejam na área de cobertura. Durante o sub-estado de inquiry, a unidade que está procurando coleta os endereços dos dispositivos Bluetooth e os clocks de todas as unidade que responderem à mensagem de inquiry, podendo então, se desejado, fazer uma conexão para qualquer um dos dispositivos através do procedimento de page.

A unidade que quer descobrir outras unidades Bluetooth entra no sub-estado de inquiry, onde transmite continuamente a mensagem de inquiry em diferentes freqüências. Uma unidade que se permite ser descoberta entra regularmente no sub-estado de inquiry scan para responder à mensagens de inquiry.

- **Inquiry scan**

Neste estado o receptor realiza scans para procurar código de acesso de inquiry o tempo suficiente para examinar 16 freqüências de inquiry.

Pode-se chegar no sub-estado de *inquiry scan* tanto pelo estado de *Standby* quanto pelo estado *Connection*. No estado *Standby*, nenhuma conexão foi estabelecida e, portanto, a unidade pode utilizar toda a capacidade para este procedimento. Antes de ir do estado *Connection* para o sub-estado inquiry scan, a unidade reserva tanta capacidade quanto for possível para realizar o scan. Se necessário, a unidade coloca as conexões ACL no modo HOLD ou PARK, mas as conexões SCO não são interrompidas.

- **Inquiry**

O sub-estado de inquiry é utilizado pela unidade que quer descobrir novos dispositivos. Entre as transmissões de inquiry, o receptor Bluetooth realiza scans por mensagens de resposta ao inquiry. Quando encontrado, todo o pacote de resposta (que é na verdade um pacote FHS) é lido, e então a unidade continua com as transmissões de inquiry. A unidade Bluetooth em um sub-estado de inquiry não reconhece as mensagens de resposta ao inquiry. Este estado continua até que seja parado pelo gerenciador do enlace (Bluetooth *link manager*) quando este decide que já tem um número suficiente de respostas, ou quando se alcança um *timeout*.

Pode-se chegar no sub-estado de *inquiry* tanto pelo estado de *Standby* quanto pelo estado *Connection*. No estado *Standby*, nenhuma conexão foi estabelecida e, portanto, a unidade pode utilizar toda a capacidade para este procedimento. Antes de ir do estado *Connection* para o sub-estado inquiry scan, a unidade deve liberar tanta capacidade quanto for possível para realizar o scan. Para assegurar isso, é recomendado que a unidade coloque

as conexões ACL no modo HOLD ou PARK, mas as conexões SCO não devem ser interrompidas.

- **Inquiry response**

Para a operação de inquiry, tem somente uma resposta do escravo, não tem resposta do mestre. O mestre escuta por respostas nos intervalos de envio de mensagens de inquiry, mas depois de ler a resposta ele continua transmitindo as mensagens de inquiry. Quando uma mensagem de inquiry é recebida no sub-estado de inquiry scan, uma mensagem de resposta contendo o endereço do destinatário deve ser retornado. Esta mensagem de resposta é um pacote FHS carregando parâmetros do dispositivo Bluetooth.

Um problema pode ocorrer quando diversas unidades Bluetooth estão próximas da unidade que realiza o inquiry e todas respondem à mensagem de inquiry simultaneamente. Porém, é altamente improvável que as todas as unidades utilizem a mesma fase da seqüência de saltos do inquiry. Entretanto, para evitar colisões entre unidade que "acordem" no mesmo canal de inquiry simultaneamente, utiliza-se um protocolo na resposta de inquiry do escravo que é capaz de contornar tal problema.

- **Estado *Connection***

No estado *Connection*, a conexão está estabelecida e pacotes podem ser trocados entre as unidades. Em ambas as unidades, o código de acesso do canal (mestre) e o clock do Bluetooth do mestre são utilizados. O mestre inicia sua transmissão nos slots pares, enquanto o escravo inicia sua transmissão nos slots ímpares.

Os primeiros pacotes de informações no estado *Connection* contém mensagens de controle que caracterizam o enlace e dão maiores informações sobre as unidades Bluetooth.

A saída do estado *Connection* é dada por um comando de *reset* ou *detach*. O comando *detach* é utilizado se o enlace foi desconectado de maneira normal. Toda a configuração de dados no controlador de enlace (*link controller*) ainda é válida. O comando *reset* reinicia todos os processos, portanto o controlador deve ser reconfigurado.

As unidades Bluetooth podem se encontrar em diversos modos de operação enquanto estiverem no estado *Connection*: modo ativo, modo *sniff*, modo *hold*, e o modo *park*.

- **Modo ativo**

No modo ativo, a unidade Bluetooth participa ativamente no canal. O mestre agenda a transmissão baseado nas demandas de tráfego dos diferentes escravos. Além disso, suporta transmissões regulares para manter os escravos sincronizados no canal.

- **Modo sniff**

Se um escravo participa de um enlace ACL, ele tem que escutar em todos os slots ACL do tráfego do mestre. Com o modo sniff, os slots de tempo onde o mestre pode iniciar uma transmissão para um escravo específico é reduzido, ou seja, o mestre só pode iniciar a transmissão em slots de tempo especificados. Esses slots sniff são espaçados de forma regular com um intervalo de T_{sniff}.

Modo hold

Durante o estado *Connection*, o enlace ACL para um slave pode ser colocado em modo hold. Isto significa que o escravo temporariamente não suporta pacotes ACL no canal. Com o modo hold, a capacidade do canal pode ser liberada para fazer outras coisas como scanning, paging, inquiring, ou atender a outras piconets. A unidade que está no modo hold também pode entrar em um modo de espera de baixa potência. Durante o modo hold, o escravo ainda mantém o seu endereço de membro ativo (AM_ADDR).

Antes de entrar no modo hold, o mestre e o escravo entram em um acordo para determinar a duração de tempo na qual o escravo permanecerá neste modo.

- **Modo park**

Quando o escravo não precisa participar do canal da piconet, mas ainda quer continuar sincronizado com o canal, ele pode entrar no modo park que é um modo de baixa potência com pouquíssima atividade no escravo. No modo park, o escravo entrega o seu endereço de membro ativo. Por outro lado, ele recebe dois novos números para serem utilizados no modo park:

- PM_ADDR: Parked Member Address (endereço de membro no modo park)
- AR_ADDR: Access Request Address (endereço de pedido de acesso)

O PM_ADDR serve para diferenciar um escravo em park dos outros escravos em park. Este endereço é utilizado no procedimento de saída do modo park inicializado pelo mestre. O AR_ADDR é utilizado pelo escravo no procedimento de saída do modo park inicializado pelo próprio escravo.

O escravo em park “acorda” em intervalos regulares para escutar o canal para re-sincronizar e verificar as mensagens de broadcast. Para suportar a sincronização e o acesso ao canal pelos escravos em park, o mestre suporta mais um canal: *beacon channel*.

Além de utilizar o modo park para um consumo de baixa potência, ele também pode ser utilizado para conectar mais de sete escravos a um único mestre, porém, somente sete escravos podem estar no modo ativo. Não há limitação para o número de escravos em modo park.

- **Scatternet**

Múltiplas piconets podem cobrir uma mesma área. Desde que cada piconet tenha um mestre diferente, as piconets trabalham independentemente, cada uma com sua própria seqüência de saltos do canal e fase determinados pelo seu respectivo mestre.

Se múltiplas piconets cobrem a mesma área, uma unidade pode participar de duas ou mais piconets sobrepostas. Uma unidade Bluetooth pode agir como um escravo em várias piconets, mas apenas como um mestre em uma única piconet. Um grupo de piconets nas quais existem conexões entre as diferentes piconets é chamada de scatternet.

- **Gerenciamento de potência**

Recursos são incluídos no Bluetooth para assegurar uma operação em baixa potência.

Para minimizar o consumo de potência, a manipulação de pacotes é minimizada tanto no lado do transmissor quanto do receptor. No lado do transmissor a potência é minimizada através do envio apenas de dados úteis. Isso significa que se necessitar trocar somente informações de controle de enlace, pacotes NULL serão utilizados. No lado do receptor, o processamento de pacotes é feito em diferentes passos. Se nenhum código de acesso for encontrado na janela de busca, o transceptor volta a “dormir”.

O tipo do pacote indica quantos slots o pacote deve ocupar. Dessa maneira um escravo não endereçado no primeiro slot pode voltar a dormir nos slots restantes que aquele pacote ocupar.

Além disso, ainda têm os modos do estado *Connection* que reduzem o consumo de potência, são eles: sniff, hold e park.

- **Supervisão de enlace**

Uma conexão pode cair devido a várias razões como um móvel se movendo para fora do raio de alcance. Como isso pode acontecer sem nenhum aviso prévio, é importante monitorar o enlace tanto no mestre quanto no escravo para evitar possíveis colisões quando o AM_ADDR for designado para outro escravo.

Para isso, o mestre e o escravo utilizam temporizadores de supervisão de enlace. Sempre que um pacote chega corretamente, este temporizador é reiniciado. Quando este temporizador atinge o tempo limite, a conexão é reiniciada. [1, 2, 3, 4]

2.2 - JAVA

2.2.1 – Socket

Sockets em Java permitem que dois aplicativos, na mesma máquina ou não, comuniquem-se enviando dados um para o outro.

Dados na Internet são geralmente transmitidos na forma de datagramas. Cada um desses datagramas possui um cabeçalho e um campo para dados. Neste caso, são usados protocolos não orientados a conexão, ou seja, cada datagrama é enviado independentemente e não existe garantia de que ele será entregue corretamente. Além disso, o remetente não recebe notificação sobre o recebimento.

Quando se faz uma aplicação distribuída, geralmente se torna necessário que o aplicativo remetente receba notificação de recebimento, os destinatários precisam ser capazes de pedir retransmissão de dados caso haja erros e precisam conhecer a ordem dos mesmos.

Algumas aplicações distribuídas, geralmente as que tratam de trafego em tempo real, não precisam deste controle sobre o trafego. Elas precisam mais da velocidade que protocolos não orientados a conexão conseguem fornecer do que um rígido controle de erros.

Para o Sistema de Gerenciamento e Distribuição de Mensagens de Texto, fez-se necessário o uso de protocolos orientados a conexão, uma vez que controle de erro é imprescindível neste tipo de aplicação. Para isso, usa-se TCP. Este protocolo da camada de Comunicação prove controle de tráfego.

O protocolo TCP/IP é quase sempre implementado como parte do sistema operacional. Assim, os Sockets em Java acessam essas funcionalidades dos sistemas para prover conexão entre aplicativos.

As classes e interfaces que suportam sockets em Java estão no pacote java.net. As classes mais importantes são:

- Socket: permite conexão entre dois processos conhecidos. As duas aplicações devem ter um instance de Socket para se comunicarem. A classe que realmente faz o trabalho de criar uma conexão e SocketImpl.
- ServerSocket: gerencia o handshake entre um cliente e um servidor.

Para se estabelecer uma conexão entre um cliente e um servidor, primeiramente deve-se conhecer o endereço IP e a porta do servidor. Endereços IPs especificam um computador na rede ou na Internet. Já portas são números inteiros que especificam uma conexão com um host conhecido.

Quando o numero de clientes esperando conexão excede o tamanho máximo da fila, o servidor passa a recusar conexões com novos clientes.

Cada conexão é gerenciada com um objeto socket.

Algumas questões se tornam importantes quando transmitimos dados: o canal pode não ser seguro e as comunicações podem requerer muita banda. Uma maneira de resolver este problema é usar uma subclasse de Socket. Dessa forma, pode-se customizar os fluxos com cifração e compressão. [5, 6]

2.2.2 – RMI

A Estrutura básica

Ao desenvolver aplicações distribuídas baseadas em sockets, vê-se que o código necessário para tal se enquadra dentro de cinco categorias:

- O código que contém a lógica de negócio. De fato, esse é o código e o que justifica a existência da aplicação;
- A interface gráfica do cliente;
- Código que lida com a serialização e desserialização dos dados a serem enviados pelo meio. Esse código é tratado pelas classes que estendem os output e input streams. É uma parte um tanto quanto trabalhosa da aplicação, pois todo esse processo depende do que está sendo enviado pelo canal e deve ser baseado em algum protocolo. Consiste em uma parte significante do software;
- Código que inicia e configura a aplicação;

- Código cujo único propósito é tornar a aplicação distribuída mais robusta. Isso geralmente envolve, verificação de tipos comuns de erros na parte cliente, para evitar a sobrecarga do servidor. Dessa maneira é possível tratar melhor a questão do load-balancing, distribuindo a carga entre vários servidores com mais eficiência.

Os dois primeiros itens devem ser desenvolvidos em qualquer tipo de aplicação. Caso contrário a aplicação não seria necessária. O terceiro e o quarto itens são diferentes. A maioria do código necessário para desenvolvê-los pode ser escrito automaticamente, sem muito esforço. Escrever código que lida diretamente com os streams de bytes pode um tanto tedioso e tem tendência a erros.

O RMI é uma tecnologia que facilita o desenvolvimento do terceiro e quarto itens, pois gera todo o código necessário automaticamente.

Chamada de métodos remotos

Pode-se dizer que o RMI foi desenvolvido para que dois programas Java, rodando em máquinas virtuais distintas, podem se comunicar como se estivessem fazendo uma chamada a um método local.

O RMI se baseia em dois tipos de objetos gerados pelo compilador *rmic*. Esses arquivos devem ser gerados pelo servidor, e são chamados de *stubs* e *skeletons*. Um *stub* é um objeto que simula um objeto do servidor na máquina virtual cliente. Ele implementa os mesmos métodos do objeto do servidor, mantém automaticamente uma conexão socket aberta com a JVM onde o objeto do servidor está residente e é responsável pelo processo de serialização e desserialização na JVM cliente. O *skeleton* é um objeto responsável por manter as conexões de rede e por serializar e desserializar os dados na JVM servidora.

O procedimento pelo qual um cliente se comunica com um servidor pode ser resumido pelos seguintes passos.

- O Cliente obtém uma instância da classe *Stub*. O *stub* é automaticamente gerado no servidor e implementa todos os métodos que a classe do servidor implementa;
- O Cliente chama um método do *stub*. Essa chamada de fato é a mesma caso o cliente fizesse uma chamada a um método qualquer de um objeto residente na mesma máquina virtual;
- Internamente, o *stub* cria uma conexão *socket* como *skeleton* no servidor ou aproveita uma conexão existente. Ele serializa toda a informação associada à chamada do

método, incluindo o nome do método e os argumentos, e envia essas informações pelo *socket* para o *skeleton*;

- O *skeleton* desserializa essas informações, reconstituindo os parâmetros dos métodos, sejam eles tipos primitivos ou objetos. O retorno do método é então serializado e enviado de volta para o *stub*;
- O *stub* por sua vez, desserializa os dados e o encaminha para o código cliente.

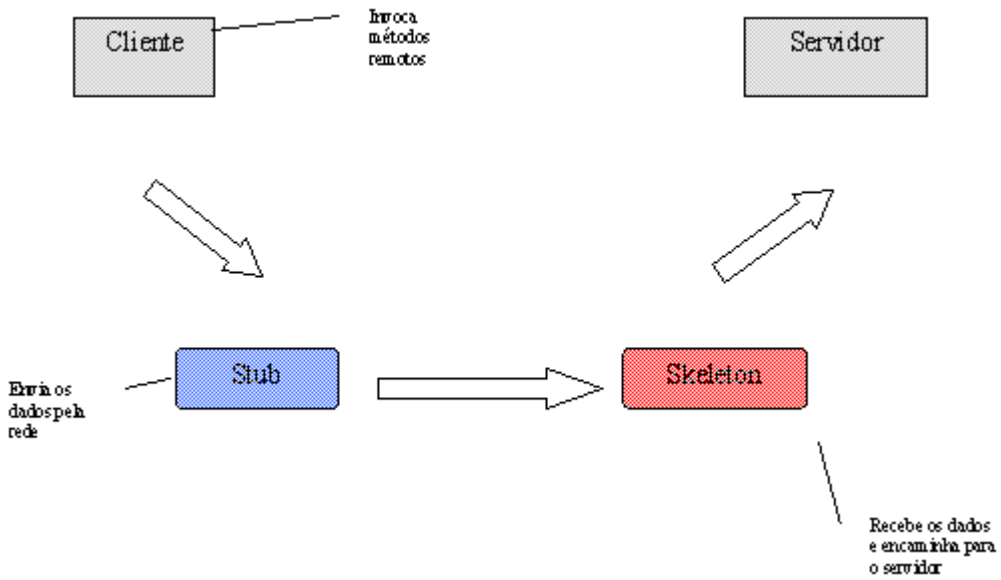

Figura 2.12 – Funcionamento do RMI.

Um ponto importante a ser observado é que todas as informações que trafegam na rede devem ser serializáveis. Isso inclui os objetos.

Para que um objeto seja serializável, sua classe deve implementar a interface `java.io.Serializable`. Essa interface não contém métodos, sua utilidade é de marcar os objetos da classe como serializáveis. È claro que implementar essa interface simplesmente não garante que se possa converter uma objeto em uma seqüência de bytes. De fato, todos os atributos dessa classe, sejam eles tipos primitivos ou objetos devem também ser serializáveis, de modo que o objeto possa ser completamente reconstruído no outro lado da conexão.

Segurança e RMI

Como foi dito, o *stub* representa o objeto do servidor na JVM cliente. Existem duas maneiras para que o cliente obtenha esses *stubs*. A primeira é mais simples (porém

pouco prática) é: o desenvolvedor do servidor, após gerar *stubs* por meio do compilador RMI, deve entregá-lo ao responsável pelo aplicativo cliente para que ele inclua esse arquivo no mesmo diretório da interface remota.

Esse método é um tanto quanto primitivo, pois sempre que o servidor sofrer atualizações, os *stubs* devem ser distribuídos a todas as aplicações cliente, uma por uma.

A segunda e melhor forma é o cliente fazer o *download* desse arquivo sempre que instanciar um objeto remoto. O problema é: o que garante que o arquivo baixado é confiável e veio realmente do servidor desejado? Um *hacker* poderia implementar a interface remota e produzir um *stub* que substituísse o *stub* original, enganando o cliente.

Quando um servidor RMI é iniciado, uma estrutura chamada de *RMI Registry* também é iniciada. Nessa estrutura, ficam armazenados todos os *stubs* gerados pelo servidor, e de fato é nela que o cliente deve procurar por eles. Para que o sistema se torne seguro, é necessário garantir que os objetos disponíveis no *registry* sejam autênticos.

Uma solução é permitir toda tentativa de dispor um objeto no registro somente ao mesmo processo que roda o *registry*. Isso não evita que *hackers* vasculhem o sistema e descubram quais objetos estão no registro, ou quais métodos são chamados, mas evita que os *stubs* sejam substituídos no registro.

De fato, para se usar a opção de baixar o *stub* do servidor, deve-se editar as políticas de segurança da máquina virtual, além de alguns arquivos de configuração. Essa opção é mais trabalhosa, mas com certeza é a mais escalonável em termos de engenharia de software. [6, 7]

2.2.3 – Servlets e JSP´s

A tecnologia Servlet é a base do desenvolvimento de aplicativo Web usando a linguagem de programação Java. A API de Servlets, criada em 1996 com o intuito de gerar dinamicamente código HTML, é um componente fundamental da plataforma J2EE. Ela é uma das tecnologias Java mais importantes, e é a tecnologia subjacente para uma outra tecnologia Java popular para desenvolvimento de aplicativo: Java Server Pages (JSP).

O servlet é uma classe Java que pode ser automaticamente carregada em e executada por um servidor Web especial. Esse servidor Web cliente do servlet é chamado

de um *Container servlet*. Servlets interagem com clientes por meio de um modelo solicitação-resposta baseado em HTTP. Porque a tecnologia servlet trabalha sobre HTTP, um Container servlet precisa suportar HTTP como o protocolo para solicitações de cliente e respostas de servidor. Entretanto, um Container servlet também pode suportar protocolos semelhantes, tal como HTTPS (HTTP sobre SSL) para transações seguras.

Um aplicativo servlet também pode incluir conteúdo estático, tais como páginas HTML e arquivos de imagem. Permitir que o Container servlet sirva conteúdo estático não é preferível, pois o conteúdo é mais rápido se servido por um servidor HTTP mais potente, tal como colocar um servidor Web na frente, para gerenciar todas as solicitações do cliente. O servidor Web serve o conteúdo estático e passa aos Containers servlet todas as solicitações de cliente servlets.

2.2.3.1 – Como Funciona um Servlet

Um servlet é carregado pelo Container servlet na primeira vez que o servlet é solicitado. Então, a ele é encaminhada a solicitação do usuário, ele a processa e retorna a resposta ao Container servlet que, por sua vez, envia a resposta de volta ao usuário. Depois disso, o servlet permanece na memória aguardando outras solicitações – ele não é descarregado da memória, a menos que o Container de servlet veja uma diminuição de memória. No entanto, cada vez que o servidor é solicitado, o Container de servlet compara o carimbo de horário do servlet carregado com o arquivo de classe servlet. Se o carimbo de horário de arquivo de classe for mais recente, o servlet é recarregado na memória. Dessa maneira, você não precisa reiniciar o Container servlet sempre que atualizar o seu servlet.

2.2.3.2 – O Container Servlet Tomcat

Atualmente estão disponíveis vários Containers servlets. O mais popular – e aquele reconhecido como o Container servlet/JSP oficial – é o Tomcat. Originalmente projetado pela Sun Microsystems, o código fonte Tomcat foi entregue à Apache Software Foundation, em outubro de 1999. No Apache, o Tomcat foi incluído como parte do Projeto Jakarta, um dos projetos da Apache Software Foundation.

O próprio Tomcat é um servidor Web. Isso significa que você pode utilizar Tomcat para solicitar serviços HTTP em servlets, assim como arquivos estáticos (HTML, arquivos

de imagem, etc). No entanto, na prática, como ele é mais rápido em solicitação não servlet, não JSP, normalmente Tomcat é usado como um módulo, com outro servidor Web mais robusto, como o servidor Web Apache ou Microsoft Internet Information Server. Apenas solicitações para servlets ou páginas JSP são passadas para Tomcat.

2.2.3.3 – Java Server Pages (JSP)

Java Server Pages é uma outra tecnologia Web para desenvolver aplicativos Web. JSP foi lançada quando a tecnologia de servlet tinha atingido popularidade como uma das melhores tecnologias disponíveis. JSP, porém, não se destina a substituir servlet. Na verdade, JSP é uma extensão da tecnologia de servlet, e é prática comum usar ambas, servlet e páginas JSP nos mesmos aplicativos Web.

De acordo com o Web site da Sun, "A tecnologia JSP é uma extensão da tecnologia servlet, criada para suportar autoria de páginas HTML e XML". A JSP funciona quase como qualquer outra página, sendo normalmente acessada através de um cliente navegador mas com a única diferença que o código Java será executado no servidor.

JSP é uma página da Web que contém código Java junto com HTML usando tags especiais (<% e %>). A idéia de colocar código de linguagem de programação junto com HTML não é tão nova. A Microsoft possui o ASP (Active Server Pagens) enquanto o Netscape o SSJS (Server-Side Javascript), usando código baseado em visualbasic e javascript respectivamente.

O Web Container interpreta o arquivo JSP, o compila e transforma em um servlet. Assim sendo, logo que o arquivo JSP é chamado pela primeira vez por um cliente, um servlet que o representa é criado, aplicando todos os benefícios do mesmo para uma página JSP. [6, 8, 9]

2.2.4 – Struts

2.2.4.1 – Definição

O Struts é um framework para desenvolvimento de aplicações Web. Define-se um framework como um conjunto de classes, interfaces e utilitários que, além de prover

soluções para problemas semelhantes, tem como maior finalidade aumentar a produtividade dos profissionais envolvidos no desenvolvimento de aplicações, poupando-os de tarefas repetitivas e suscetíveis a erros.

No desenvolvimento de aplicações Web, a mistura de código Java com HTML e JavaScript pode trazer muitos problemas no desenvolvimento, questão que se agrava quando aplicação possui um design gráfico complexo e funcionalidades avançadas, como é o caso desse Projeto Final de Graduação. Torna-se difícil a manutenção do código e a interação entre a equipe de desenvolvimento e a de design gráfico fica compremetida.

Um passo importante é padronizar a arquitetura das aplicações, usando as melhores práticas do mercado e evitando partir do zero. É nesse ponto que entram os frameworks Web, entre os quais o Struts do projeto Apache Jakarta é o mais conhecido e respeitado. Além de ser software livre e bem documentado (tanto na Web como em diversos livros), o Struts é flexível, extensível e suportado pelas principais ferramentas de desenvolvimento (entre elas Eclipse, Jbuilder, Jdeveloper, NetBeans e IDEA).

O Struts é baseado em uma variação do tradicional padrão de arquitetura MVC (Model – View - Controller) e se integra bem, na camada de apresentação, com outras tecnologias e frameworks web tais como JSP, Jakarta Velocity e XSTL. Mais detalhes da arquitetura MVC será dado posteriormente. Para a camada de dados, as opções são igualmente amplas: de JDBC a Enterprise JavaBeans. É a arquitetura do Struts que o torna tão extensível e adaptável.

2.2.4.1 – Como Funciona o Struts

O ponto de partida do Struts é o Servlet org.apache.struts.ActionServlet, responsável por gerenciar o fluxo de dados e requisições dos usuários entre as camadas de visualização (*View*) e lógica de negócio (*Controller*) da aplicação. O ActionServlet é o único servlet de uma aplicação construída com o Struts. O arquivo *web.xml*, que é o descritor de implementação de uma aplicação Web, deve conter um mapeamento genérico, direcionando páginas diferentes da aplicação para esse servlet. Em geral é utilizado o padrão de URL (*URL pattern*) “*.do”.

O framework fornece um controlador constituído por três elementos principais: o **ActionServlet** (servlet controlador), o **RequestProcessor** e as classes de ação, ou **Actions**. O ActionServlet e o RequestProcessor são fornecidos prontos pelo framework; as ações devem ser implementadas pelo desenvolvedor estendendo a classe abstrata Action. São as

ações as responsáveis por acessar o modelo, fazendo a ligação entre a lógica de negócio e o servlet.

De modo similar, o Struts fornece os **ActionForms**, JavaBeans criados pelo desenvolvedor, que têm suas propriedades preenchidas automaticamente pelo Struts a partir de formulários HTML. Os ActionsForms fazem a ponte entre a visão (páginas JSP) e o controlador (ações e servlet).

Além desses, o Struts inclui uma série de elementos adicionais importantes, como um pool de conexões (configurado no arquivo *struts-config.xml*), e uma vasta biblioteca de tags, contemplando praticamente todas as tarefas necessárias no desenvolvimento de páginas JSP.

Quando o ActionServlet recebe uma requisição, ela é repassada para o RequestProcessor, que processa e valida os dados recebidos. No arquivo *struts-config.xml*, são mapeadas as URLs para as ações responsáveis pelo seu processamento. Escolhida a ação, ela é instanciada e em seguida é chamado seu método **execute**, que realiza chamadas ao modelo. Ao final da execução, a ação retorna uma indicação para o controlador sobre qual visão deve ser exibida.

Os ActionsForms também são listados no *struts-config.xml* e estes devem ser instanciados e populados para cada URL. Estes são inseridos como atributos da requisição ou da sessão HTTP, estando portanto disponíveis tanto para as ações responsáveis por processar a requisição quanto para as páginas JSP responsáveis por exibir o resultado.

2.2.4.1 – Fluxo Básico

A figura abaixo apresenta o fluxo de controle de uma aplicação baseada no Struts

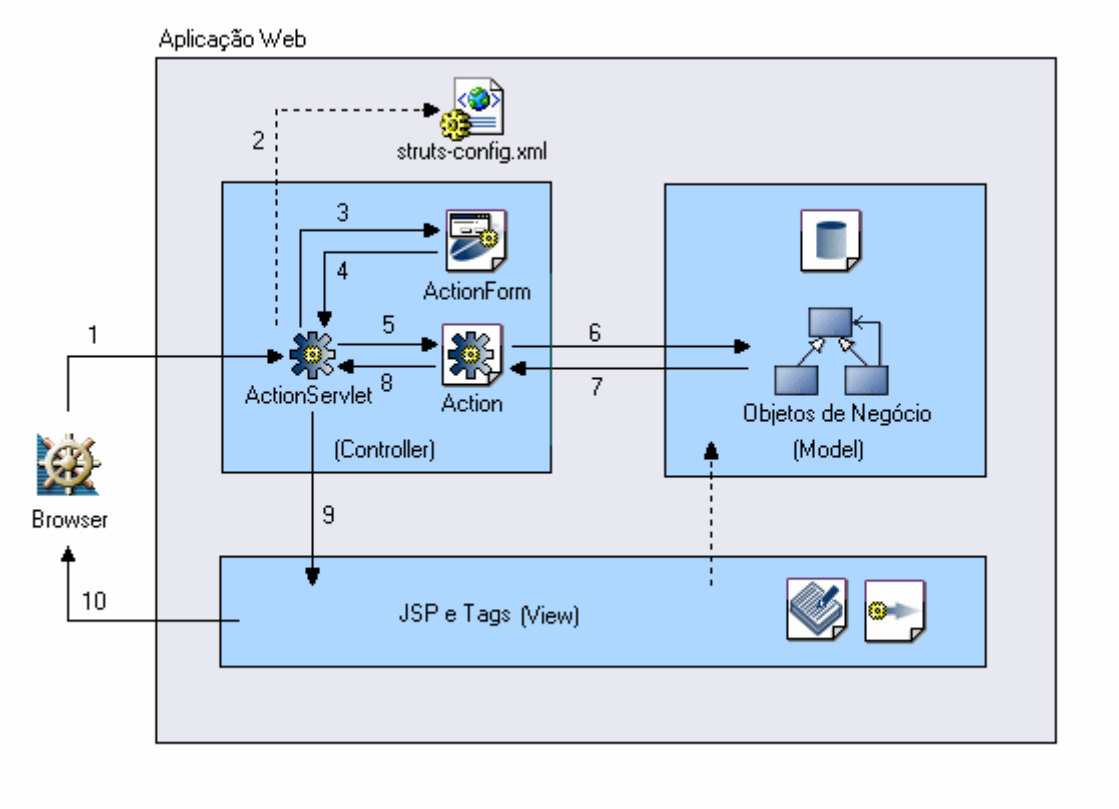

Figura 2.13 – Fluxo de navegação nos componentes do Struts.

1. O usuário faz uma solicitação através de uma url no browser. Ex: http://localhost:8080/cadastro/listUsers.do. Note que no final da url tem um .do que será usado para invocar (na verdade mapear) o servlet controller da struts.
2. Se for a primeira solicitação que o Container recebeu para esta aplicação, ele irá invocar o método init() da ActionServlet (controller da Struts) e irá carregar as configurações do arquivo struts-config.xml em estruturas de dados na memória. Vale lembrar que esta passagem só será executada uma única vez, pois nas solicitações subsequentes, a servlet consulta estas estruturas na memória para decidir o fluxo a ser seguido.
3. Baseado no fluxo definido no arquivo struts-config.xml, e que neste momento já se encontra carregado em estruturas na memória, o ActionSerlet identificará qual o ActionForm (classe para a validação dos dados) irá invocar. A classe ActionForm através do método **validate** irá verificar a integridade dos dados que foram recebidos na solicitação que vem do browser.
4. O controle da aplicação é retomado pelo ActionServlet, que verifica o resultado da verificação do ActionForm.

- Se faltou alguma coisa (campo não preenchido, valor inválido, etc), o usuário recebe um formulário HTML (geralmente o mesmo que fez a solicitação), informando o motivo do não atendimento da solicitação, para que o usuário possa preencher corretamente os dados para fazer uma nova solicitação.
- Se não faltou nenhuma informação, ou seja, todos os dados foram enviados corretamente, o controller passa para o próximo passo (Action).

5. O ActionServlet, baseado no fluxo da aplicação (estruturas já carregadas em memória) invoca uma classe Action. A classe Action passará pelo método **execute** que irá delegar a requisição para a camada de negócio.
6. A camada de negócio irá executar algum processo (geralmente popular um bean, ou uma coleção). O resultado da execução deste processo (objetos já populados) será usado na camada de apresentação para exibir os dados.
7. Quando o controle do fluxo da aplicação votar ao Action que invocou o processo da camada de negócio, será analisado o resultado, e definido qual o mapa adotado para o fluxo da aplicação. Neste ponto, os objetos que foram populados na camada de negócio serão "atachados" como atributos na seção do usuário.
8. Baseado no mapeamento feito pelo o Action, o Controller faz um forward para o JSP para apresentar os dados.
9. Na camada de apresentação (View), os objetos que foram setados como atributos da sessão do usuário serão consultados para montar o HTML para o browser.
10. Chega o HTML da resposta requisitada pelo usuário.

O Controller já vem implementado na Struts, embora, caso seja possível estendê-lo a fim de adicionar funcionalidade. O fluxo da aplicação é programado em um arquivo XML através das ações que serão executadas. As ações são classes base implementadas pela framework seguindo o padrão MVC. Assim devemos estendê-las a fim de adicionar a funcionalidade desejada.

A geração da interface é feita através de custom tags, também já implementadas pela Struts, evitando assim o uso de Scriptlets (códigos Java entre <%> e <%>), deixando o código JSP mais limpo e fácil de manter. [10, 11, 12, 13]

2.3 - BANCO DE DADOS

2.3.1 – JDBC

JDBC é uma API (Application Programming Interface) Java que fornece uma solução para o acesso multiplataforma a banco de dados. Praticamente toda a arquitetura JDBC é baseada em interfaces e especificações, e é isso que torna a API tão geral e padronizada para bancos de dados tão distintos como o MySQL, Oracle e Microsoft SQLServer.

O acesso a dados com JDBC é baseado em drivers JDBC, que são implementados por conjuntos de classes que conhecem intimamente a arquitetura e os comandos do banco dados.

Um fato interessante é que os drivers JDBC são implementados pelos fabricantes dos bancos. A Sun entra apenas com as especificações (e até estas são desenvolvida em conjunto com outras empresas). Isso garante que as implementações dos drivers sejam eficientes e leves.

A maioria das empresas fabricantes de banco de dados fornecem drivers JDBC gratuitamente junto com seus produtos e muitas disponibilizam esses drivers na Internet para download gratuito. O driver utilizado no aplicativo deste Projeto Final foi o *mysql-connector-java-3.1.8a* fornecido na página da empresa do banco de dados utilizado no Projeto, o MySQL. [6, 9, 14]

2.3.2 - MySQL

Um dos mais rápidos programas para servidores de SQL (*Structured Query Language*), hoje no mercado, é o MySQL, desenvolvido pela T.c.X. DataKonsultAB. Este programa está disponível para download em sua versão original em www.mysql.com, em inglês, e também em www.mysql.com.br para sua versão brasileira, onde encontram−se projetos de tradução e documentação do MySQL em português.

Além de oferecer vários recursos não existentes em outros servidores, o MySQL tem a vantagem de ser totalmente gratuito para uso tanto comercial, quanto privado, em conformidade com a licença pública GPL.

As principais metas da equipe de desenvolvimento do MySQL é construir um servidor rápido e robusto.

Os recursos acima mencionados incluem:

− Capacidade de lidar com um número ilimitado de usuários;

− Capacidade de manipular mais de cinqüenta milhões (50.000.000) de registros;

− Execução muito rápida de comandos, provavelmente o mais rápido do mercado;

− Sistema de segurança simples e funcional.

Devido a sua popularidade, ser totalmente gratuito, bem documentado que o MySQL foi escolhido para este aplicativo do Projeto Final de Graduação. [9, 14]

2.4 - DESIGN PATTERNS

Os padrões J2EE são um conjunto de soluções com base no J2EE para problemas comuns. Eles refletem os conhecimentos e experiências coletivas de arquitetos Java no Sun Java Center, obtidas a partir de inúmeras execuções bem-sucedidas de empreendimentos do J2EE. O Sun Java Center esteve planejando soluções para a plataforma J2EE desde o seu início, com a finalidade de obter características do Quality of Service (QoS) como escabilidade, disponibilidade, desempenho, segurança, confiabilidade e flexibilidade.

Esse padrões descrevem problemas típicos encontrados pelos desenvolvedores de aplicações de empresas e fornecem soluções para esses problemas. Os padrões capturam a essência dessas soluções e representam o refinamento de soluções que acontecem ao longo do tempo e a partir de experiências coletivas. Para colocar de outra maneira, elas extraem as questões essenciais de cada problema, oferecendo soluções que representam em refinamento aplicável de teoria e prática.

Os padrões de projeto, ou design patterns, vêm despertando interesse na comunidade de projetistas de software por proporcionar elementos que conduzem ao reaproveitamento de soluções para projetos, e não apenas à reutilização de código. Os padrões de projeto permitem evidenciar os aspectos essenciais de problemas comuns, levando a sua compreensão mais ampla e orientando a construção de sistemas que exibam efetivamente as qualidades desejadas. Com a implementação de projeto em Java, pode-se verificar as vantagens de sua utilização, ao mesmo tempo que passa-se a conhecer soluções robustas para certos problemas e esse foi um dos motivo que procurou-se implementar esses design patterns no aplicativo do Projeto Final. [10, 14, 15]

2.4.1 – MVC (*Model-View-Controller*)

O pattern **Model-View-Controller (MVC)** reforça um projeto modular e de fácil manutenção e força a separação de camadas além de permitir a separação do modelo de dados das várias formas que o dado possa ser acessado e manipulado. Um sistema MVC é dividido em um modelo de dados, um conjunto de visões e um conjunto de controladores. As visões fornecem a interface do usuário e, em uma aplicação padrão, consistem de JSP. O controlador é geralmente implementado como um ou mais servlets Java. O modelo consiste de um código que fornece acesso direto ao banco de dados relacional.

MVC é útil principalmente para aplicações grandes e distribuídas onde dados idênticos são visualizados e manipulados de formas variadas. Como o MVC facilita a divisão de trabalho por conjuntos de habilidades, este pattern é bastante adequado para empresas de desenvolvimento que suportam desenvolvimento modular e concorrente com muitos desenvolvedores. Promovendo a portabilidade de interfaces e do back-end, o pattern MVC também torna fácil testar e manter suas aplicações. A chave para o MVC é a separação de responsabilidades. As visões podem usar o modelo de dados para exibir resultados, mas elas não são responsáveis por atualizar o banco de dados. Os controladores são responsáveis por selecionar uma visão apropriada e por fazer alterações no modelo de dados. O modelo, por sua vez, é responsável por representar os dados base da aplicação. Algumas vezes o modelo de dados também incluirá a lógica de negócio (as regras para manipular os dados de negócio), e algumas vezes a lógica de negócio existirá na camada do controlador.

Na verdade existem quatro componentes em uma aplicação MVC, não três, já que o cliente é uma parte integrante de toda a operação. O cliente envia uma requisição para o servlet, que interage com o modelo de dados para efetuar a ação requisitada pelo cliente. O controlador então passa informações para uma visão, que as formata de acordo com as necessidades do cliente. Dependendo dos tipos de clientes suportados, uma atividade pode ter muitas visões. Desacoplar as visões do processo de requisição torna mais fácil criar novas visões para novos formatos. Uma aplicação poderia apresentar uma interface HTML e depois adicionar visões WML (para dispositivos wireless) e visões XML (para Web services) futuramente.

As aplicações Struts seguem o design pattern MVC. Os três componentes principais são o Controller de servlet, as Java Server Pages (a View) e a lógica de negócios da aplicação (o Model). [10, 15, 16, 17, 18, 19]

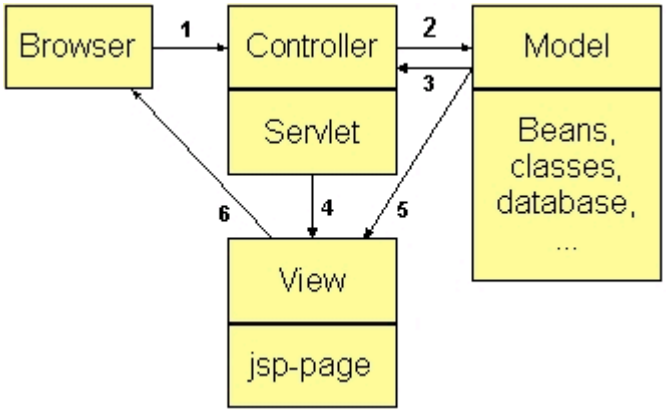

Figura 2.14 – O padrão MVC.

2.4.2 – VO (*Value Object*)

2.4.2.1 – Problema

As aplicaçõe J2EE implementam componentes de negócios do lado do servidor como beans de sessão e beans de entidade. Alguns métodos expostos pelo componentes de negócios retornam dados para o cliente. Frequentemente, o cliente chama os métodos get de um objeto de negócios várias vezes até obter todos os valores dos atributos.

Um bean é uma classe Java muito simples onde se segue algumas convenções de codificação simples. Um bean funciona como um container para uma coleção de propriedades que ou se ajusta através de métodos "setters" ou lê-se através de métodos "getters". Ajustar alguns ou todas as propriedades em um bean geralmente permitirá que o bean faça algumas processamentos como por exemplo, ler um banco de dados e retornar os resultados.

Os beans de sessão representam os serviços de negócios e não são compartilhados entre os usuários. Os beans de entidade, por outro lado, são objetos transacionais e multi-usuários representando dados persistentes. Um bean de entidade expõe os valores dos atributos fornecendo um método de acesso (também referenciado como um método getter ou get) para cada atributo que ele deseja expor.

À medida que a utilização desses métodos remotos aumenta, o desempenho da aplicação pode diminuir significativamente. Portanto, utilizar chamadas múltiplas para

obter métodos que retornam valores de atributos únicos é ineficiente na obtenção de valores de dados de um enterprise bean.

Desse modo se uma aplicação cliente necessita trazer grande quantidade de informações de um serviço remoto ela terá como problema um custo de rede e performance extremamente alto se for buscar uma informação por vez.

2.4.2.2 – Solução

Utilizar um Value Object para encapsular os dados de negócios. Uma única chamada de método é utilizada para enviar e recuperar o objeto de dados. Quando o cliente solicita ao enterprise bean para os dados de negócios, o enterprise bean pode criar objeto de dados, preenchê-lo com seus valores de atributos e passá-lo por valor para o cliente.

Os clientes normalmente requerem mais de um valor de um enterprise bean. Para reduzir o número de chamadas remotas e evitar a sobrecarga associada, é melhor utilizar objetos de daos para transportar os dados do enterprise bean para o seu cliente.

Quando um enterprise bean utiliza um objeto de dados, o cliente realiza uma única chamada de método remota para o enterprise bean para solicitar o objeto de dados, em vez de inúmeras chamadas de método remotas para obter valores de objeto de dados, copia valores para o objeto e retorna para o cliente. O cliente recebe o objeto de dados para obter valores de atributo individuais do objeto de dados. Como o objeto de dados é passado pelo valor para o cliente, todas as chamadas para a instância do objeto de Daos são chamadas locais em vez de chamadas de métodos remotas. [17, 18]

2.4.3 – DAO (*Data Access Object*)

2.4.3.1 – Problema

Muitas aplicações J2EE do mundo real precisam utilizar dados persistentes em algum momento. Para muitas aplicações, o armazenamento persistente é implementado com mecanismos diferentes e há diferenças marcantes nas APIs utilizadas para acessá-los. Outras aplicações talvez precisem acessar dados que residem em sistemas separados.

As aplicações podem utilizar a API JDBC para acessar dados residentes em um sistema de gerenciamento de banco de dados relacional (RDBMS, relational database management system). A API JDBC permite o acesso padrão e o tratamento de dados no

armazenamento persistente, como um banco de dados relacional. A JDBC permite às aplicações J2EE utilizarem declarações SQL, que são meios padrões para acessar as tabelas do RDBMS. Porém, mesmo dentro de um ambiente RDBMS, a sintaxe e o formato real das declarações SQL podem variar dependendo do produto de banco de dados específico.

Há ainda uma variação maior com tipos diferentes de armazenamentos persistentes. Mecanismos de acesso, suportados por APIs e recursos variam entre tipos diferentes de armazenamentos persistentes como o RDBMS, bancos de dados orientados a objetos, arquivos planos e assim por diante. Dessa maneira, origens de dados diferentes oferecem desafios a aplicação e o código de acesso de dados. Mas incluir a conectividade de dados e o código de acesso de dados dentro desses componentes tornam difícil e cansativa a migração da aplicação de um tipo de origem de dados para o outro. Quando a origem de dados muda, os componentes precisam ser alterados para tratar o novo tipo de origem de dados.

2.4.3.2 – Solução

Utilizar um Data Access Object (DAO) para extrair e encapsular todos os acessos à origem de dados. O DAO gerencia a conexão com a origem de dados para obter e armazenar dados.

O DAO implementa o mecanismo de acesso exigido para trabalhar com a origem de dados. A origem de dados poderia ser um armazenamento persistente como um RDBMS. O componente de negócios que conta com o DAO utiliza a interface mais simples exposta pelo DAO para seus clientes. O DAO oculta completamente os detalhes de implementação da origem de dados de seus clientes. Como a interface exposta pelo DAO a sues clientes não se altera quando a implementação da origem de dados subjacentes se altera, esse padrão permite que o DAO se adapte a esquemas diferentes de armazenamento sem afetar seus clientes ou componentes de negócios. Essencialmente, o DAO age como um adaptador entre o componente e a origem de dados. [17, 18]

2.5 – OBEX

OBEX (Object EXchange) é um protocolo de comunicação que permite objetos de dados serem transferidos entre dois dispositivos, sendo que os mesmos podem estar fisicamente conectados ou não. O protocolo OBEX foi inicialmente criado pela Associação de Dados Infravermelhos (IrDA), mas depois se tornou um dos protocolos adotados pelo Bluetooth. A figura abaixo representa uma comparação entre a pilha de protocolos Bluetooth e IrDA:

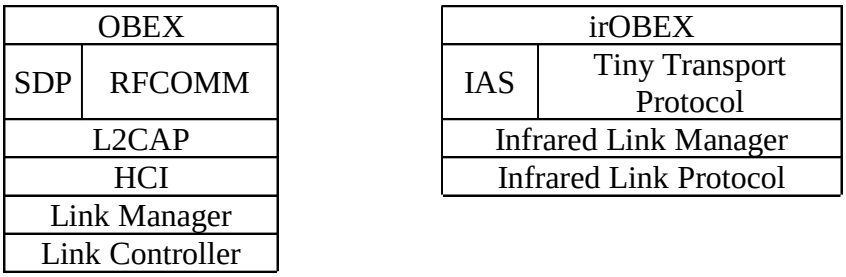

Figura 2.15 – OBEX em Bluetooth e em IrDA.

Na especificação Bluetooth, o OBEX é um protocolo subjacente que é usado para implementar os seguintes perfis:

– Generic Object Exchange Profile

– Object Push Profile

– Synchronization Profile

– File Transfer Profile

– Basic Imaging Profile

– Basic Printing Profile

O protocolo OBEX é composto por uma arquitetura cliente / servidor simples. O cliente OBEX retira e coloca objetos de dados nos servidores OBEX. A definição de OBEX pode ser resumida em duas partes: O *Modelo de Objeto* OBEX, que provê a definição de objetos OBEX e da informação em como transferi-los. E, a *Protocolo de Sessão* OBEX, que define o *handshaking* necessário entre o cliente e o servidor quando os mesmo vão transferir objetos entre dispositivos.

No Modelo de Objeto OBEX, todos os detalhes sobre um objeto são representados por atributos chamados cabeçalhos (*headers*). Foram definidos 17 cabeçalhos para os atributos de objetos OBEX, porém a especificação Java OBEX usa somente 12 deles como constantes da interface `java.obex.HeaderSet`. Ainda é possível criar seus próprios cabeçalhos, desde que siga algumas regras.

O Protocolo de Sessão OBEX especifica todas as regras para a comunicação entre clientes e servidores OBEX. O esquema de comunicação é um simples processo de pedido e resposta: O cliente envia um pedido e o servidor responde. Ambos, pedido e resposta são enviados em pacotes.

O cliente se comunica com o servidor através de oitos operações simples:

- CONNECT
- DISCONNECT
- PUT
- GET
- SETPATH
- ABORT
- CREATE – EMPTY
- PUT – DELETE

O servidor OBEX, por sua vez, responde ao cliente através de códigos de resposta que informam o cliente sobre o status da conexão e transferência.

O cliente inicia o processo de comunicação enviando um pacote de pedido (*request packet*) com uma operação de CONNECT. O servidor responde com um pacote de resposta (*response packet*) que contém o código de resposta, que informa sucesso ou falha no processo.

A operação de PUT permite ao cliente enviar um objeto de dados para o servidor. Os clientes podem recuperar objetos do servidor através de um pacote de pedido GET. A operação de SETPATH é usada em conjunto com a operação de PUT e GET para traçar o diretório do servidor. O cliente pode ainda usar a operação ABORT para finalizar a sessão prematuramente. Com as operações CREATE-EMPTY e PUT-DELETE o cliente cria um arquivo vazio no servidor e remove um objeto do servidor, respectivamente.

3 – O SISTEMA GERENCIADOR DE MENSAGENS

3.1 - INTRODUÇÃO

Desenvolvido para rodar em um ambiente corporativo, o Sistema de Gerenciamento de Mensagens é a primeira parte do projeto, um protótipo e um estudo para validação do

projeto final, utilizando a linguagem Java. Esse protótipo consiste em um sistema distribuído para o gerenciamento e envio de mensagens de texto entre dispositivos móveis. Foi incluída também a possibilidade de enviar email.

O usuário irá solicitar o envio de uma mensagem do seu celular para um destinatário cadastrado no sistema. Este será responsável por definir qual a melhor forma de envio para o destinatário (Bluetooth, SMS ou por email).

3.2 – AS PARTES DO SISTEMA

Foi identificada a necessidade dos seguintes aplicativos para o sistema distribuído:

1. ***Aplicativo do Móvel:*** desenvolvido usando J2ME para ser usado em dispositivos móveis que implementem MIDP 2.0. Para os testes do projeto foi usado o Nokia 6600.

2. ***Access Point (Ponto de Acesso):*** usando J2SE, este aplicativo foi desenvolvido para rodar em computadores com interface Bluetooth e que implementem a pilha de protocolos desenvolvida pelo Atinav.

3. ***Servidor de Banco de Dados***: é o servidor central do sistema e o responsável pelo gerenciamento e envio de mensagens. Foi desenvolvido utilizando-se o J2SE e pode ser instalado em qualquer computador da Rede.

4. ***Aplicativo Desktop***: possui a interface gráfica para o gerenciamento do sistema. É capaz de adicionar ou remover usuários e grupos.

Além desses aplicativos principais existem também:

1. ***Servidor de Email***: desenvolvido em J2SE, é responsável por direcionar as mensagens para um servidor de email qualquer. À época do desenvolvimento do projeto, foi utilizado o servidor do Yahoo! A escolha foi baseada no fato deste servidor permitir o envio de mensagens anônimas. Sendo assim, não é necessário que cada usuário do sistema tenha uma conta no Yahoo!. Eles poderão utilizar suas contas pessoais de email, mesmo que sejam em outros servidores.

2. ***Servidor de SMS***: Um acordo entre a UnB e a Claro tornou possível a criação de uma VPN entre um PC na universidade e o servidor de SMS da empresa. Este aplicativo, instalado no PC, é responsável por encaminhar as mensagens por ele recebidas para serem enviadas por SMS. [5, 6]

3.2.1 – O Servidor

A finalidade da parte servidora é prover recursos e serviços para os demais componentes do sistema, como:

- Comandos de gerência de usuários, grupos e pontos de acesso;
- Comandos de requisição de informações do banco de dados;
- Serviço de envio de mensagens SMS;
- Serviço de envio de email.

O Servidor é formado por alguns componentes como mostra a figura abaixo:

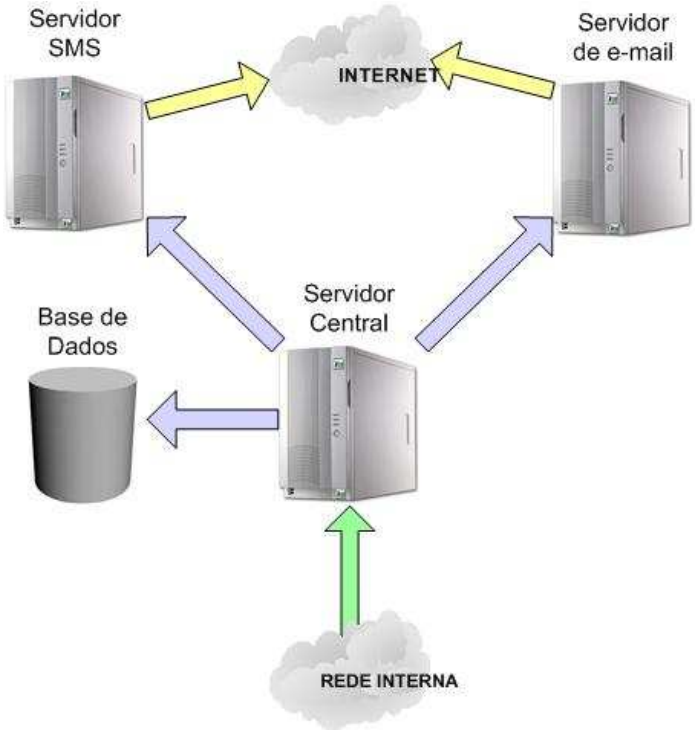

Figura 3.1 - Arquitetura do Servidor.

Servidor SMS

É responsável por enviar mensagens de texto para os celulares dos usuários do sistema. Este servidor está conectado a uma operadora de telefonia móvel através de uma VPN (*Virtual Private Network*) e recebe os pedidos de envio de mensagens vindos do Servidor Central que por sua vez recebe os pedidos vindos da Rede Interna.

Servidor de e-mail

É responsável por enviar e-mail para a caixa de email dos usuários do sistema. Este servidor utiliza o protocolo SMTP (*Simple Mail Transfer Protocol*) de um servidor disponível na Internet (*smtp.mail.yahoo.com.br*) para enviar a mensagem requisitada pelo Servidor Central que por sua vez é requisitado pela Rede Interna.

Apresenta uma interface gráfica onde pode-se configurar o servidor SMTP, como mostrado na figura a seguir:

Figura 3.2 - Interface Gráfica do Servidor de Email.

Base de Dados

Armazena os dados dos Access Points Bluetooth que estão na rede, dos Grupos de usuários e as informações de todos os usuários cadastrados no sistema. A estrutura está mostrada na figura abaixo:

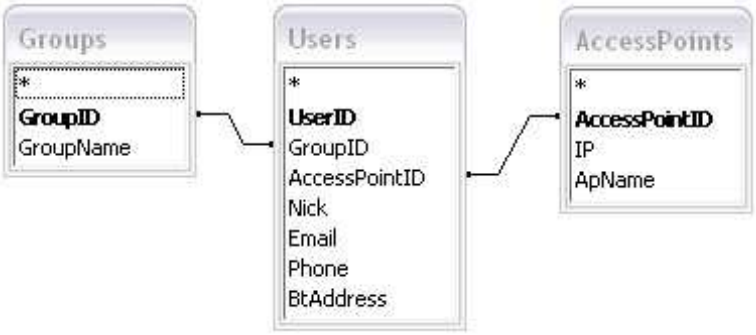

Figura 3.3 - Estrutura da Base de Dados.

Servidor Central

O Servidor Central, como o próprio nome já diz, centraliza toda a parte servidora do sistema. Todos os pedidos de envio de mensagens ou de consulta ao banco de dados são feitos para este servidor que por sua vez encaminha o pedido para outros servidores periféricos ou faz a consulta à base de dados do sistema.

Apresenta uma interface gráfica bastante simples, como mostrada na figura abaixo, de onde se inicia/interrompe a execução do servidor, podendo também visualizar as tabelas da base de dados.

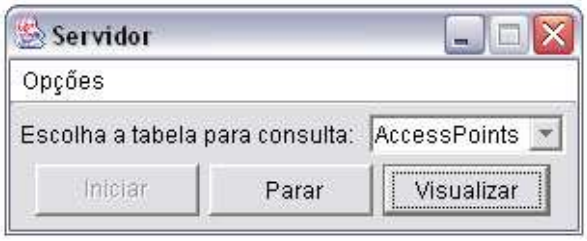

Figura 3.4 - Interface gráfica do Servidor.

É importante notar que esta estrutura pode ser completamente implementada em um único computador, com a base de dados e todos os servidores sendo executados na mesma máquina, ou de forma distribuída, como mostrado na figura 3.1. [5, 6]

3.2.2 – O Desktop

O objetivo deste aplicativo desenvolvido em Java é simplificar as operações de gerência, tais como: inserir e excluir grupos, inserir e excluir usuários e, ainda, enviar mensagens.

A parte gráfica deste aplicativo foi totalmente desenvolvida usando *Swing* (uma das APIs que compõem a JFC - *Java Foundation Classes*). O *Swing* tem um conjunto mais completo de componentes de interface gráfica que o *Abstract Windowing Toolkit* (AWT) e foi escolhido justamente por apresentar mais vantagens em relação ao AWT. Dentre estas vantagens destaca-se a possibilidade de mudar o *look and feel* dos componentes de forma

que estes possam ter a mesma aparência em sistemas operacionais diferentes ou, ainda, que eles tenham a aparência característica de cada sistema operacional.

A primeira tela do aplicativo é mostrada abaixo.

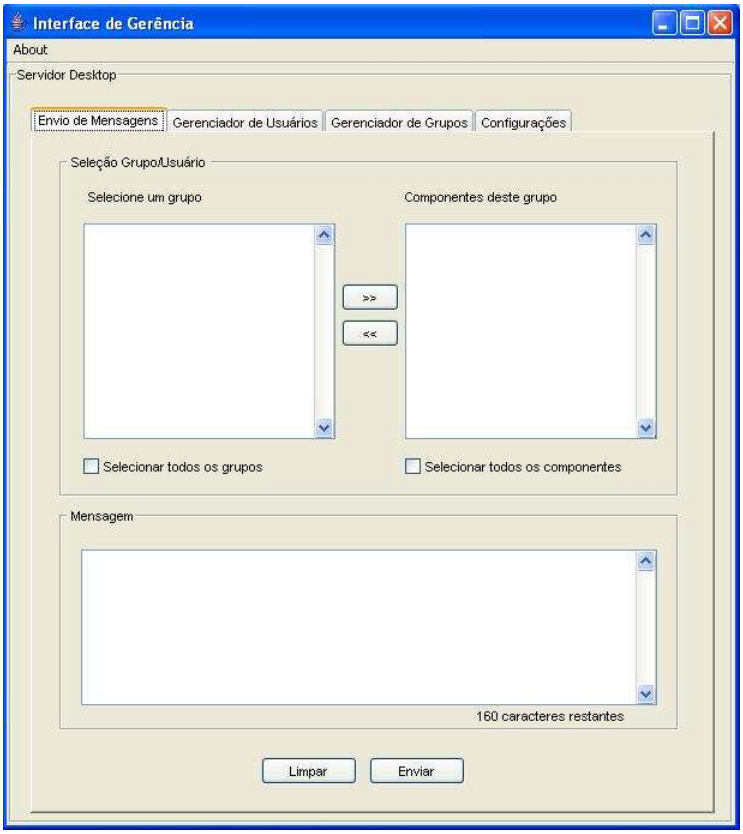

Figura 3.5 – Tela de envio de mensagens.

Esta tela oferece a opção do envio de mensagens. Primeiro é escolhido um ou mais grupos e depois são escolhidos os usuários aos quais se deseja enviar uma mensagem. Os componentes usados nesta tela são relativamente básicos: `JButton`, `JLabel`, `JPanel`, `JCheckBox`, `JList` e `JTextArea` (estes dois últimos juntos com `JScrollPane`).

A próxima figura mostra em primeiro plano a janela para inserir um novo usuário e, em segundo plano, a tabela com os usuários já cadastrados. Para montar a tabela dos usuários foi usado o componente `JTable` que, assim como o `JList`, implementa a arquitetura *delegate-model*.

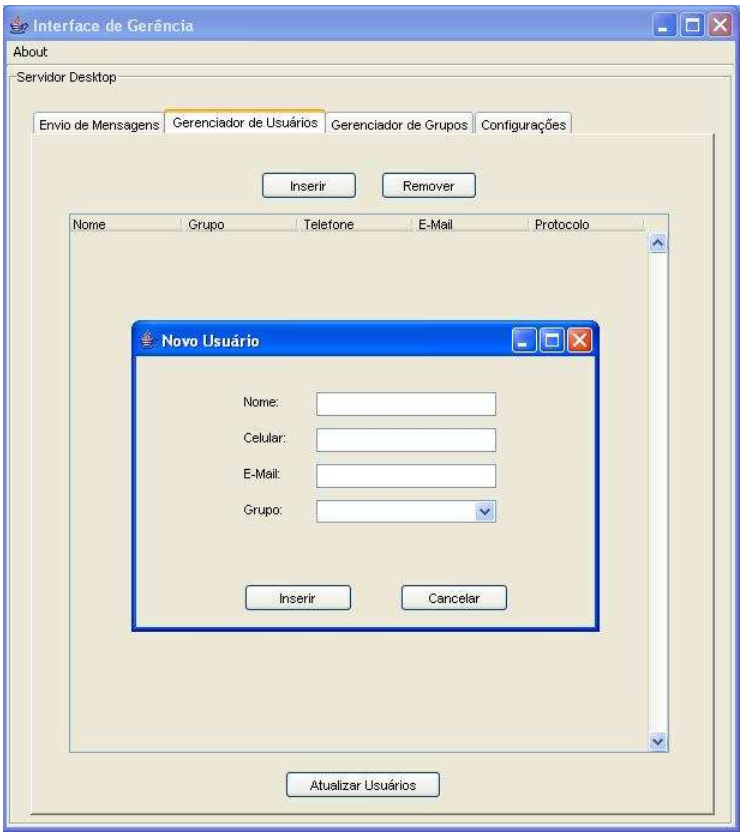

Figura 3.6 – Lista dos usuários e inserção de novos usuários.

Na próxima figura pode ser vista a tela de inserção de um novo grupo e a lista dos grupos existentes. Percebe-se a grande semelhança de figura 3.6 com a figura 3.7.

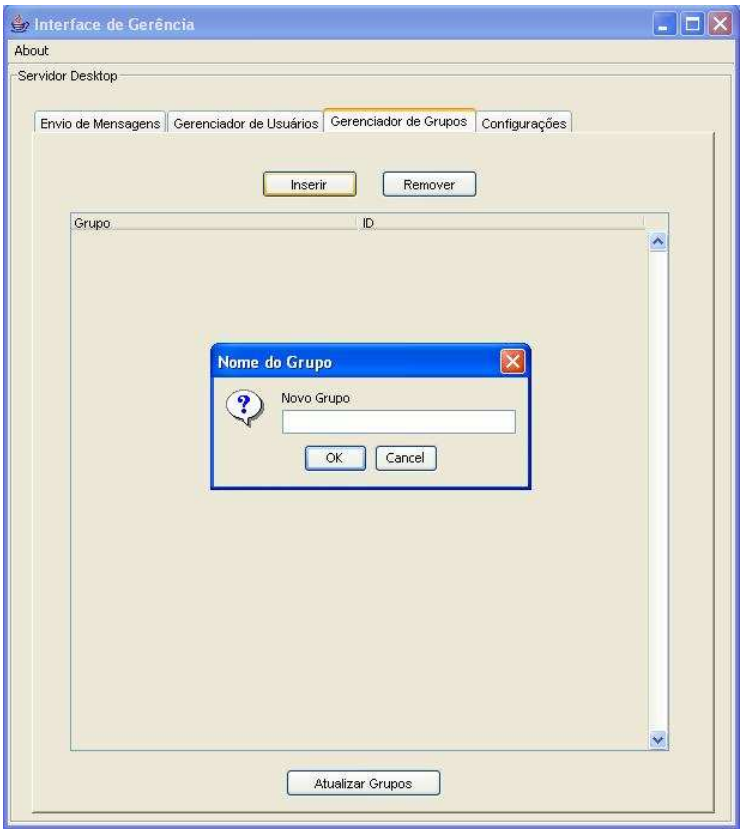

Figura 3.7 – Lista de grupos e inserção de novos grupos.

A última tela do aplicativo é a tela de configurações. Na versão 1.0 do aplicativo a única configuração disponível para o usuário é a escolha do servidor. O aplicativo de gerência busca no servidor principal toda a base de dados dos grupos e usuários e é através de uma interface de comunicação pré-definida que são feitas as inclusões e exclusões. Essa comunicação é feita através de chamadas a métodos remotos com *Remote Method Invocation* (RMI), uma solução em Java para computação distribuída.

Especificando o endereço do servidor e acionando o botão “Atualizar” é feita a busca pelo serviço no endereço especificado pelo usuário. Caso não hajam erros nesta fase, os dados referentes os grupos e usuários são automaticamente atualizados.

Figura 3.8 – Tela de configurações.

O aplicativo apresentado aqui é uma proposta para facilitar a gerência dos usuários do sistema. É uma solução que pode, por exemplo, ajudar uma secretária a enviar mensagens a respeito de reuniões e outras informações de interesse para um grupo de pessoas. Por isso, devido às suas características, pode ser muito útil em um ambiente corporativo. [5, 6]

3. 2.3 – O Access Point Bluetooth

Introdução

Fazendo uma analogia com as redes 802.11, o sistema de "*instant message*" desenvolvido sente a necessidade de um Access Point (AP) que faça um papel de ponte entre as redes 802.15 e a rede local (LAN). Além disso, cada AP deverá saber quais são os dispositivos em seu domínio, de maneira a diminuir o overhead na rede IP (uma vez que o servidor central é acessado pela rede IP) e o delay entre o envio e a chegada da mensagem, pois não será necessário que um AP faça um inquiry sempre que solicitado.

O AP deverá ser capaz, portanto de se comunicar com redes IP e Bluetooth. Usou-se para tanto a linguagem de programação Java e as seguintes tecnologias: Socket, Remote Method Invocation (RMI) e JSR-82 (API Java para Bluetooth).

Descrição do AP

Na figura seguinte, está mostrado um fluxograma desde a solicitação do móvel até o envio do pedido ao servidor.

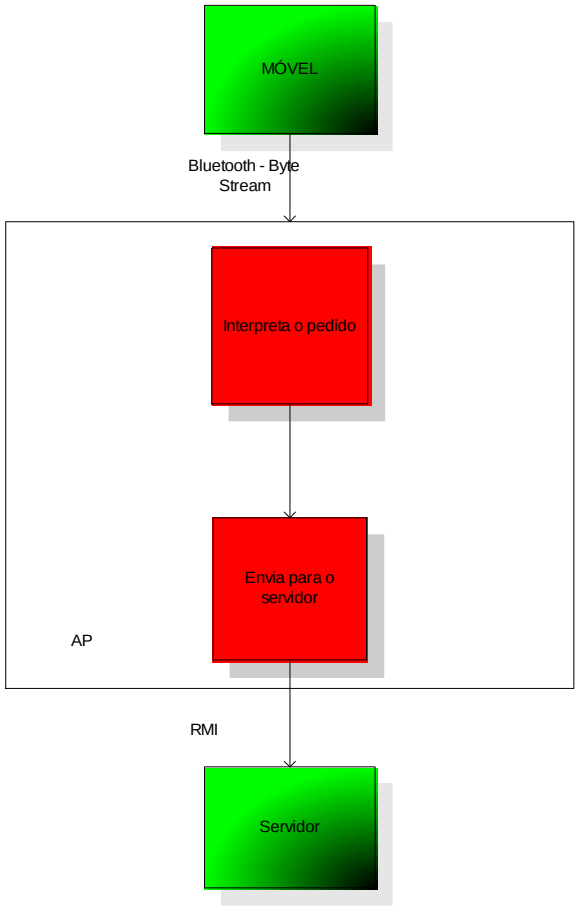

Figura 3.9 – Esquema da comunicação entre AP e Servidor.

O AP recebe as solicitações do móvel que são tratadas no nível da API por array de bytes. Esses bytes devem ser decodificados para que o AP seja capaz de interpretar o pedido do móvel. Para tanto, uma comunicação em um nível tão baixo necessita de um protocolo e de um identificador de *requests* para que o AP possa fazer o pedido corretamente para o servidor. Esse protocolo será detalhado mais adiante.

Ao saber qual foi o pedido do móvel, o AP envia uma solicitação ao servidor usando uma tecnologia já mencionada, chamada de *Remote Method Invocation* ou Invocação de Método Remoto. Como o próprio nome diz, é possível invocar métodos de um objeto que esteja residente na memória de uma máquina remota como se esse objeto fosse local. Nesse caso, essa tecnologia é bastante útil, uma vez que o AP faz inúmeros pedidos ao servidor RMI e esses pedidos se traduzem em vários métodos. Os benefícios que o RMI traz ficarão mais claros quando ele for comparado com os *Sockets*.

Como já foi dito o papel do AP não se limita a somente encaminhar as solicitações do móvel ao servidor. Ele também desempenha um papel fundamental, na medida em que mantém uma base de dados, atualizada constantemente, com os usuários sobre seu raio de ação. Quando o AP realiza uma busca periódica por dispositivos Bluetooth, ele comunica ao servidor quais são os novos móveis encontrados ou quais os móveis que não foram mais encontrados, ou seja, que não estão em seu raio de ação. Assim, o servidor mantém em uma base de dados todos os usuários do sistema e sua localização (seu AP). Dessa maneira quando um servidor recebe um pedido de algum AP “A” para enviar mensagem para um móvel “B”, o servidor faz uma consulta na base de dados e verifica em que AP o móvel “B” se encontra, fazendo assim o pedido direta e unicamente a quem interessa. Ou seja, estando o móvel no AP “C” o servidor envia a mensagem a ser entregue ao móvel “B” diretamente para o AP “C”. Assim, quando o AP “C” recebe a mensagem para entregar ao móvel, ele já sabe que este móvel encontra-se em seu raio de ação. Porém, se o móvel de destino estiver no mesmo AP que o móvel de origem (remetente) não é necessário que o pedido de envio de mensagem suba até o servidor. A próprio AP, consultando a sua base de dados local identifica o móvel de destino e se encarrega do envio da mensagem.

O AP por dentro

Ao receber um pedido por Bluetooth, o AP o encaminha para a classe Protocol, responsável por interpretar as solicitações. Ao interpretá-las, essa classe cria um objeto específico para cada tipo de pedido e o encaminha para um método da classe EventDispatcher que seja capaz de tratá-lo. Note que a classe Protocol implementa uma

espécie de roteamento com os diversos tipos de pedidos, pois, para cada um, invoca um método específico de EventDispatcher. Essa última, por sua vez, tem uma referência para o objeto remoto capaz de receber as solicitações. Essa classe é a única que faz interface com o objeto remoto *CommunicationServer* .

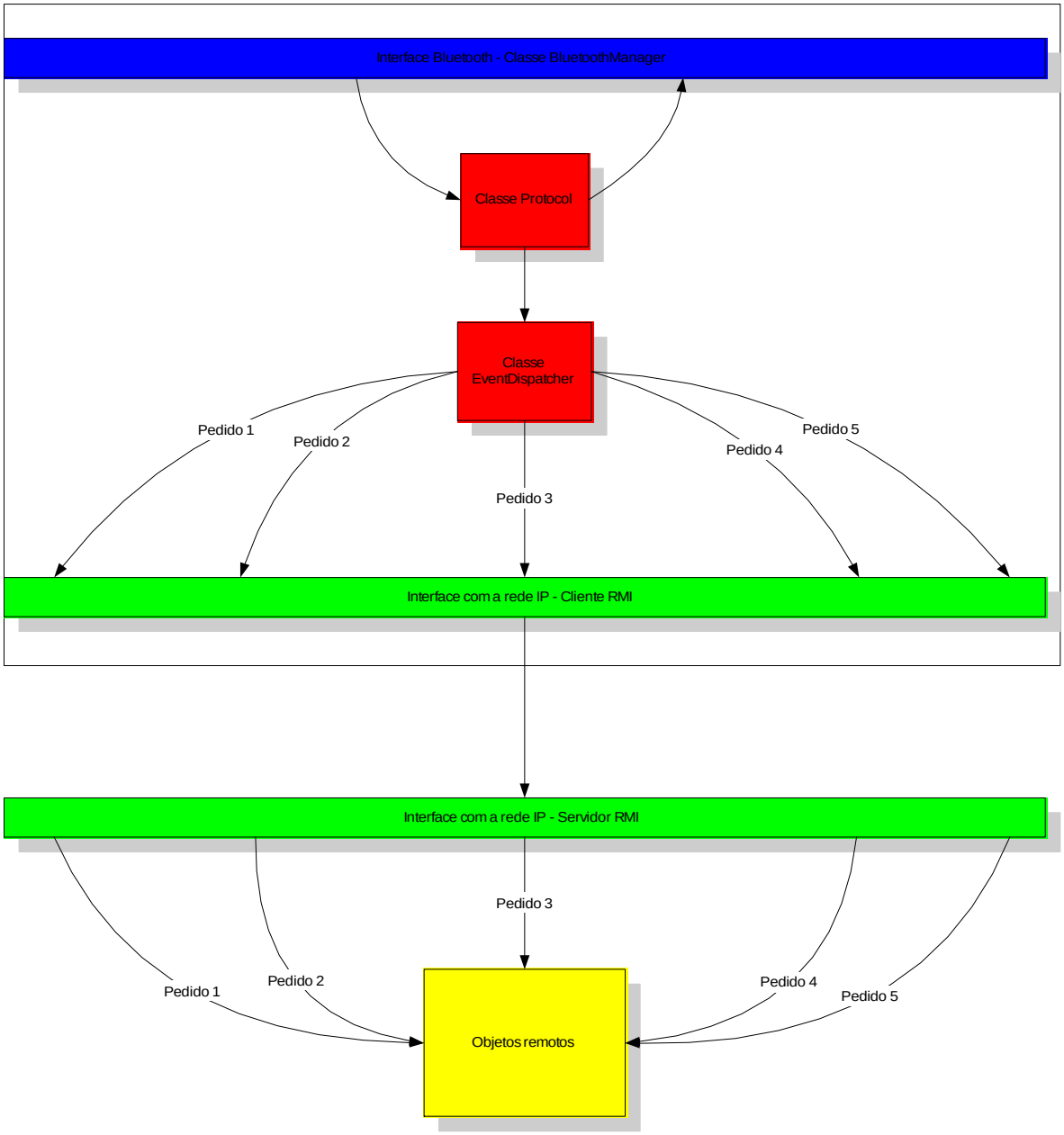

Figura 3.10 – Fluxograma dos pedidos do móvel.

Toda essa discussão se refere apenas ao AP recebendo pedidos do móvel. Mas como o AP também enviará mensagem originadas de móveis de outros AP's, ele também deverá ser capaz de receber mensagens do servidor, como mostrado na figura abaixo.

Ao receber um pedido do servidor, o AP fará uma operação análoga de quando recebe o pedido de um móvel. Ele criará um objeto para encapsular aquele pedido e o encaminhará para a classe *ObexMessage* que será responsável por enviar as mensagens via

Obex para os dispositivos Bluetooth desejados. A grande diferença entre essas duas situações expostas é que o pedido do servidor é feito via *Socket* e não por RMI. De fato, esse é ponto falho do sistema e deve ser corrigido em uma versão futura. Essa solução, como já foi dito anteriormente foi escolhida para contornar o problema de o AP ser ao mesmo tempo servidor e cliente RMI.

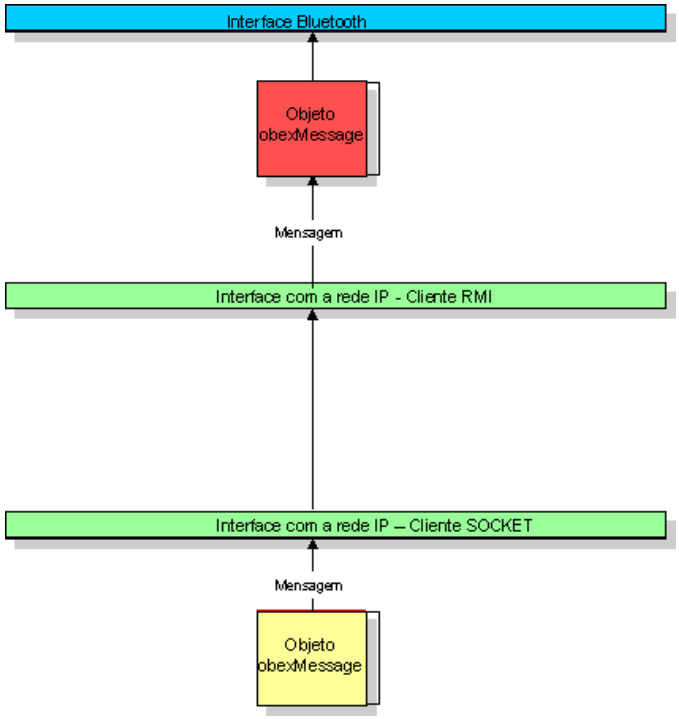

Figura 3.11 – Comunicação entre Servidor e AP via Socket.

Necessidade de um Protocolo de Identificação

Para toda a comunicação do AP para o celular, e também para que o AP consiga registrar o móvel em seu raio de cobertura, é necessário que o AP tenha conhecimento do endereço Bluetooth do celular. A solução adotada para isso foi a seguinte: Assim que um móvel é cadastrado no sistema pelo Desktop, ele recebe um número de protocolo que foi estabelecido pelo servidor. Em seguida, o móvel deve fazer o seu “login” na rede enviando o seu protocolo para o AP. Após isso acontecer o AP conhecerá não só o protocolo do móvel como também seu endereço Bluetooth.

Toda essa informação é enviada para o servidor, que irá relacionar o endereço Bluetooth ao móvel com aquele protocolo específico. Então, todas as vezes que o servidor precisar enviar uma mensagem para ser entregue ao móvel via OBEX, ele enviará também o endereço Bluetooth desse móvel que se encontra armazenado em seu Banco de Dados.

Vale lembrar que este protocolo foi definido apenas para a comunicação do AP para o celular. Caso o celular deseja se comunicar com o AP, ele consegue fazê-lo mesmo que ainda não tenha se “logado” no sistema enviando o seu protocolo de identificação. Ou seja, um móvel já cadastrado pelo Desktop no sistema, mas que ainda não enviou o seu protocolo de identificação ao AP, pode enviar mensagens a outros móveis cadastrados no sistema. Porém, este mesmo móvel só receberá mensagens que por ventura forem enviadas a ele via SMS ou email. Pois, como ainda não enviou o protocolo de identificação o AP não é capaz de registrá-lo em seu raio de ação e desta forma o móvel não estará em AP nenhum, não podendo receber as mensagens por Bluetooth. [1, 3, 4, 5, 6, 20]

3.2.4 – O Celular

Escolha da tecnologia

O desenvolvimento para dispositivos móveis depende das tecnologias que estão disponíveis nos próprios dispositivos para que as aplicações sejam implementadas.

Atualmente existem algumas tecnologias para desenvolvimento em dispositivos móveis, entre elas podemos citar Java 2 Micro Edition – J2ME, que é o conjunto de várias tecnologias móveis, entre elas é comum encontrar o Connected Limited Device Configuration - CLDC e o Mobile Information Device Profile – MIDP como tecnologias mais implementadas nos aparelhos atuais.

Projeto da Interface Gráfica de Usuário

A interface gráfica do aplicativo deve ser baseada na tecnologia usada. As tecnologias CLDC/MIDP fornecem duas formas de concepção de Interface Gráfica de Usuário, a saber: Interfaces de alto nível e interfaces de baixo nível.

A diferença entre uma interface de alto nível e a interface de baixo nível é a quantidade de controle que temos sobre a apresentação do aplicativo. Uma interface de alto nível pode ter um processo de desenvolvimento acelerado devido à quantidade de componentes prontos para o uso, entretanto não há muito controle sobre a forma que esses componentes serão apresentados. Uma interface de baixo nível permite grande controle sobre a interface gráfica do aplicativo.

No projeto foi escolhido o uso de interface de alto nível devido às necessidades serem completamente atendidas através dos componentes já prontos.

Projeto da Navegação do aplicativo

Quando se desenvolve um aplicativo para celular deve-se ter cuidados especiais no que se refere à navegação do aplicativo. No Gerenciador de Mensagens foi adotado um esquema de telas progressivas para que fique claro para o usuário final o seu progresso para realizar as ações desejadas. Para aumentar a eficiência do aplicativo disponibilizou-se dois modos de acesso: online e off-line.

Pode-se verificar quais são os grupos existentes e em seguida verificar quais são os usuários de um grupo escolhido, então pode-se enviar uma mensagem ao(s) usuário(s) que foi(ram) escolhido(s). Outra ação disponível apenas no modo online é a ativação de protocolo, que é necessária antes que o usuário comece a usar o aplicativo.

A diferença entre os dois modos é a quantidade de acessos ao Access Point (AP), pois no modo online acessa-se o AP para realizar as ações de verificar grupos, verificar usuários, enviar mensagem e enviar número de ativação de protocolo. No modo off-line usa-se os grupos e usuários que já foram recuperados no modo online e então acessa-se o AP somente para enviar mensagem ao(s) usuário(s) escolhido(s).

As telas geralmente têm opção para voltar ou para cancelar quando são telas de conexão.

Projeto do Armazenamento Persistente

O número de ativação de protocolo fica armazenado no Record Management System (RMS) do celular. Podendo ser alterado ou modificado conforme for necessário, mas mantendo-se sempre acessível mesmo que o celular seja desligado ou que a aplicação seja terminada.

Uma única tabela foi projetada para armazenar apenas esse número de ativação, evitando o armazenamento de qualquer outro dado desnecessário para economizar os recursos do celular.

Projeto da Comunicação entre o dispositivo móvel e o Access Point

Para que exista comunicação coerente com o AP é necessário que fiquem bem definidos os dados que serão recebidos e enviados tanto do celular para o AP quanto do AP para o celular.

Foi definido um padrão de comunicação onde o primeiro inteiro enviado significaria a ação que é solicitada quando o dado é enviado a partir do celular ou a ação que está sendo respondida quando o dado é enviado a partir do AP.

As ações disponíveis são: solicitação de grupos, solicitação de usuários de um grupo, envio de mensagem para usuários e ativação de número de protocolo.

Cada ação foi otimizada para que a quantidade de dados trocada seja mínima.

Escolha de tecnologia usada para transmitir dados

A tecnologia escolhida foi o Bluetooth para a troca de informações entre o AP e o celular.

Para usar essa tecnologia usou-se a API desenvolvida pelo projeto Inova Mobile para o uso de Bluetooth. Essa API é conveniente pois é bastante similar à API usada no AP (Atinav) e dessa forma a comunicação entre os membros da equipe do celular e os membros da equipe do AP fica bastante facilitada e padronizada.

Os padrões seguidos no mercado para busca por dispositivos, conexão, troca de dados e desconexão, como pode ser visto no código do aplicativo, foram os mais comuns e também visualmente na interface gráfica de usuário. [1, 3, 6, 20]

3.3 – TESTES E EXEMPLOS DO SISTEMA

Funcionamento do Sistema

A figura abaixo mostra um exemplo de rede em um ambiente corporativo.

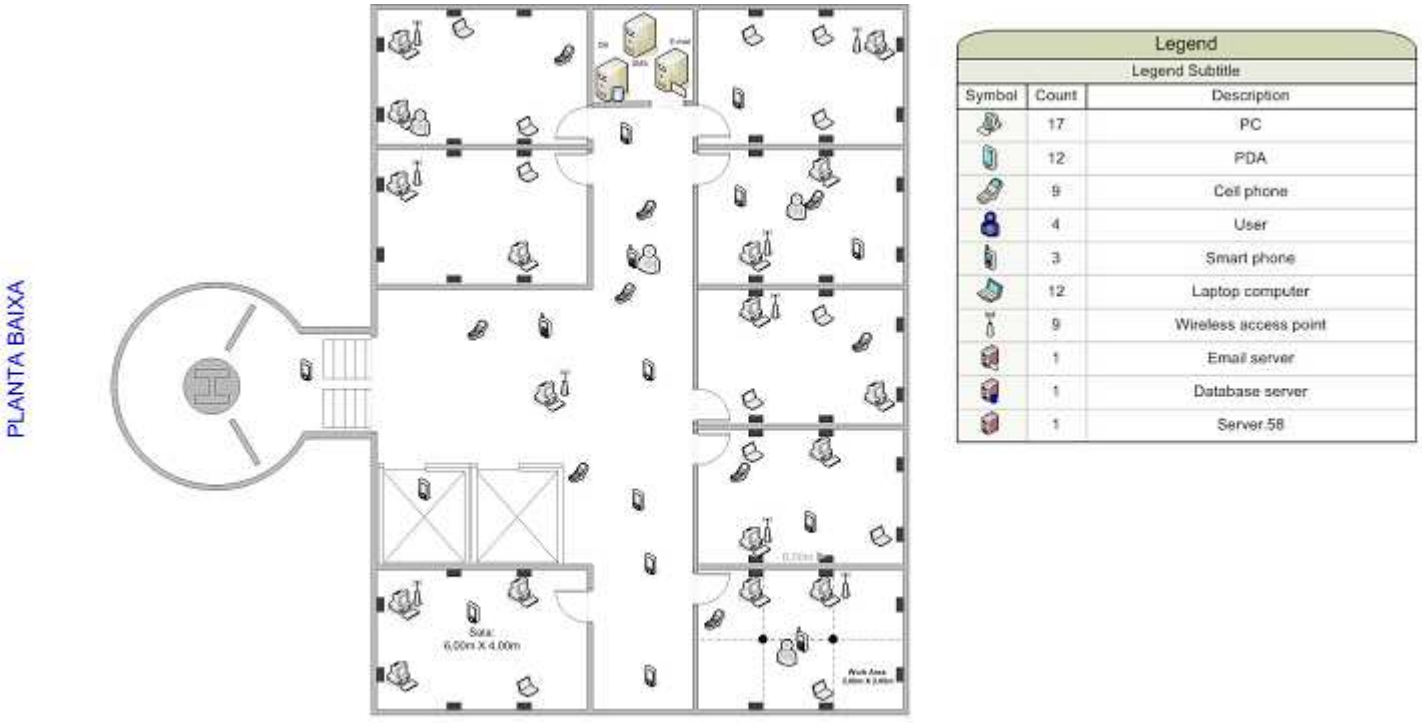

Figura 3.12 - Rede do sistema em ambiente corporativo.

Os dispositivos utilizados pelo sistema estão indicados.

O Desktop é o responsável por cadastrar os usuários e seus grupos no servidor de Banco de Dados. Para a comunicação entre os dois aplicativos foi desenvolvido a interface em Java LoginServer, responsável por adicionar, apagar ou modificar grupos, usuários e Pontos de Acesso no banco de dados; e a interface CommunicationServer, que é capaz de recuperar usuários, grupos e Pontos de Acesso, bem como enviar mensagens. A figura abaixo ilustra essa implementação:

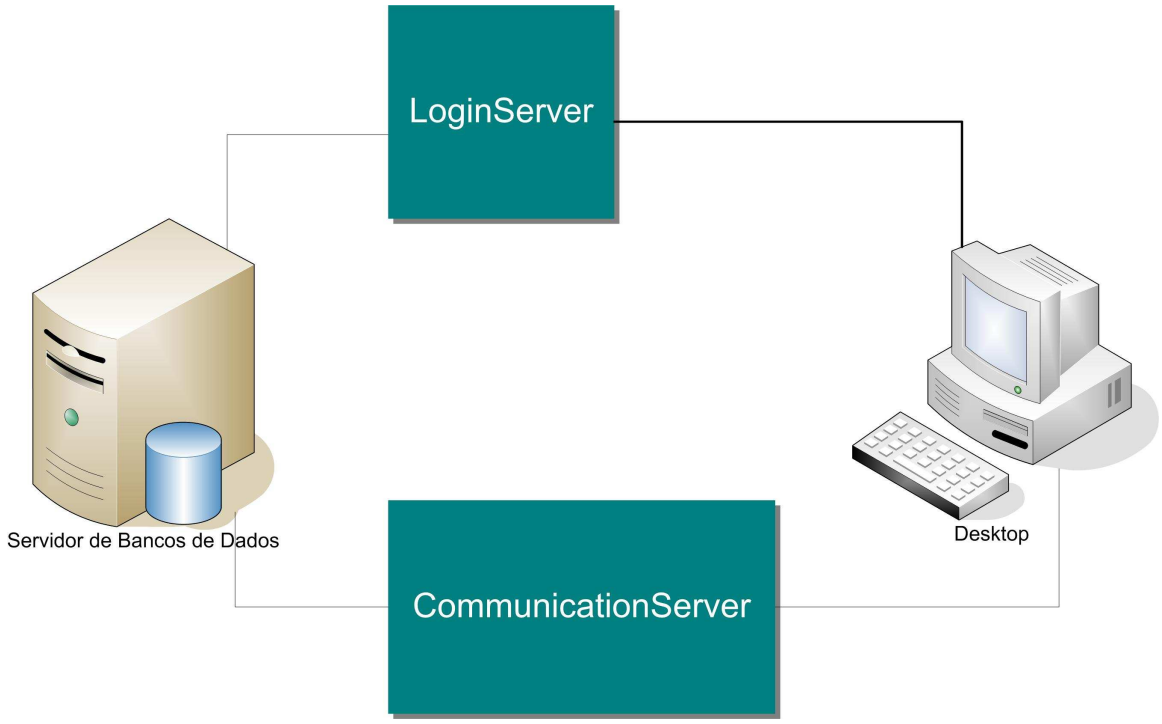

Figura 3.13 - Interfaces de comunicação entre Servidor e Desktop.

O Servidor do banco de dados, que implementa todos os métodos dessas duas interfaces, responde às solicitações do Desktop atualizando o banco. Abaixo segue figura com as tabelas contidas no Banco de Dados:

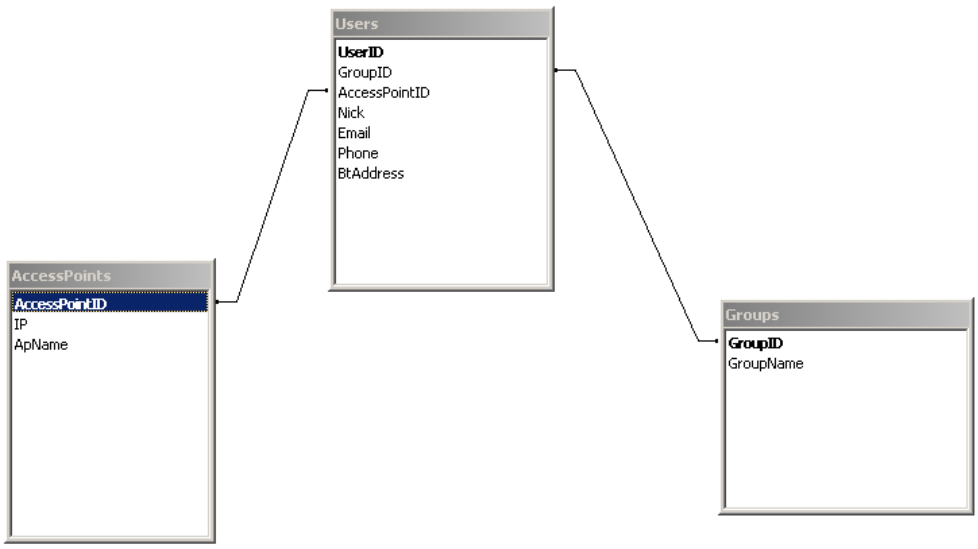

Figura 3.14 - Tabelas do Banco de Dados.

Ao adicionar um usuário, o servidor gera um número de protocolo. Este foi definido, por questões de simplificação nesta primeira etapa do projeto, como sendo o número de ID do usuário. O Desktop recebe esse protocolo e deve informá-lo ao usuário o qual poderá utilizar o sistema. Esse processo será utilizado para armazenar o endereço Bluetooth do usuário.

Devido a uma limitação do protocolo Bluetooth, um dispositivo Bluetooth não é encontrável enquanto está procurando por outros. Contornando isso, desenhamos o Ponto de Acesso para que tenha intervalos entre uma procura por dispositivos e outra, passando parte do tempo visível e outra parte invisível.

O AP é capaz de se comunicar com os dispositivos móveis por meio de OBEX e usando um protocolo definido pelos respectivos desenvolvedores. A figura abaixo ilustra essa idéia:

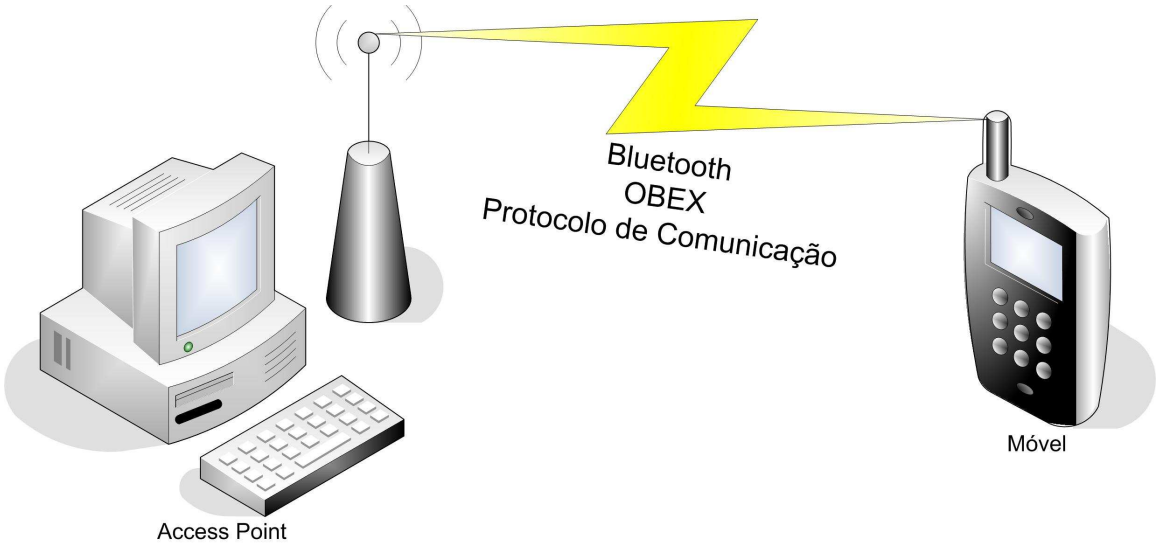

Figura 3.15 - Comunicação entre Celular e AP.

Também se comunica com o Servidor de Banco de Dados, por RMI, usando a interface CommunicationServer e ControllerServer, responsável por registrar usuários como pertencentes a esse Ap. A figura abaixo ilustra as interfaces:

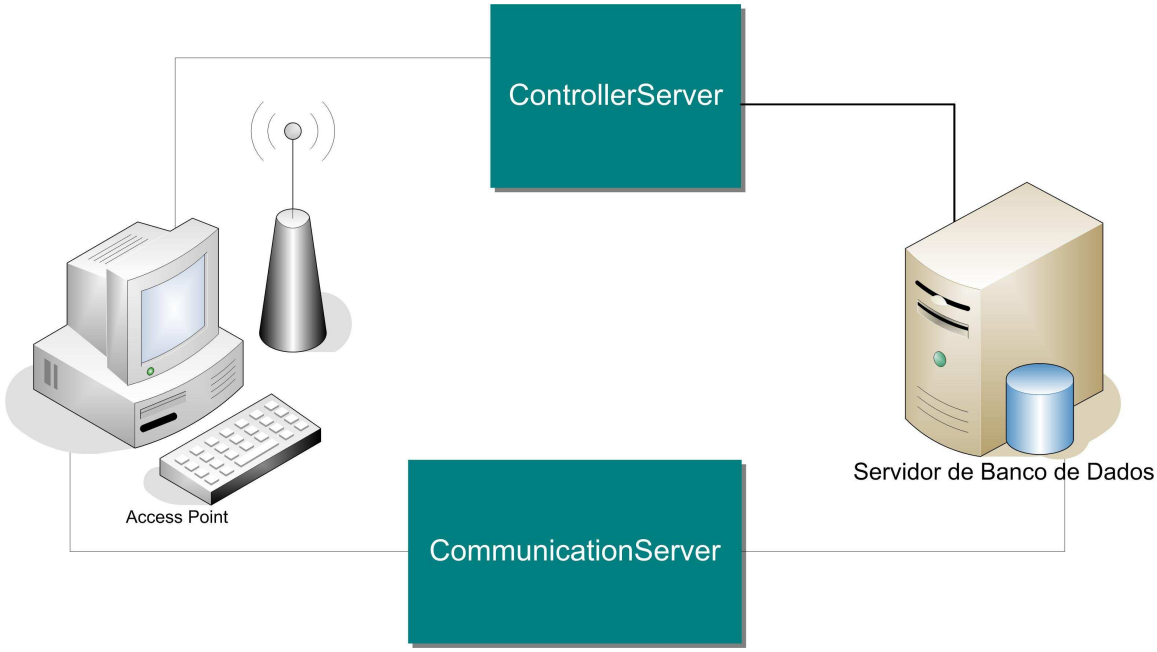

Figura 3.16 - Interfaces de comunicação entre AP e Servidor.

Quando o nosso usuário tentar usar o sistema pela primeira vez, o Ap irá perguntar o número de protocolo. Ao receber a confirmação do dispositivo móvel, aquele enviará o protocolo e o endereço Bluetooth do usuário para o Servidor. Agora o sistema conhece todos os dados necessários. Enquanto este processo não estiver completo, o

usuário poderá apenas receber mensagens geradas pelo sistema via SMS ou email. Isso é o que acontece com aqueles que, por ventura, não possuam esse tipo de endereço em seus dispositivos móveis.

Tendo realizado esse processo, o usuário estará totalmente inserido no sistema. E seu registro será realizado no AP (registro de visitantes) e no Servidor (registro de usuários). Agora, toda vez que o usuário entrar nas dependências da empresa, estando o AP no período de procura, irá localizá-lo, comparando os dados do mesmo com o seu registro de visitantes. Se não for encontrado, o AP irá solicitar ao Servidor o registro do usuário como pertencente àquele AP. O Servidor, por sua vez se encarregará de verificar se o usuário está registrado em outro AP. Neste caso, um aviso ao AP solicitante é emitido impossibilitando-o de atualizar seu registro de visitantes com o novo usuário. Esse processo impede a ocorrência de Hand-over (troca de AP) desnecessários.

Agora, qualquer um logado no sistema será capaz de localizar e ser localizado.

A figura abaixo ilustra os casos de possível ocorrência de hand-over.

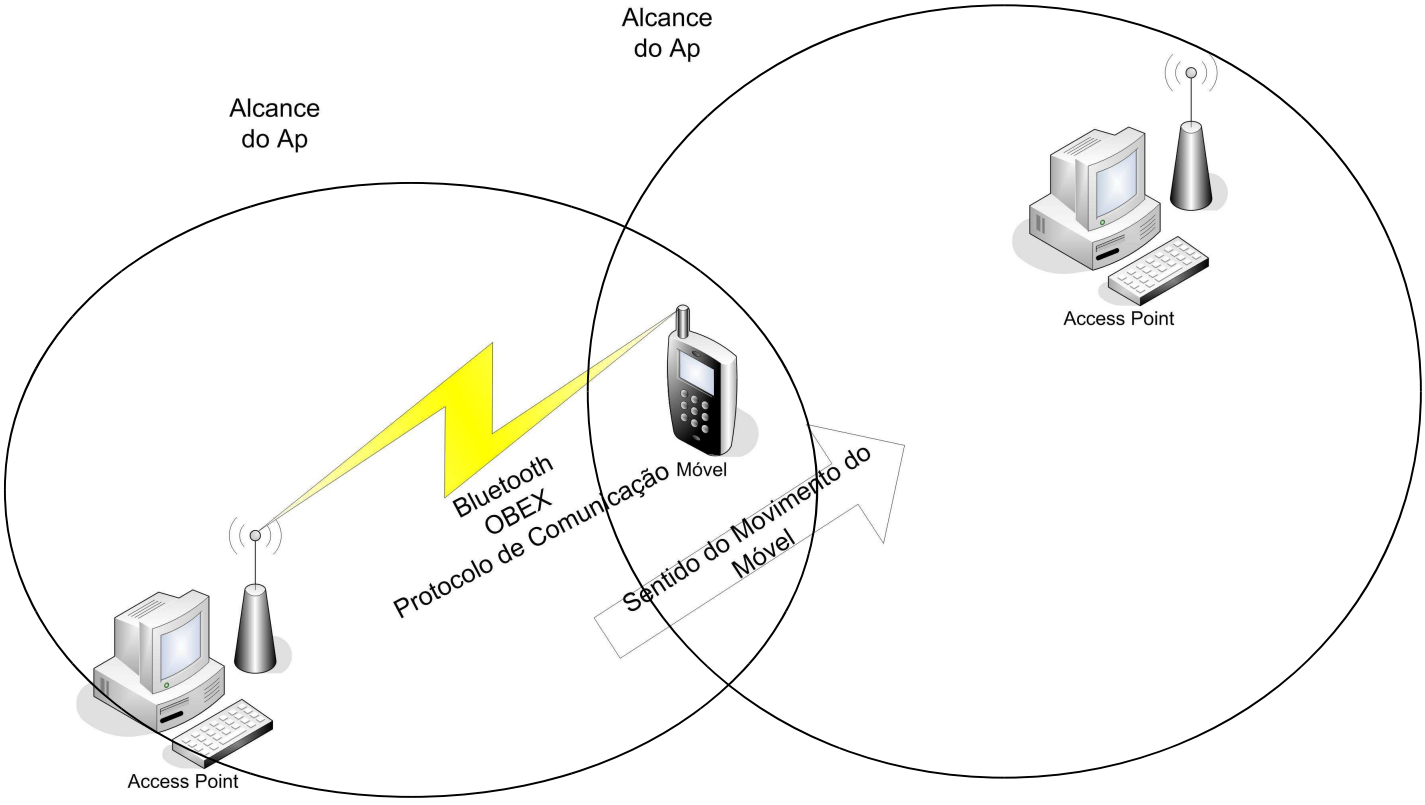

Figura 3.17 - Ocorrência de Hand-over no sistema.

O processo de hand-over entre APs é controlado pelo Servidor de Banco de Dados. Quando o Ponto de Acesso que controla o usuário não o encontrar mais, durante uma procura, ele irá comunicar a ocorrência para o Servidor de Bando de Dados. Agora,

com o usuário não controlado por nenhum AP, o primeiro que o encontrar e solicitar o registro será eleito o novo AP do usuário.

Esse método foi escolhido por ser o mais adequado para o nosso protótipo, o qual não demanda sofisticação nesta primeira etapa. Para a próxima etapa do projeto, iremos implementar hand-over baseado em Histerese e limiar de potência do sinal.

Envio de Mensagens

Quando um usuário cadastrado deseja enviar uma mensagem, ele deverá seguir o seguinte procedimento:

Primeiro o usuário faz uma procura por APs que estão próximos. Ele receberá uma lista com todos os dispositivos Bluetooths na área. Deverá então escolher um AP na lista apresentada. Não é necessário que o AP escolhido seja o mesmo que controla esse usuário. Qualquer Ponto de Acesso que estiver encontrável poderá manejar o envio de mensagem. O AP que controla o dispositivo será usado apenas quando o usuário for receber uma mensagem.

Escolhido o AP, o usuário se conecta a ele. Usando o protocolo definido pelos desenvolvedores do aplicativo do dispositivo móvel e do AP, os dois se comunicam. O usuário pede uma lista dos grupos cadastrados no banco de dados. O Ponto de Acesso recebe esse pedido e o repassa ao Servidor de Banco de Dados utilizando a interface CommunicationServer e RMI. O servidor acessa o banco e verifica a tabela Groups. O resultado da pesquisa é então retornado ao Ap. Este deve então formatar o resultado recebido de acordo com o protocolo e o envia ao dispositivo móvel.

A lista com os grupos aparece na tela do móvel. O usuário escolhe o grupo no qual se encontra o destinatário. Ao fazer isso, envia uma solicitação ao AP para mostrar os usuários do grupo escolhido. O processo então se repete com o AP fazendo a solicitação ao Servidor, que acessa a tabela Users e retorna todos os usuários que estão no grupo escolhido. O AP formata o resultado recebido e o envia ao dispositivo. Em sua tela, aparece a lista de usuários daquele grupo.

Ao encontrar o destinatário na lista, o usuário pode agora escrever a mensagem. Quando pressiona Enviar, o dispositivo faz a solicitação de envio de mensagem para o AP, o qual não precisa se preocupar com a forma com que a mensagem será entregue, apenas cria um objeto genérico MessageRequest. Por meio da interface CommunicationServer, utiliza esse objeto para solicitar o envio de mensagens ao servidor.

O servidor, ao receber o pedido, verifica a localização do destinatário. Isso é feito consultando o Banco de Dados na tabela Users e AccessPoints. A partir desse ponto podem ocorrer três processos diferentes:

1. O destinatário foi encontrado:

Com a confirmação no Banco de Dados de que o destinatário se encontra nas dependências da empresa, o Servidor se encarrega de enviar o objeto MessageRequest ao Ap ao qual o destinatário pertence. Para isso é usado Socket. Não foi feita uma interface para usar RMI neste tipo de comunicação de forma a evitar que Ap e servidor trocassem de papel quanto a qual é servidor RMI e qual é cliente. Assim, o servidor RMI é sempre o aplicativo Servidor de Banco de Dados e o servidor de Socket é sempre o Ap.

O Ponto de Acesso verifica se o destinatário realmente encontra-se em seu domínio. Caso isso não seja confirmado, o objeto MessageRequest é retornado ao Servidor para que este repita o processo.

Com a localização positiva, o AP solicita a conexão ao dispositivo. Um aviso de que existe um AP tentando se conectar com o destinatário aparece em sua tela. Se não for aceita, a mensagem será perdida. Com a aceitação, começa o envio da mensagem para o dispositivo móvel por meio de OBEX. Ao concluir, a conexão é finalizada e o destinatário poderá agora ler a mensagem. Como explicado acima, não há confirmação de recebimento.

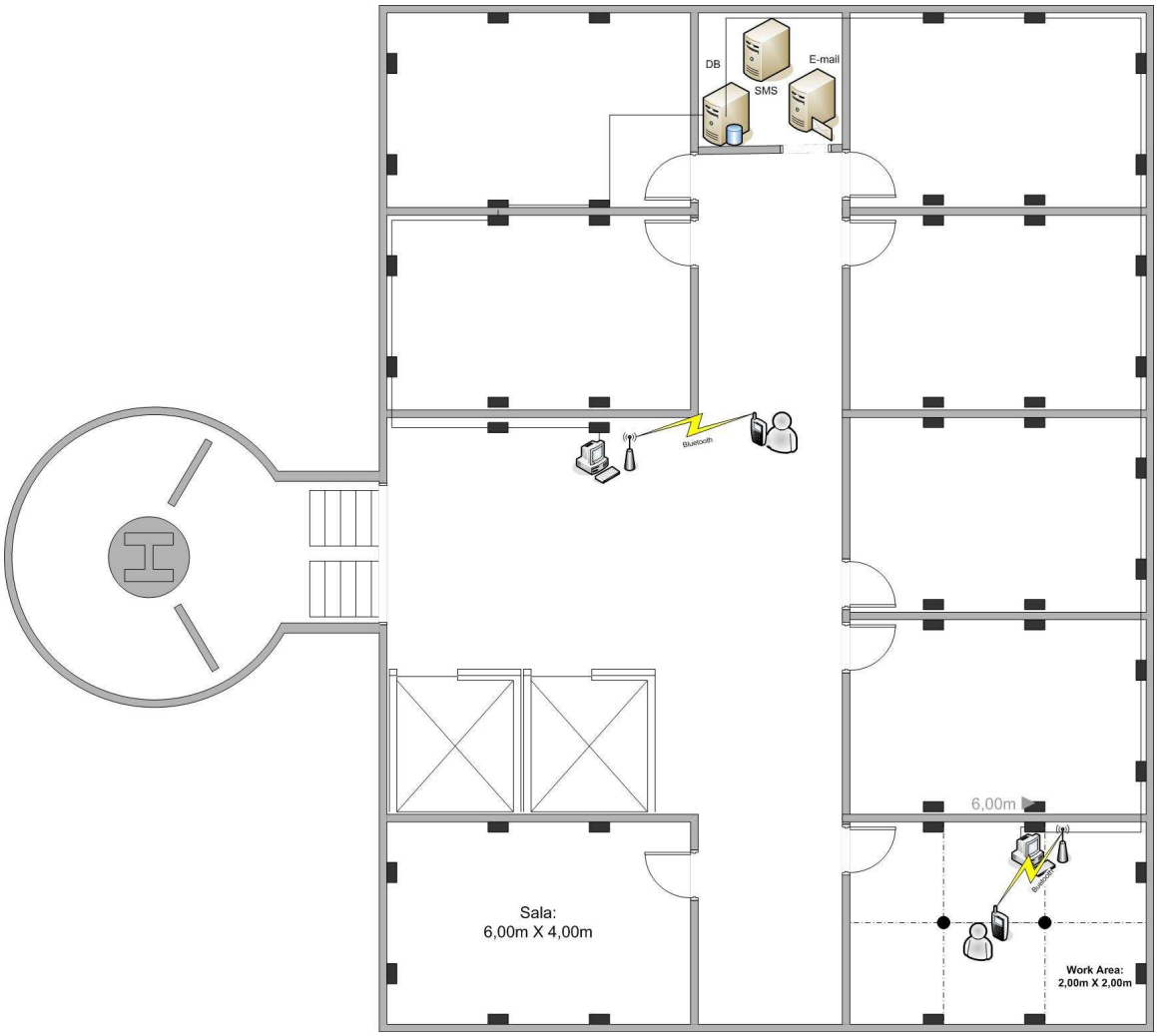

Figura 3.18 - Ilustração do caso 1 – Destinatário encontrado.

2. O destinatário não foi encontrado:

O Servidor, por ser incapaz de localizar o destinatário, se conecta ao Cliente de SMS. Por este cliente estar em uma máquina protegida por um Firewall em um Sistema Unix, não foi possível o uso de RMI. O Servidor de Banco de Dados não foi capaz de carregar um objeto do Cliente de SMS. Portanto, toda a comunicação entre eles é feita com Socket.

O Servidor envia a mensagem para o Cliente de SMS. Este se encarrega de conectar-se ao Servidor de SMS da Claro para que a mensagem seja enviada. Este Servidor, por políticas da própria empresa, envia SMS apenas para celulares da Claro. Se o celular do destinatário for da Claro, a mensagem será enviada.

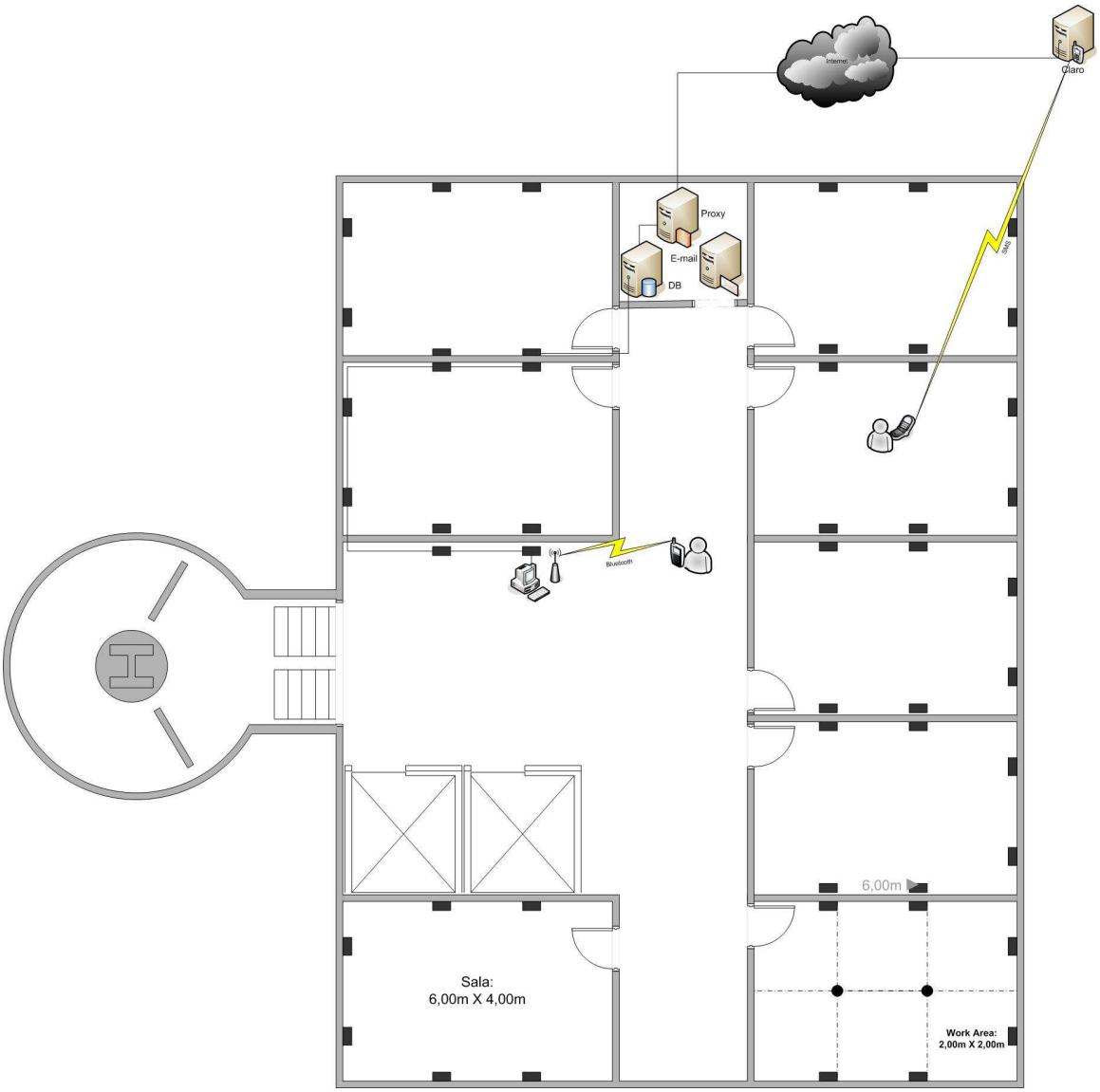

Figura 3.19 - Ilustração do caso 2 – Destinatário não encontrado.

3. O destinatário não foi encontrado e seu celular não é da Claro:

Se o Cliente de SMS no item anterior tiver solicitado o envio de SMS para um celular que não seja da Claro, o servidor de SMS irá retornar uma mensagem de erro. O Cliente, então, avisa ao Servidor de Banco de Dados sobre o erro, que toma a decisão de enviar a mensagem por email.

Usando a interface SMTPServerObject, o Servidor de Banco de Dados solicita o envio de email para o Cliente de Email.

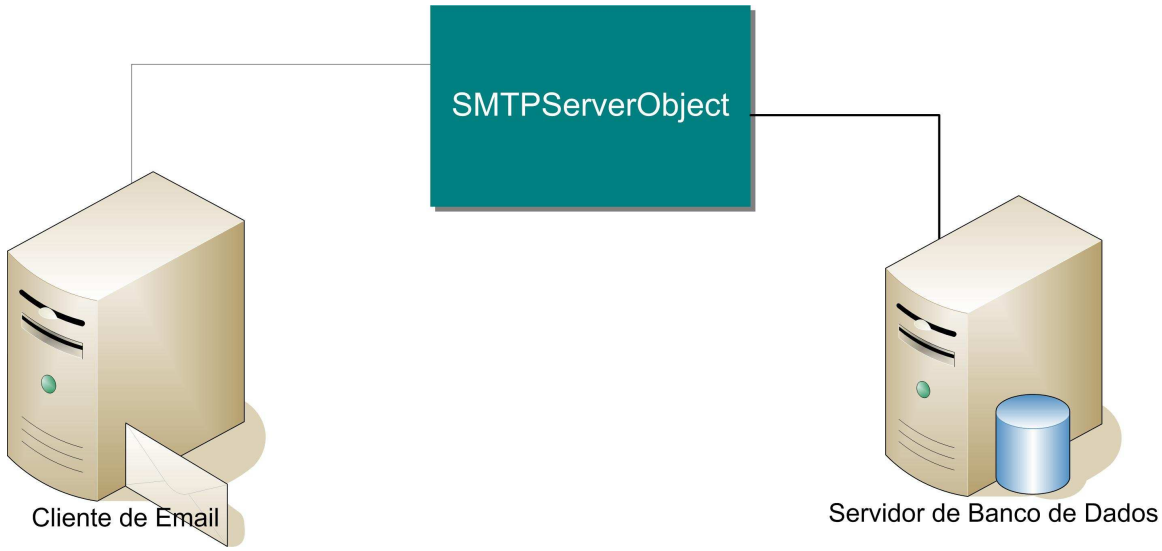

Figura 3.20 - Comunicação entre Servidor de Banco de Dados e Servidor de Email.

Este se conecta ao servidor de SMTP do Yahoo! e o utiliza para enviar o email. O campo “From” é configurado com o endereço de email do usuário que enviou a mensagem, de forma que o destinatário possa respondê-la normalmente como um email comum. A resposta será recebida na caixa de entrada do email do usuário que enviou a mensagem original.

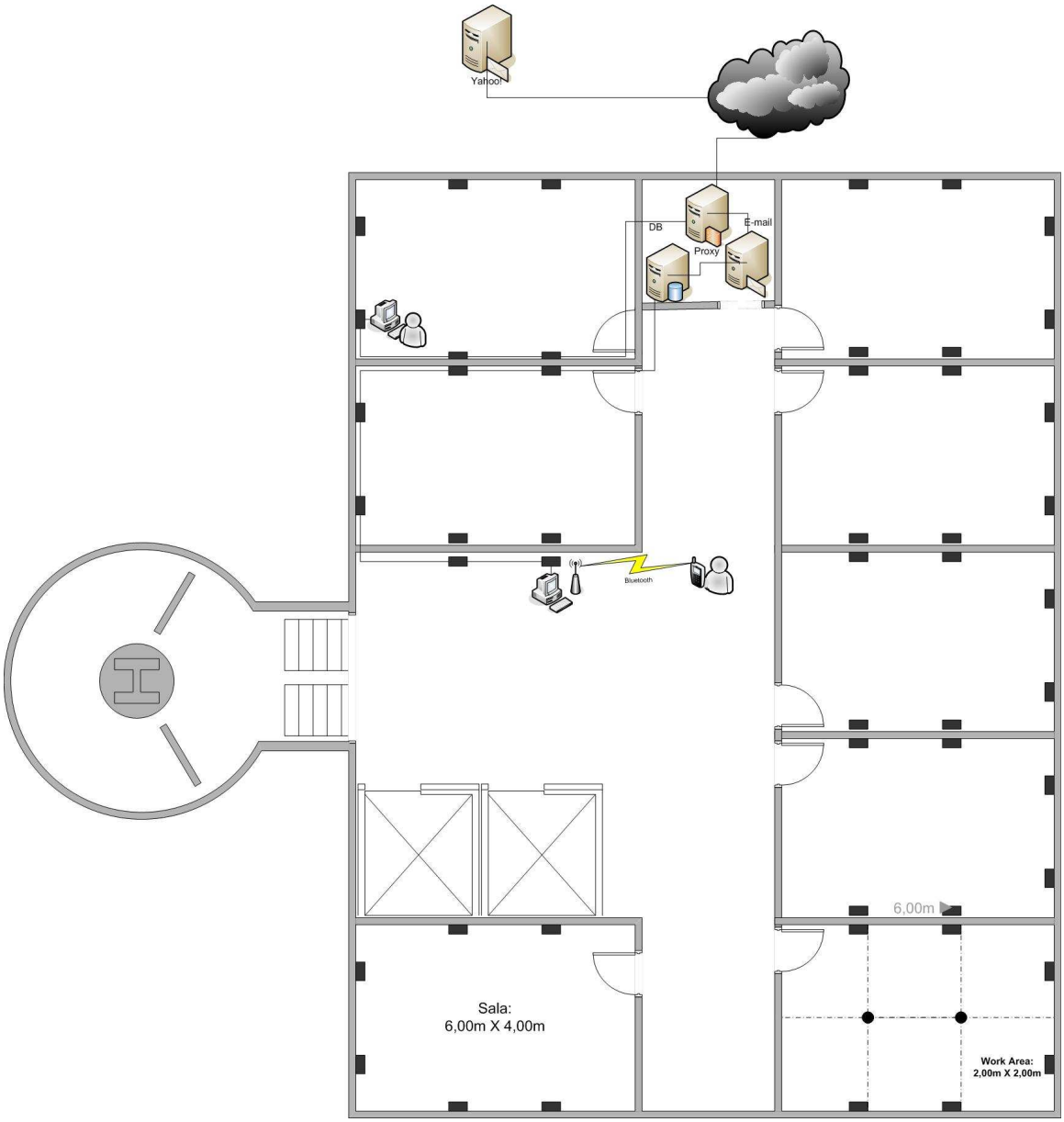

Figura 3.21 - Ilustração do caso 3 – Destinatário não encontrado com celular que não é da Claro.

4 – O SISTEMA GERENCIADOR DE PROJETOS

4.1 – INTRODUÇÃO

O objetivo deste aplicativo desenvolvido em Java é simplificar as operações de gerência, tais como: cadastrar eventos, inserir e excluir grupos, inserir e excluir usuários e enviar mensagens.

4.2 – AS PARTES DO SISTEMA

O Sistema Gerenciador de Projetos também está dividido em quatro grandes partes: AP, Servidor, Celular e o Aplicativo Web. As três primeiras partes são as mesmas do Sistema Gerenciador de Mensagens que foram desenvolvidas no Projeto Final I. No Projeto Final II foi desenvolvido o Aplicativo Web.

4.2.1 – O Aplicativo Web

O Aplicativo Web, semelhante ao Desktop do Sistema Gerenciador de Mensagens, também faz cadastros de grupos e usuários, além de mandar mensagens, mas difere dele em algumas funcionalidades como o cadastro de perfis, prioridade e status do projeto.

Outra diferença do Aplicativo Web para o Desktop é que este deve ser instalado na máquina em que o Desktop rodar como aplicativo ao passo que aquele pode ser acessado em qualquer computador que tenha acesso a Internet. E como o foco principal deste Projeto foi a comunicação absoluta, deste modo foi escolhido que o Aplicativo Web fosse feito em J2EE e não em J2SE.

A parte gráfica deste aplicativo, o Aplicativo Web, foi desenvolvida usando JSP (Java Server Pages) como forma de apresentação ao usuário e utiliza como base o framework Struts e o Container Web utilizado foi o Tomcat. Além disso foram utilizados padrões de projetos nas classes de negócio e de modelo do aplicativo para uma melhor estruturação do código assim como uma mais fácil manutenção.

A figura abaixo é uma das telas do sistema, e é a tela de cadastros de usuários:

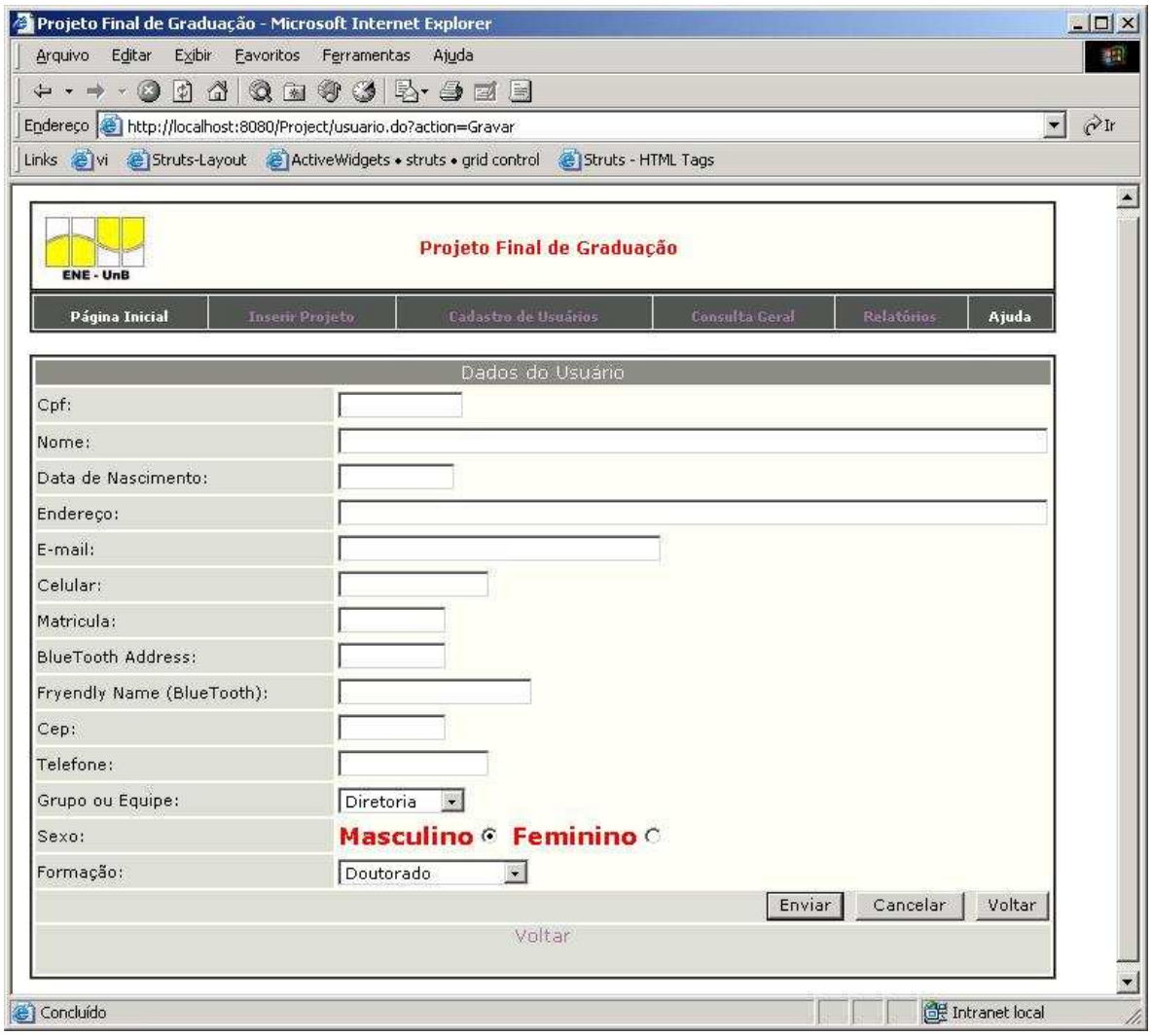

Figura 4.1 – Cadastro de usuários do Aplicativo Web.

Assim como a tela de cadastro de usuário existe também as telas de cadastro de projeto e de cadastro básico, aos quais são cadastrados os Grupos, a Formação, o Status do projeto, o Tipo de projeto e a Prioridade do projeto.

No aplicativo existe também a possibilidade de listagem dos conteúdos cadastrados, pois com a listagem fica fácil a visualização para fazer uma atualização ou então remover algum dado do Banco de Dados. A figura 4.2 abaixo mostra a listagem de dois usuários cadastrados no Banco de Dados:

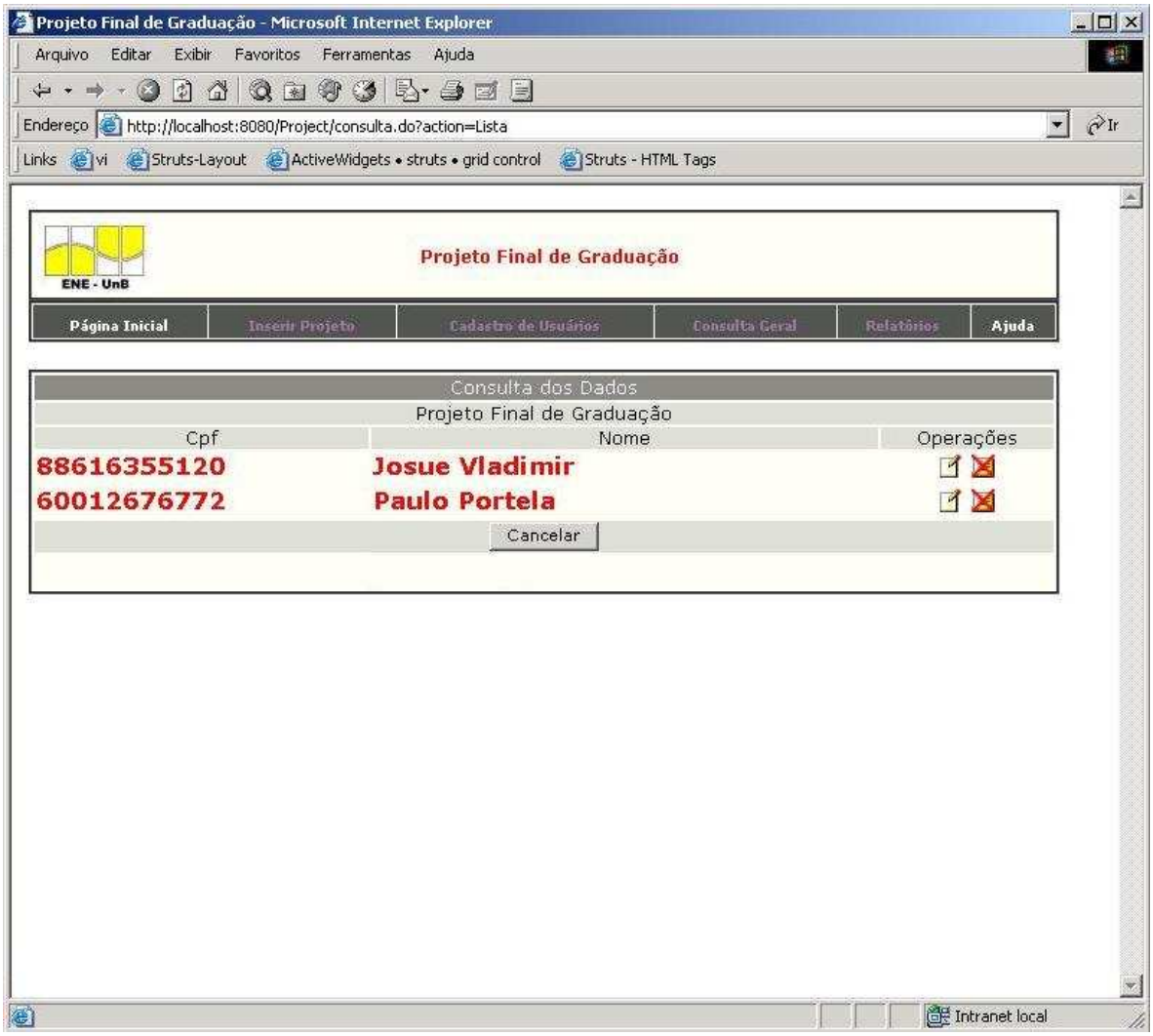

Figura 4.2 – Cadastro de usuários do Aplicativo Web.

O Aplicativo Web é uma proposta para facilitar a gerência de projetos dos usuários do sistema. É uma solução que pode, por exemplo, ajudar uma secretária a enviar mensagens a respeito de reuniões e outras informações de interesse para um grupo de pessoas. Por isso, devido às suas características, pode ser muito útil em um ambiente corporativo e também muito prático, pois pode ser acessado em qualquer local onde se tem a Internet. [6, 8, 10, 11, 12, 14, 18]

4.3 – ARQUITETURA

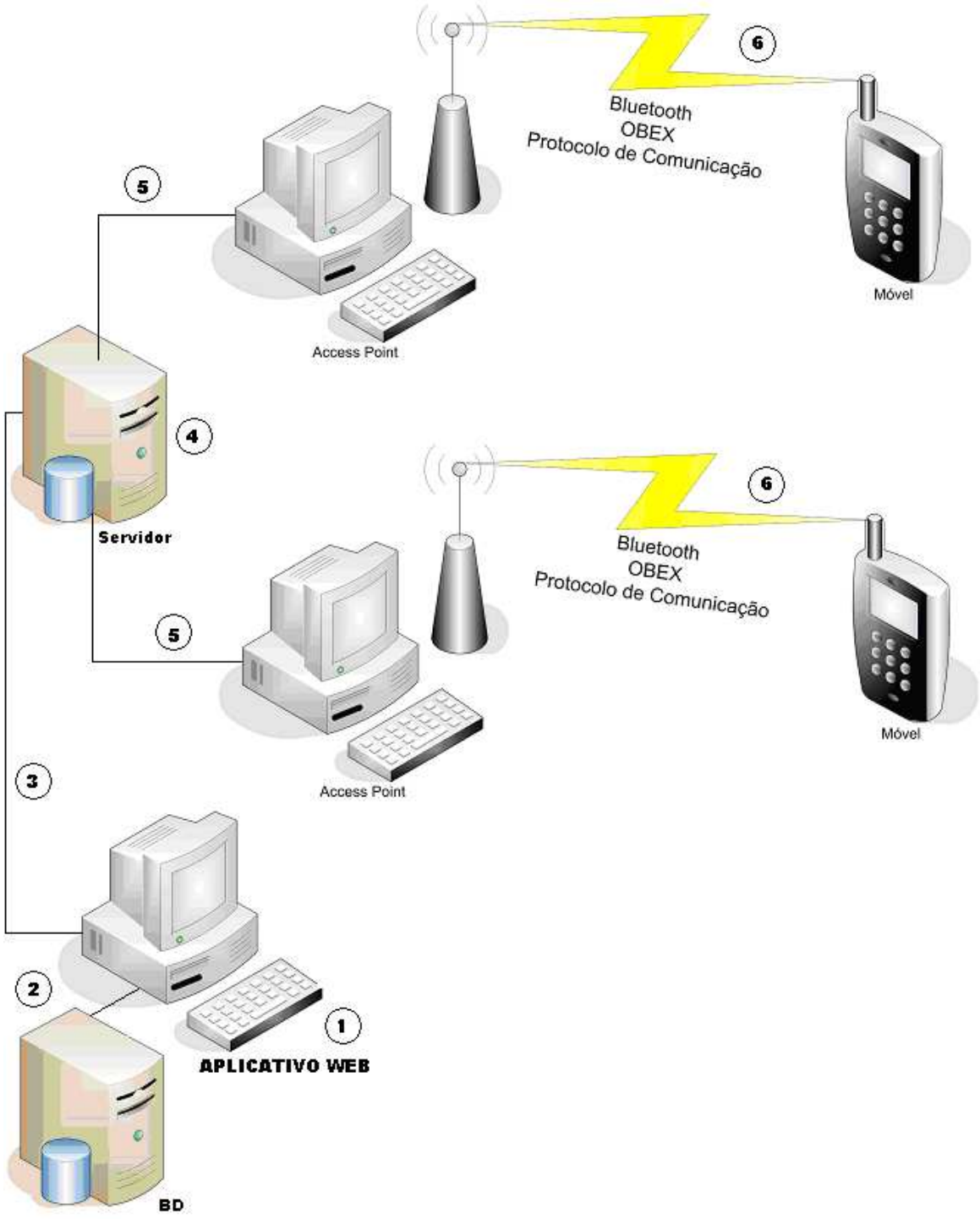

Figura 4.3 – Arquitetura do Aplicativo Web.

Abaixo segue um exemplo de fluxo da aplicação:

O Aplicativo Web faz um cadastro de um projeto;

1. Os dados inseridos na página são armazenados no Banco de Dados;
2. Ao realizar o cadastro de um projeto o usuário pode enviar aos participantes do projeto 3 tipos de mensagens: o SMS (para celulares da Claro), o Email ou um SMS Bluetooth. Supondo que o Gerente de Projetos selecionou a opção *Envio de Mensagem via Bluetooth* . Os dados cadastros são então enviados ao servidor.
3. O Servidor verifica se os usuários do projeto cadastrado estão em algum dos APs;
4. Ao achar um usuário em um AP o Servidor envia ao AP os dados do Projeto Cadastrado.
5. Ao receber os dados do projeto o AP estabelece uma conexão Bluetooth com o celular e envia-lhe os dados.
6. Caso ocorra alguma exceção na transmissão via Bluetooth os dados são enviados para o email do usuário.

4.4 - BANCO DE DADOS

O diagrama de blocos abaixo mostra como foi montada a base de dados do Aplicativo Web. As setas mostram os relacionamentos entre as tabelas.

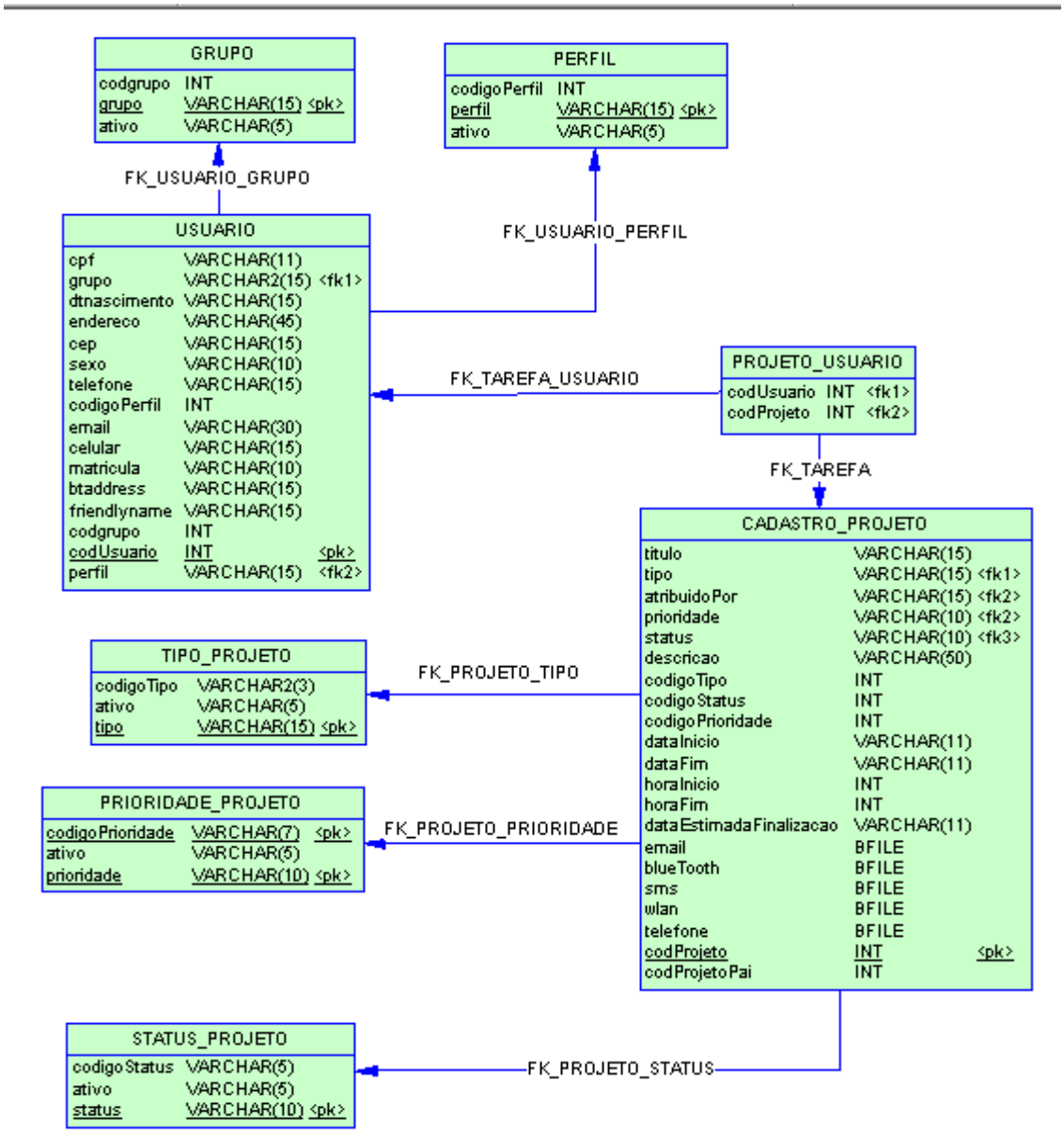

Figura 4.4 – Diagrama de blocos do Banco de Dados.

5 - CONCLUSÃO

A proposta inicial do projeto foi o desenvolvimento de diversas funcionalidades para uma sistema gerenciador de projetos aos quais pudesse dar auxílio a processos focado em uma comunicação absoluta, utilizando-se, para tanto, um conjunto de tecnologias de ponta.

Com a necessidade de uma comunicação rápida e segura, as pessoas vêm buscando cada vez mais uma maior mobilidade e consequentemente uma maior conectividade, e estas foram as principais motivações para a concepção do Sistema Gerenciador de Projetos. Com o avanço da tecnologia, os novos lançamentos de dispositivos móveis estão vindo com novas funcionalidades tecnológicas, como o Bluetooth e o WLAN, evidenciando nesse mercado um rápido crescimento, possibilitando uma grande demanda de aplicativos para estes dispositivos de modo a facilitar a produtividade de profissionais.

Os principais pontos que geraram dificuldades para o desenvolvimento do projeto foram a falta de domínio, inicialmente, das tecnologias utilizadas – o que acarretou uma necessidade de estudo das novas tecnologias adotadas utilizando padrões de projeto e o framework Struts como base do aplicativo.

Para o desenvolvimento do projeto foi necessário o estudo das tecnologias utilizadas, tais como, a linguagem Java – J2ME, J2SE e J2EE (RMI, JDBC, Servlet, JSP), –, Bluetooth, XML, - Design Patterns, Struts e *MySQL* – algumas das quais se encontram conceituadas neste trabalho.

Para o projeto, foram idealizados serviços que possibilitariam:

a) a inclusão de marcos (metas) para os projetos em uma área gráfica denominada Gráfico de Gantt, aos quais permite exibir graficamente as durações das tarefas e as datas de início e término em uma escala de tempo;
b) a possibilidade de gerar relatórios em diversos modos (HTML, PDF) facilitando a visualização dos projetos cadastrados;

c) a possibilidade de páginas pessoais listando as tarefas do usuário nos respectivos projetos em que este estivesse cadastrado.

Uma versão mais simplificada desses serviços foi desenvolvida devido à pouca disponibilidade de tempo para a conclusão do projeto, considerando que boa parte do tempo do projeto foi destinado a estudos de tecnologias, ficando assim como sugestões para aperfeiçoamento em projetos futuros. Outras sugestões seriam desenvolvimentos de novos serviços aplicados a gerência de projetos, ramo este muito importante no cotidiano das empresas atuais.

6 - REFERÊNCIAS BIBLIOGRÁFICAS

[1] - HOPKINS, Bruce. ANTONY, Ranjith. **Bluetooth for Java**. New York, Editora Apress, 2003.

[2] - IEEE Computer Society. **802.15.1 – Part 15.1: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Wireless Personal Area Networks (WPANs)**. New York, The Institute of Electrical and Electronics Engineers, Inc, 2002.

[3] - JSR-82 Expert Group. **Java APIs for Bluetooth Wireless Technology (JSR-82)**. Versão1.0a, Austin, Motorola – Wireless Software, Applications & Services, 2002.

[4] - NOKIA, Forum. **Bluetooth Technology Overview**. Versão 1.0, Forum.Nokia.Com, 2003.

[5] - DEITEL, H.M. DEITEL, P. J. **Java Como Programar**. Quarta Edição. Porto Alegre, Editora Bookman, 2003.

[6] – Tecnologia Java, http://java.sun.com/

[7] - GROSSO, William. **Java RMI**. Primeira Edição, Editora O'Reilly, 2001.

[8] - KURNIAWAN, BUDI. **Java para a Web com Servlets, JSP e EJB.** Rio de Janeiro, Editora Ciência Moderna Ltda., 2002.

[9] Tecnologia Java, http://www.portaljava.com.br/

[10] - HUSTED, TED; DUMOULIN, CEDRIC; FRANCISCUS, GEORGE; WINTERFELDT, DAVID. **Struts em ação.** Rio de Janeiro, Editora Ciência Moderna Ltda., 2004.

[11] – Framework Struts, http://struts.apache.org

[12] – Framework Struts, http://struts.application-servers.com/

[13] – Endo, Eduardo. **Struts**, In: Mundo Java – Número 02 – Ano I

[14] - Tecnologia MySQL, http://www.mysql.org

[15] – Reis, Glauco. **J2EE Patterns**, In: Mundo Java – Número 06 – Ano I

[16] – Jandl Jr., Peter. **Padrões de Projeto em Java**, In: Mundo Java – Número 06 – Ano I

[17] - ALLEN, PAUL R. **Sun Certified Enterprise Architect for J2EE: guia oficial de certificação.** Rio de Janeiro, Campos, 2003.

[18] - ALUR, DEEPAK. **Core J2EE™ patterns: as melhores práticas e estratégias de design.** Rio de Janeiro, Campus, 2002.

[19] – Design Patterns, http://www.linhadecodigo.com.br/

[20] – Tecnologia Bluetooth, http://www.atinav.com/