UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

Desenvolvimento de aplicativos para o Wireless Village

por

Marcos Garrison Dytz

Projeto Final de Graduação

Área de concentração: Engenharia Elétrica
Linha de Pesquisa: Telecomunicações
Orientador: Prof. Dr. Leonardo R. A. X. de Menezes

Brasília, Dezembro de 2003.

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

Desenvolvimento de aplicativos para o Wireless Village

Marcos Garrison Dytz

DISSERTAÇÃO DE GRADUAÇÃO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA UNIVERSIDADE DE BRASÍLIA, COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE ENGENHEIRO DE REDES DE COMUNICAÇÃO.

APROVADA POR:

LEONARDO R. A. X. DE MENEZES (ORIENTADOR)

PAULO HENRIQUE P. DE CARVALHO (EXAMINADOR 1)

ADSON FERREIRA DA ROCHA (EXAMINADOR 2)

BRASÍLIA, 16 DE DEZEMBRO DE 2003.

FICHA CATALOGRÁFICA

DYTZ, MARCOS GARRISON
DESENVOLVIMENTO DE APLICATIVOS PARA O WIRELESS VILLAGE [Brasília – DF] 2003.
xiv, 133p., 297mm (ENE/UnB, Graduação, Engenharia Elétrica, 2003)
Dissertação de graduação – Universidade de Brasília, Departamento de Engenharia Elétrica

Capítulos

1. Conceitos de Programação orientada a objetos	2. Java 2, Micro Edition
3. Servlets	4. Conceitos da linguagem XML
5. Sistema *Wireless Village*	6. Implementação

Apêndices

A. PhoneBook.sql	B. User.sql
C. Sap_CSPservlet.java	D. BD.java
E. SAP_Authentication.java	F. PhoneBookSearch.java
G. WVClient.java	H. NameSearch.java
I. PhoneSearch.java	J. PrimitiveRequest.java
K. AbstractRequest.java	L. WVConnector.java
M. WVConnection.java	N. Session.java
O. XMLMessage.java	P. XMLUtils.java

REFERÊNCIA BIBLIOGRÁFICA

DYTZ, M. G. Desenvolvimento de aplicativos para o Wireless Village. Brasília, 2003. Dissertação (Engenheiro de Redes de Comunicação) – Departamento de Engenharia Elétrica, UnB – DF.

CESSÃO DE DIREITOS

NOME DO AUTOR: Marcos Garrison Dytz
TÍTULO DA DISSERTAÇÃO DE GRADUAÇÃO: Desenvolvimento de aplicativos para o Wireless Village
GRAU / ANO: Engenheiro / 2003

_____

Marcos Garrison Dytz
SHIN QL 14 Conjunto 5 Casa 10
CEP 71530-055 – Brasília/DF – Brasil

Dedico este trabalho a meus pais.

AGRADECIMENTOS

Gostaria de agradecer minha família por todo suporte dado ao longo da minha vida, sem vocês não teria conseguido alcançar os êxitos que consegui. Uma pessoa que chegou mais tarde, mas que me ajudou muito foi minha namorada, Larisse, e fico feliz por ter ela ao meu lado nos momentos de tensão.

Agradeço aos professores Leonardo R. A. X. de Menezes e Paulo Henrique de P. de Carvalho pelo direcionamento que deram ao projeto durante nossas reuniões semanais e aos demais professores do Departamento de Engenharia Elétrica da Universidade de Brasília por todo apoio ao longo do meu curso.

Por fim, gostaria de agradecer pelo apoio e suporte prestados por todos os membros do projeto *Wireless Village* e os funcionários do Instituto Nokia de Tecnologia – INdT.

RESUMO

O propósito deste projeto foi modelar e implementar um aplicativo que utilize os recursos já desenvolvidos para o *Wireless Village* Server onde as especificações utilizadas para a o desenvolvimento deste servidor podem ser vistas nas referências [22], [23], [24], [25] e [26].

O *Wireless Village* é uma proposta feita pela Ericsson, Motorola e Nokia que visa especificar um padrão de troca de mensagens XML para um serviço de localização, um serviço de troca de mensagens instantâneas, um serviço de compartilhamento de conteúdo e um serviço de grupo, de forma que os quatro módulos trabalhando conjuntamente, proporcionem uma nova experiência para o usuário de aparelhos móveis.

Ao longo deste ano, foram implementados o serviço de presença e o serviço de troca de mensagens na parte cliente, um emulador para aparelho móvel; na parte servidor, um aplicativo Servlet; e os aplicativos responsáveis pelos serviços. Todo o sistema foi desenvolvido utilizando a linguagem Java e suas diversas APIs para executar as tarefas lógicas e a linguagem XML para definir o formato da troca de mensagens entre as diferentes entidades que formam o sistema.

Neste trabalho, otimizou-se o código já criado para o *Wireless Village* que se encontra anexado a este trabalho e se desenvolveu um novo aplicativo que utilize parte dos recursos disponibilizados pela implementação do *Wireless Village* Server.

O aplicativo em questão foi uma lista telefônica direta/reversa que permite que um usuário móvel faça uma procura pelo nome de uma pessoa a partir de seu número de telefone ou procure o telefone de uma pessoa a partir de seu nome. Para o desenvolvimento desse aplicativo, foi necessária a criação de novas classes e métodos para o aparelho móvel, a criação de novas classes e métodos para o Servlet, e a criação de novas tabelas no banco de dados do sistema. Todas essas tarefas precisaram ser feitas com uma total integração com o sistema já desenvolvido para evitar falhas de comunicação entre o aparelho móvel e o servidor *Wireless Village*.

ABSTRACT

The main objectives for this project was to design and develop an application that uses the resources already created for the Wireless Village Server in which all the specifications used for the development of this server can be found in the references [22], [23], [24], [25] and [26].

The Wireless Village is a proposal made by Ericsson, Motorola and Nokia which intends to rule a standard for the exchange of XML messages for localization services, instant messaging services, content sharing services and group services for the mobile platform in which all four modules working together would deliver a new experience of usage for mobile users.

Through out this year, the presence and instant messaging services were developed in the client side, an emulator of the mobile telephone; in the server side, a Servlet application; and the applications responsible for the two services. All development of the logical tasks of the system was made using the Java language and the several APIs offered by this language and the XML language was used for defining the format of messages interchanges among the different entities that compose the system.

In this work, the code already developed for the Wireless Village was optimized, this code can be found at the end of this dissertation and a new application was developed to use most of the resources available in the latest build of the Wireless Village Server.

The application developed was a direct/reverse phonebook search engine that allows the mobile user to search for the name of a person through a telephone number or find the telephone number of a person through its name. In the development of this application, it was necessary to write new classes and methods for the mobile emulator, new classes and methods for the Servlet and create new tables in the database of the system. All of these tasks had to be accomplished in total integration with the system already developed in order to avoid communication failures between the mobile telephone and the Wireless Village Server.

ÍNDICE

LISTA DE TABELAS

LISTA DE FIGURAS

LISTA DE ABREVIAÇÕES

Abreviação	Significado
ANSI	American National Standards Institute
API	Applications Programming Interface
CDC	Connected Device Configuration
CLDC	Connected Limited Device Configuration
DOM	Document Object Model
DTD	Document Type Definition
GCF	Generic Connection Framework
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
IM	Instant Messaging
IMPS	Instant Messaging and Presence Services
J2ME	Java 2 Micro Edition
J2SE	Java 2 Standard Edition
JAR	Java Archive
JAXB	Java API for XML Binding
JAXP	Java API for XML Parsing
JCA	Java Cryptography Architecture
JCE	Java Cryptography Extension
JNI	Java Native Interface
JSR	Java Specification Request
JVM	Java Virtual Machine
KVM	Kilo Virtual Machine
MID	Mobile Information Device
MIDP	Mobile Information Device Profile
MIN	Mobile Identification Number
MMS	Multimedia Messaging Service
RMI	Remote Method Invocation
SAP	Service Access Point
SAX	Simple API for XML
SMS	Short Messaging Service
SQL	Structured Query Language
UML	Universal Modeling Language
URI	Uniform Resource Indicator
URL	Uniform Resource Locator
WAP	Wireless Application Protocol
WML	Wireless Markup Language
WV	Wireless Village
WVS	Wireless Village Server
XML	Extensible Markup Language
XSL	Extensible Stylesheet Language
XSLT	XSL Transformations

INTRODUÇÃO

O desenvolvimento de aplicativos com a linguagem Java para dispositivos móveis tem aumentado em popularidade, algumas causas são as grandes vantagens que esta linguagem proporciona para o desenvolvedor que pode desenvolver o aplicativo visando um determinado aparelho, mas devido a portabilidade oferecida pela linguagem, poderia terminar em todos aparelhos móveis que possuem uma máquina virtual Java embutida em seus sistemas operacionais. Porém, a linguagem Java possui diversas outras vantagens que a tornam a linguagem indicada para o desenvolvimento de aplicativos móveis, tais como: linguagem de programação padrão para o desenvolvimento em *desktops*, servidores ou aparelhos móveis; diversas APIs que permitem o controle de todas funções do aparelho móvel; adaptada para um ambiente com poucos recursos de memória e processamento; linguagem em contínuo desenvolvimento.

A maneira como a linguagem Java obtém essa independência do sistema operacional é através da utilização de uma máquina virtual, sendo esta uma camada intermediária no processo de compilação. Esta camada adicional é a responsável por interpretar o *bytecode* gerado no ato de compilação do aplicativo. Devido a este fator de portabilidade, a linguagem Java pode ser utilizada no desenvolvimento de aplicativos para redes GSM, CDMA ou TDMA sem qualquer interferência do tipo de rede utilizada no código sendo desenvolvido, pois a linguagem faz toda a abstração da tecnologia utilizada, ou seja, um aplicativo feito para rodar em uma rede GSM rodará sem qualquer modificação em uma rede CDMA, sendo necessário somente que os dois aparelhos móveis tenham uma máquina virtual Java embutida neles. Daí não foi necessário um estudo sobre as diferentes tecnologias de transporte de sinal por uma rede celular para o desenvolvimento deste aplicativo, portanto não é feita uma introdução sobre estas tecnologias nesta dissertação.

Juntando todas vantagens obtidas com a linguagem Java com a padronização para troca de mensagens que pode se obter com a linguagem XML, então é possível o desenvolvimento de aplicativos que se comuniquem com qualquer outro dispositivo eletrônico que tenha uma implementação de XML, podendo ser um relógio digital, um PDA, um laptop, um servidor ou até um supercomputador. Portanto, a combinação destas duas linguagens permite que se desenvolva qualquer aplicativo que requeira uma interação com outros sistemas e é exatamente isto que o aplicativo desenvolvido neste trabalho tenta mostrar.

Este trabalho propõe toda a modelagem, estudo e implementação de um aplicativo para aparelhos móveis. O aplicativo, em questão, é uma lista telefônica direta/reversa que utiliza uma entrada do nome ou telefone da pessoa do qual a informação é requisitada usando a interface gráfica do aparelho celular para receber essa informação. A partir da entrada dessa informação, então a mesma é traduzida[1] de objeto Java em um documento XML. Em seguida, a mensagem XML é transmitida para o servidor do sistema utilizando o protocolo HTTP, o servidor recebe esse pedido a partir do Servlet e traduz a mensagem XML para forma de objetos Java. Com o pedido em

[1] O termo traduzir será usado durante todo o trabalho para representar a transformação da informação de um formato para outro.

forma de objetos Java, então o servidor faz uma consulta no banco de dados do sistema de acordo com a informação passada pelo usuário no celular. Obtido o resultado da consulta ao banco de dados, o servidor traduz este resultado que está na forma de um objeto Java em um documento XML e transmite essa mensagem para o aparelho celular. Para concluir, o aparelho móvel traduz a resposta em forma de documento XML para um objeto Java e apresenta o resultado dessa consulta no aparelho móvel do usuário.

Como pode ser visto acima, o sistema é basicamente composto de três entidades: o aparelho móvel, o Servlet e o banco de dados. Além disso, todo o sistema utiliza duas linguagens, Java para o controle de todas operações lógicas executadas pelo sistema e XML para toda troca de informação entre o aparelho móvel e o servidor.

Este trabalho foi dividido em oito capítulos, sendo o primeiro esta introdução e o último sendo a conclusão. O capítulo 2 é composto por uma breve descrição de programação orientada a objetos, conceitos da linguagem Java e uma introdução a padrão de modelagem UML, onde todos esses conceitos são muito úteis para entender todo o restante desta dissertação. O capítulo 3 detalha o ambiente de desenvolvimento *Java 2, Micro Edition* que foi utilizado para o desenvolvimento do sistema no lado do aparelho móvel. O capítulo 4 explica os passos para o desenvolvimento de Servlets onde esta tecnologia foi utilizada no desenvolvimento do *container* para recebimento dos pedidos XML por parte do servidor. O capítulo 5 traz uma descrição dos detalhes que compõem a linguagem XML, assim como as diferentes técnicas para tradução de informação Java em XML, e vice-versa. O capítulo 6 apresenta o *Wireless Village* que foi o sistema utilizado como base para o desenvolvimento deste aplicativo. Por último, o capítulo 7 apresenta todos os detalhes da implementação do sistema e as ferramentas utilizadas.

É importante se ressaltar que alguns termos foram mantidos em sua forma original, em inglês, devido a sua utilização na literatura técnica em língua inglesa ou por nenhum termo traduzir esses termos para o português passando a mesma idéia expressa na língua inglesa. Todos esses termos se encontram em itálico ao longo deste documento. Além disso, toda classe, método ou qualquer referência à código em linguagem Java possui uma formatação diferente para facilitar na visualização destes termos, a formatação usada foi fonte Courier, número 10.

CONCEITOS DE PROGRAMAÇÃO ORIENTADA A OBJETOS

Para uma compreensão dos capítulos posteriores desse trabalho, faz-se necessário um estudo introdutório sobre programação orientada a objeto que será apresentado neste capítulo. A seguir, pode-se encontrar uma breve introdução sobre os conceitos utilizado na linguagem Java e uma descrição da linguagem de modelagem UML. O texto utilizado foi retirado, em partes, da referência [12].

2.1 Programação Orientada a Objeto

Programação orientada a objetos (POO) é uma metodologia de programação adequada ao desenvolvimento de sistemas de grande porte, provendo modularidade e reusabilidade. A POO introduz uma abordagem na qual o programador visualiza seu programa em execução como uma coleção de objetos co-operantes que se comunicam através de mensagens. Cada um dos objetos é instância de uma classe e todas as classes formam uma hierarquia de classes unidas via relacionamento de herança. Existem alguns aspectos importantes na definição de POO:

- Usa objetos, e não funções ou procedimentos como seu bloco lógico fundamental de construção de programas.
- Objetos comunicam-se através de mensagens.
- Cada objeto é instância de uma classe.
- Classes estão relacionadas às outras através de mecanismos de herança e polimorfismo.

Programação orientada a objetos dá ênfase à estrutura de dados, adicionando funcionalidade ou capacidade de processamento a estas estruturas. Em linguagens tradicionais, a importância maior é atribuída a processos e sua implementação em subprogramas. Em linguagens orientadas a objetos, ao invés de passar dados a procedimentos, requisita-se que objetos realizem operações neles próprios.

Alguns aspectos são fundamentais na definição de programação orientada a objetos:

2.1.1 Objetos

Na visão de uma linguagem imperativa tradicional (estruturada), os objetos aparecem como uma única entidade autônoma que combina a representação da informação (estruturas de dados) e sua manipulação (procedimentos), uma vez que possuem capacidade de processamento e armazenam um estado local. Pode-se dizer que um objeto é composto de:

- *Propriedades*: são as informações, estruturas de dados que representam o estado interno do objeto. Em geral, não são acessíveis aos demais objetos.

- *Comportamento*: conjunto de operações, chamados de métodos, que agem sobre as propriedades. Os métodos são ativados (disparados) quando o objeto recebe uma mensagem solicitando sua execução. Embora não seja obrigatório, em geral uma mensagem recebe o mesmo nome do método que ela dispara. O conjunto de mensagens que um objeto está apto a receber está definido na sua interface.
- *Identidade*: é uma propriedade que diferencia um objeto de outro; ou seja, seu nome.

Enquanto que os conceitos de dados e procedimentos são freqüentemente tratados separadamente nas linguagens de programação tradicionais, em POO eles são reunidos em uma única entidade: o objeto. A Figura 2.1 apresenta uma visualização para um objeto.

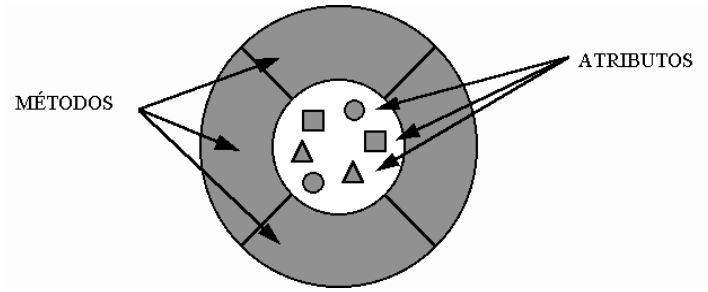

Figura 2.1 - Representação para um objeto

No mundo real não é difícil a identificação de objetos (em termos de sistemas, objetos são todas as entidades que podem ser modeladas, não apenas os nossos conhecidos objetos inanimados).

Uma vez que objetos utilizam o princípio da abstração de dados, o encapsulamento de informação proporciona dois benefícios principais para o desenvolvimento de sistemas:

- *Modularidade*: o código-fonte para um objeto pode ser escrito e mantido independentemente do código-fonte de outros objetos. Além disso, um objeto pode ser facilmente migrado para outros sistemas.
- *Ocultamento de informação*: um objeto tem uma interface pública que os outros objetos podem utilizar para estabelecer comunicação com ele. Mas, o objeto mantém informações e métodos privados que podem ser alterados a qualquer hora sem afetar os outros objetos que dependem dele. Ou seja, não é necessário saber como o objeto é implementado para poder utilizá-lo.

2.1.2 Mensagens

Um objeto sozinho não é muito útil e geralmente ele aparece como um componente de um grande programa que contém muitos outros objetos. Através da interação destes objetos pode-se obter uma grande funcionalidade e comportamentos mais complexos.

Objetos de software interagem e se comunicam com os outros através de mensagens. Quando o objeto A deseja que o objeto B execute um de seus métodos, o objeto A envia uma mensagem ao objeto B. Algumas vezes o objeto receptor precisa de mais informação para que ele saiba exatamente o que deve fazer; esta informação é transmitida juntamente com a mensagem através de *parâmetros.*

Uma mensagem é formada por três componentes básicos:

- O objeto a quem a mensagem é endereçada (receptor);
- O nome do método que se deseja executar;
- Os parâmetros (se existirem) necessários ao método.

2.1.3 Classe

"É a definição dos atributos e funções de um tipo de objeto. Cada objeto individual é então criado com base no que está definido na classe. Por exemplo, *homo sapiens* é uma classe de mamífero; cada ser humano individual é um objeto dessa classe."

Objetos de estrutura e comportamento idênticos são descritos como pertencendo a uma classe, de tal forma que a descrição de suas propriedades pode ser feita de uma só vez, de forma concisa, independente do número de objetos idênticos em termos de estrutura e comportamento que possam existir em uma aplicação. A noção de um objeto é equivalente ao conceito de uma variável em programação convencional, pois especifica uma área de armazenamento, enquanto que a classe é vista como um tipo abstrato de dados, uma vez que representa a definição de um tipo.

Cada objeto criado a partir de uma classe é denominado de *instância* dessa classe. Uma classe provê toda a informação necessária para construir e utilizar objetos de um tipo, cada instância pertence a uma classe e uma classe pode possuir múltiplas instâncias. Devido ao fato de todas as instâncias de uma classe compartilharem as mesmas operações, qualquer diferença de respostas a mensagens aceitas por elas é determinada pelos valores das variáveis de instância.

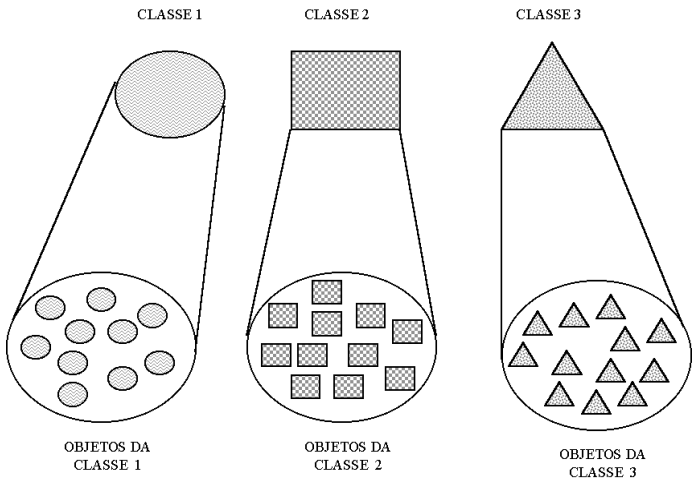

Figura 2.2 - Relacionamento entre classes e objetos

A Figura 2.2 ilustra o relacionamento entre classes e objetos. Cada objeto instanciado a partir de uma classe possui as propriedades e os comportamentos definidos na classe, da mesma maneira que uma variável incorpora as características do seu tipo. A existência de classes proporciona um ganho em reusabilidade, pois o código das operações e a especificação da estrutura de um número potencialmente infinito de objetos estão definidos em um único local, a classe. Cada vez que um novo objeto é instanciado ou que uma mensagem é enviada, a definição da classe é reutilizada. Caso não existissem classes, para cada novo objeto criado, seria preciso uma definição completa do objeto.

O maior benefício proporcionado pela utilização das classes é a reusabilidade de código, uma vez que todos os objetos instanciados a partir dela incorporam as suas propriedades e seu comportamento.

2.1.4 Metaclasses

Uma metaclasse é uma classe de classes. Pode-se julgar conveniente que, em uma linguagem ou ambiente, classes também possam ser manipuladas como objetos. Por exemplo, uma classe pode conter variáveis contendo informações úteis, como:

- o número de objetos que tenham sido instanciados da classe até certo instante;
- um valor médio de determinada propriedade, calculado sobre os valores específicos desta propriedade nas instâncias (por exemplo, média de idade de empregados).

2.1.5 Métodos

Um método implementa algum aspecto do comportamento do objeto. Comportamento é a forma como um objeto age e reage, em termos das suas trocas de estado e troca de mensagens.

Um método é uma função ou procedimento que é definido na classe e tipicamente pode acessar o estado interno de um objeto da classe para realizar alguma operação. Pode ser pensado como sendo um procedimento cujo primeiro parâmetro é o objeto no qual deve trabalhar. Este objeto é chamado receptor. Abaixo é apresentada uma notação possível para o envio de uma mensagem (invocação do método).

Construtores são usados para criar e inicializar objetos novos. Tipicamente, a inicialização é baseada em valores passados como parâmetros para o construtor. Destrutores são usados para destruir objetos. Quando um destrutor é invocado, as ações definidas pelo usuário são executadas, e então a área de memória alocada para o objeto é liberada. Em algumas linguagens, como C++, o construtor é chamado automaticamente quando um objeto é declarado. Em outras, como *Object Pascal*, é necessário chamar explicitamente o construtor antes de poder utilizá-lo.

Um exemplo de utilização de construtores e destrutores seria gerenciar a quantidade de objetos de uma determinada classe que já foram criados até o momento. No construtor pode-se colocar código para incrementar uma variável e no destrutor o código para decrementá-la.

2.1.6 Herança

O conceito de herança é fundamental na técnica de orientação a objetos. A herança permite criar um novo tipo de objeto - uma nova classe - a partir de outra já existente.

A nova classe mantém os atributos e a funcionalidade da classe da qual deriva; por isso, dizemos que ela "herda" as características daquela classe. Ao mesmo tempo, ela pode receber atributos e funções especiais não encontrados na classe original.

Uma das vantagens da herança é a facilidade de localizar erros de programação. Por exemplo, caso um objeto derivado de outro apresente um erro de funcionamento; se o objeto original funcionava corretamente, é claro que o erro está na parte do código que implementa as novas características do objeto derivado. A herança permite, também, reaproveitar o código escrito anteriormente, adaptando-o às novas necessidades.

Isso é muito importante porque os custos de desenvolvimento de software são muitos elevados. A mão-de-obra altamente especializada é cara; o processo é demorado e sujeito a ocorrências inesperadas.

2.1.7 Polimorfismo

Polimorfismo refere-se à capacidade de dois ou mais objetos responderem à mesma mensagem, cada um a seu próprio modo. A utilização da herança torna-se fácil com o polimorfismo. Desde que não seja necessário escrever um método com nome diferente para responder a cada mensagem, o código é mais fácil de entender.

Outra forma simples de polimorfismo permite a existência de vários métodos com o mesmo nome, definidos na mesma classe, que se diferenciam pelo tipo ou número de parâmetros suportados. Isto é conhecido como *polimorfismo paramétrico*, ou *sobrecarga de operadores*[2]. Neste caso, uma mensagem poderia ser enviada a um objeto com parâmetros de tipos diferentes (uma vez inteiro, outra real, por exemplo), ou com número variável de parâmetros. O nome da mensagem seria o mesmo, porém o método invocado seria escolhido de acordo com os parâmetros enviados.

Alguns benefícios proporcionados pelo polimorfismo:

- *Legibilidade do código*: a utilização do mesmo nome de método para vários objetos torna o código de mais fácil leitura e assimilação, facilitando muito a expansão e manutenção dos sistemas.
- *Código de menor tamanho*: o código mais claro torna-se também mais enxuto e elegante. Pode-se resolver os mesmos problemas da programação convencional com um código de tamanho reduzido.

2.1.8 Vantagens da POO

[2] Em inglês, *overloading*.

A POO tem alcançado tanta popularidade, devido às vantagens que ela traz. A reusabilidade de código é, sem dúvida, reconhecida como a maior vantagem da utilização de POO, pois permite que programas sejam escritos mais rapidamente. Todas as empresas sofrem de deficiência em seus sistemas informatizados para obter maior agilidade e prestar melhores serviços a seus clientes. Um levantamento feito na AT&T, a gigante das telecomunicações nos EUA, identificou uma deficiência da ordem de bilhões de linhas de código. Uma vez que a demanda está sempre aumentando, procura-se maneiras de desenvolver sistemas mais rapidamente, o que está gerando uma série de novas metodologias e técnicas de construção de sistemas (por exemplo, ferramentas CASE). A POO, através da reusabilidade de código, traz uma contribuição imensa nesta área, possibilitando o desenvolvimento de novos sistemas utilizando-se muito código já existente. A maior contribuição para reusabilidade de código é apresentada pela herança.

Escalabilidade pode ser vista como a capacidade de uma aplicação crescer facilmente sem aumentar demasiadamente a sua complexidade ou comprometer o seu desempenho. A POO é adequada ao desenvolvimento de grandes sistemas uma vez que se pode construir e ampliar um sistema agrupando objetos e fazendo-os trocar mensagens entre si. Esta visão de sistema é uniforme, seja para pequenos ou grandes sistemas (logicamente, deve-se guardar as devidas proporções).

O encapsulamento proporciona ocultamento e proteção da informação. Acessos a objetos somente podem ser realizados através das mensagens que ele está habilitado a receber. Nenhum objeto pode manipular diretamente o estado interno de outro objeto. De modo que, se houver necessidade de alterar as propriedades de um objeto ou a implementação de algum método, os outros objetos não sofrerão nenhum impacto, desde que a interface permaneça idêntica. Isto diminui em grande parte os esforços despendidos em manutenção. Além disso, para utilizar um objeto, o programador não necessita conhecer a fundo a sua implementação.

O polimorfismo torna o programa mais enxuto, claro e fácil de compreender. Sem polimorfismo, seriam necessárias listas enormes de métodos com nomes diferentes, mas comportamento similar. Na programação, a escolha de um entre os vários métodos seria realizada por estruturas de múltipla escolha (*case*) muito grandes. Em termos de manutenção, isto significa que o programa será mais facilmente entendido e alterado.

A herança também torna a manutenção mais fácil. Se uma aplicação precisa de alguma funcionalidade adicional, não é necessário alterar o código atual. Simplesmente cria-se uma nova geração de uma classe, herdando o comportamento antigo e adicionando-se o novo comportamento, ou redefinindo-se o comportamento antigo.

2.1.9 Desvantagens da POO

Apesar de das inúmeras vantagens, a POO tem também algumas desvantagens. A apropriação é apresentada tanto como uma vantagem como uma desvantagem, porque a POO nem sempre soluciona os problemas elegantemente. Enquanto que a mente humana parece classificar objetos em categorias (classes) e agrupar essas classes em relacionamentos de herança, o que ela realmente faz não é tão simples. Em vez disso, objetos com características mais ou menos similares, e não precisamente definidas, são

reunidos em uma classificação. A POO requer definições precisas de classes; definições flexíveis e imprecisas não são suportadas. Na mente humana, essas classificações podem mudar com o tempo. Os critérios para classificar objetos podem mudar significativamente. A apropriação utilizada na POO torna-a muito rígida para trabalhar com situações dinâmicas e imprecisas.

Além disso, algumas vezes não é possível decompor problemas do mundo real em uma hierarquia de classes. Negócios e pessoas têm freqüentemente regras de operações sobre objetos que desafiam uma hierarquia limpa e uma decomposição orientada a objetos. O paradigma de objetos não trata bem de problemas que requerem limites nebulosos e regras dinâmicas para a classificação de objetos.

Isto leva ao próximo problema com POO: fragilidade. Desde que uma hierarquia orientada a objetos requer definições precisas, se os relacionamentos fundamentais entre as classes-chave mudam, o projeto original orientada a objetos é perdido. Torna-se necessário reanalisar os relacionamentos entre os objetos principais e reprojetar uma nova hierarquia de classes. Se existir uma falha fundamental na hierarquia de classes, o problema não é facilmente consertado.

2.2 Linguagem Java

Java é uma linguagem orientada a objeto recente. Ela foi idealizada no ano de 1991 e a primeira versão pública foi lançada em março de 1995. Isso tornou possível analisar os defeitos das outras linguagens anteriores e assimilar as vantagens das mesmas. Por causa da grande quantidade de programadores C e C++, todos os operadores lógicos, aritméticos foram mantidos da mesma forma em Java. Mas as declarações de baixo nível, como ponteiros, foram abandonadas por causar vários problemas de desenvolvimento. Outra causa de dificuldades no desenvolvimento em C era o gerenciamento manual da memória com os comandos `malloc()`, e `free()`. Em Java, esse problema foi resolvido com a figura do G*arbage Collector* que desaloca a memória de objeto quando ele não é mais necessário. O modelo de gerenciamento de memória do Java é baseado em referências aos objetos. Quando não existe mais nenhuma referência a um objeto no gerenciador de memória, automaticamente o G*arbage Collector* desaloca a memória do objeto. Um estudo mais detalhado da linguagem Java pode ser visto nas referências [9], [10] e [11].

A principal funcionalidade da linguagem Java é sua portabilidade, conhecida pelo termo "*Write Once Run Anywhere*" – Escreva uma vez, rode em qualquer lugar – (WORA). Essa vantagem de portabilidade é muito utilizada para se desenvolver o aplicativo para *desktops* e servidores. Entretanto, as APIs feitas para a programação não são as mais adequadas para dispositivos móveis, em geral.

As características dos dispositivos móveis são drasticamente diferentes das utilizadas para servidores ou *desktops*. Para endereçar esses requerimentos de ambientes diferentes e domínios para os aplicativos, as APIs Java junto com a máquina virtual embutida são lançadas em três edições. Todas três edições completamente suportam o princípio WORA e provêem uma portabilidade completa. As três edições são:

- *Java 2 Enterprise Edition* (J2EE) para desenvolver soluções escaláveis para empresas;

- *Java 2 Standard Edition* (J2SE) para o desenvolvimento de aplicativos/applets em PCs;
- *Java 2 Micro Edition* (J2ME) para dispositivos móveis.

Por fim, quando Java foi introduzida em 1995, pareceu que o futuro da computação seriam os *applets*, pequenos programas que podiam ser baixados e executados de acordo com a demanda. Porém as conexões lentas com a Internet fizeram com que os *applets* não conseguissem sua entrada no mercado, logo a linguagem Java, como um plataforma, não decolou até o advento de *Servlet*s, programas Java que rodam nos servidores. Essa foi a primeira revolução dessa linguagem, a força do Java para o lado do servidor.

A segunda revolução é a explosão de dispositivos pequenos com Java, e está acontecendo agora. O mercado para dispositivos pequenos está expandindo rapidamente e Java é importante por duas razões. Primeiro, desenvolvedores podem escrever código e rodar o mesmo em dezenas de plataformas pequenas, sem mudanças. Segundo, Java tem importantes funcionalidades de segurança para o código baixado.

Um estudo mais aprofundado sobre o J2ME pode ser visto no Capítulo 3 e um estudo sobre Servlets pode ser visto no Capítulo 4.

2.2.1 Plataforma Java

No cenário atual, existem várias plataformas de desenvolvimento, entre elas estão Windows, Unix e Macintosh. Um software desenvolvido para um desses sistemas operacionais (SO) exige adaptações e recompilação do código-fonte para ser executado em outro SO. Isso é um grave problema dentro do cenário heterogêneo que é a Internet no qual essas várias plataformas formam uma rede mundial. A plataforma Java desenvolveu uma forma de resolver esse problema. Basicamente essa plataforma interpreta os códigos-binários feitos para uma máquina virtual que é executada pelos sistemas operacionais citados anteriormente. Isso cria a plataforma neutra, um executável que roda em qualquer sistema operacional sem a necessidade de qualquer modificação no código. Esse novo ambiente pode ser dividido entre duas partes principais.

Máquina Virtual Java (JVM): Uma máquina virtual que é emulada nos processadores atuais ou implementada em um hardware específico.

Interface de programação de aplicação Java (Java API): Interface padrão para o desenvolvimento de aplicações para esta linguagem.

A Figura 2.3 mostra como essas camadas se organizam e alguns aplicativos que podem ser feitos em Java.

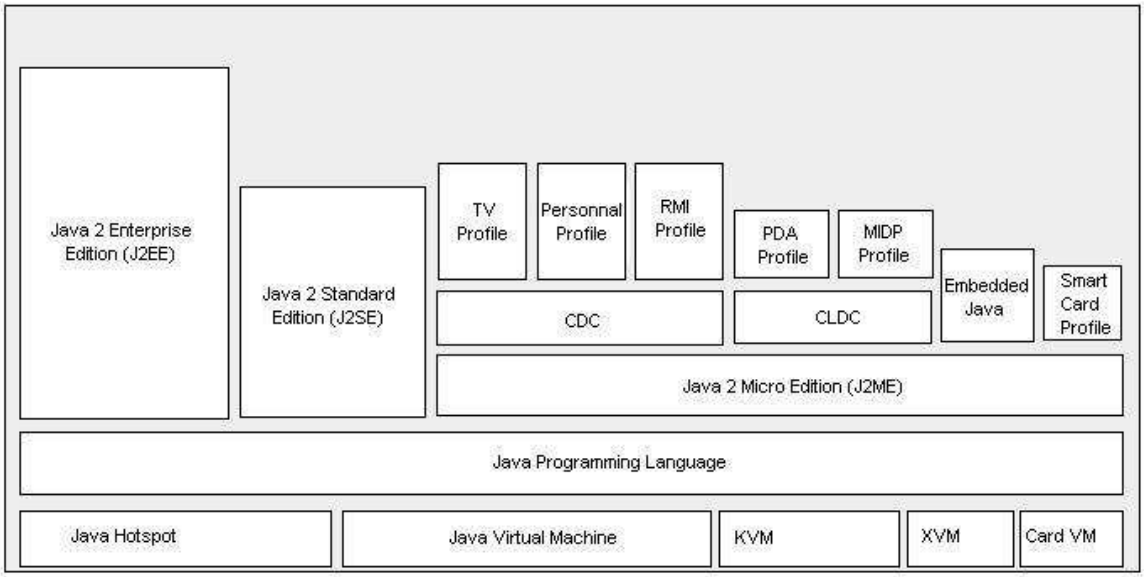

Figura 2.3 - Plataforma Java

2.2.2 Máquina Virtual Java

A máquina virtual Java é a chave para a independência dos binários dessa linguagem que tem a extensão `class`. As instruções são codificadas para os binários para uma plataforma virtual que poder ser implementada em qualquer sistema operacional. Os métodos dentro de uma classe são indexados por nomes e isso torna possível, em tempo de execução, que um novo método possa ser inserido ou cancelado.

2.2.3 Java APIs[3]

A interface de programação de aplicativos, a API, do Java apresenta inúmeras classes já implementadas para facilitar ao programador na construção de seu aplicativo, entre elas estão as JFCs. JFC corresponde à abreviatura de *Java Foundation Classes*, que abrange uma série de atributos para ajudar na construção de interfaces gráficas, as GUIs. O JFC foi lançado em 1997 na conferência de desenvolvedores de *JavaOne* e contém os seguintes pacotes:

- *Componentes Swing*: Inclui todos os componentes visuais desde botões a janelas.
- *Suporte a Look and Feel*: Oferece a qualquer programa que utilize componentes Swing a escolha de sua aparência.
- *API acessíveis*: Permite opções de acessibilidade para leitores viva-voz e dispositivos de impressão em Braile para fornecer informações da interface do usuário.
- *Java 2D* (Java 2 somente): Permite que desenvolvedores incorporem facilmente gráficos 2D de alta qualidade, textos e imagens em aplicações.

[3] O termo API será usado durante todo esse trabalho para definir um conjunto de bibliotecas que implementem uma determinada função da linguagem Java.

- *Suporte a Drag and Drop* (Java 2 somente): Fornece a operação de “arrastar e soltar” em aplicações Java, e entre estas e outras aplicações nativas.

2.2.4 Manipulação de eventos

Qualquer sistema operacional que suporte interfaces gráficas de usuário precisa constantemente monitorar o ambiente buscando por eventos tais como teclas pressionadas ou cliques do mouse para então informar esses eventos aos programas que estão em execução. Cada programa então decide o que fazer em resposta a esses eventos. A linguagem Java adota uma metodologia em termos de recursos e, conseqüentemente, na complexidade resultante. Dentro dos limites dos eventos que o pacote AWT (*Abstract Window Toolkit* – kit de ferramentas de janelas abstratas) conhece, pode-se controlar totalmente como os eventos são transmitidos desde as origens de evento (como botões e barras de rolagem) até os ouvintes de eventos.

Pode-se designar qualquer objeto para ser um ouvinte de evento – na prática, escolhe-se um objeto que possa efetuar convenientemente a resposta desejada ao evento. As origens dos eventos têm métodos que permitem registrar ouvintes de eventos neles. Quando um evento ocorre na origem, esta envia uma notificação do mesmo para todos os objetos ouvintes que foram registrados para esse evento. Como era de se esperar de uma linguagem orientada a objeto como a Java, a informação sobre o evento é encapsulada em um objeto *evento*. Em Java, todos os objetos evento, no fim, derivam da classe `Java.util.EventObject`. Segue alguns exemplos de classes ouvintes e suas respectivas ações que resultam num evento:

- `ActionListener`: Usuário clica num botão, pressiona a tecla Return enquanto está digitando em um campo de texto, ou seleciona algum item de menu.
- `WindowListener`: Usuário fecha uma janela principal.
- `MouseListener`: Usuário pressiona um botão de mouse enquanto o cursor está sobre um componente.
- `MouseMotionListener`: Usuário move o mouse sobre um componente.
- `ComponentListener`: Componente se torna visível.
- `FocusListener`: Componente recebe foco do teclado.
- `ListSelectionListener`: Tabela ou lista alteram seu conteúdo.

2.2.5 Tratamento de Exceções

Examinam-se agora os mecanismos de que a linguagem Java dispõe para lidar com dados incorretos e programas com erros. Quando o programa encontra erros durante sua execução, o ideal seria notificar o usuário do erro e permitir ao usuário salvar todo trabalho e sair do programa de forma adequada. Para tanto, a linguagem Java usa uma forma de captura de erros chamada, apropriadamente, de tratamento de exceções.

A reação tradicional a um erro num método é retornar um código de erro especial, que o método chamado possa analisar. Infelizmente, nem sempre é possível retornar um código de erro. Pode ocorrer de não haver uma maneira óbvia de distinguir entre dados válidos e inválidos. Em vez disso, a linguagem Java permite que todo

método tenha uma forma de saída alternativa caso seja incapaz de finalizar sua tarefa normalmente. Nessa situação, o método não retorna nenhum valor. Em vez disso, ele lança um objeto que encapsula a informação do erro. Além disso, a linguagem Java não ativa o código que chamou o método; em vez disso, o mecanismo de tratamento de exceções começa sua busca por um manipulador de exceção que possa lidar com essa condição de erro particular. Em Java, um objeto exceção é sempre instância de uma classe derivada de `Throwable`.

2.3 UML

A implementação de um aplicativo em sistemas orientados a objetos se torna mais eficiente se for feito uma modelagem, isto é, uma representação visual de uma especificação, um projeto ou um sistema por meio de um ou mais diagramas. O modelo deve apontar o essencial de alguns aspectos do que está sendo feito sem dar detalhes desnecessários. Seu objetivo é permitir às pessoas envolvidas no desenvolvimento pensar sobre e discutir problemas e soluções sem desviar-se.

Desenvolver um modelo para softwares complexos antes de iniciar a implementação ou renovação do projeto é tão essencial quanto ter uma planta para uma grande construção. Bons modelos são essenciais para comunicação entre equipes de projeto e para garantir solidez arquitetural. São construídos modelos de sistemas complexos porque é difícil compreender o projeto como sistema inteiro. Assim como a complexidade dos sistemas aumenta, assim aumenta a importância das boas técnicas de modelagem. Uma linguagem de modelagem deve conter:

- Elementos do Modelo – conceitos e semântica de modelagem fundamentais;
- Notação – representação visual dos elementos do modelo;
- *Guidelines* – idioma de uso em trocas.

Em frente à crescente complexidade dos sistemas, visualização e modelagem através de uma linguagem padrão e rigorosa se torna essencial. A UML é uma resposta bem definida e amplamente aceita para esta necessidade. É a linguagem de modelagem visual com chance para construir sistemas orientados a objeto.

A UML (*Unified Modeling Language*) é a sucessora dos métodos de projeto e análise orientada a objetos mais importantes dos anos 80 e 90. Apesar de ter nascido de uma junção de métodos, ela é uma linguagem de modelagem, isto é, a notação, principalmente gráfica, utilizada para expressar projetos. Esta linguagem tem sido adotada como padrão pelo OMG (*Object Management Group*) e a familiaridade com ele parece ter se tornado uma habilidade fundamental para engenheiros de software.

Este padrão, entre outras coisas, visa a modelagem de sistemas e não apenas o conceito de Orientação à Objeto, o estabelecimento de uma união fazendo com que alguns métodos conceituais sejam também executáveis e uma linguagem de modelagem usável tanto pelo homem quanto pela máquina.

2.3.1 Objetivos da Modelagem com UML

Em programas complexos, a modelagem bem estruturada constitui uma base estável para o desenvolvimento de um software otimizado. A modelagem faz uma

simplificação da realidade para o sistema possibilitado assim, a visualização e o controle da arquitetura do sistema. Através da modelagem, há uma comunicação da estrutura desejada com o comportamento do sistema. Ela facilita o controle de riscos.

Usando a UML, pode-se ver o sistema como ele é ou da forma como quer que seja, possibilitando um melhor entendimento do sistema que está sendo construído. Tem-se também a especificação da estrutura e do comportamento do sistema, um *template*[4] que pode servir de guia para a construção do sistema além de uma documentação das decisões tomadas.

A UML suporta especificações independentes de linguagem de programação ou processos de desenvolvimento, isto é, pode suportar todas as linguagens de programação razoáveis e vários métodos e processos de modelos de criação. Pode também suportar múltiplas linguagens de programação e métodos de desenvolvimento sem grandes dificuldades.

2.3.2 Fases de Desenvolvimento de Sistemas

O desenvolvimento de sistemas em UML se inicia pela Análise dos Requisitos onde se verificam as necessidades dos usuários expressas através de Use-Cases. Em seguida, se tem a Análise, na qual se obtêm as primeiras abstrações (classes e objetos), os mecanismos de domínio do problema e a colaboração entre as classes. A seguir, se passa para o Projeto (Design) onde se faz a expansão da análise em soluções técnicas, criação de novas classes, interface com periféricos, Banco de Dados, etc, e detalhamento da fase de programação.

Após estas fases iniciais, vem a fase da Programação, onde ocorre a conversão da fase de projetos para o código da linguagem de programação escolhida. É recomendável que se evite a mentalização do código nas três fases anteriores. Por fim, a fase de Testes, que pode ser dividida em três: o teste de unidade é geralmente realizado pelo programador e visa testar as classes e objetos gerados; o teste de integração verifica a cooperação das classes de acordo com o modelo; e o teste de aceitação que analisa a funcionalidade do sistema como o proposto nas primeiras especificações.

Em alguns casos, é desejado que se faça uma manutenção do sistema após sua implantação. Nestes casos, da fase de testes, volta-se ao início, refazendo todas as fases até fazer novos testes com o programa após a manutenção.

Uma orientação sobre todos os conceitos e elementos utilizados em UML pode ser visto na referência [8].

2.3.3 Considerações finais a respeito de UML

A UML é uma linguagem de modelagem com regras e vocabulário focado na representação física e conceitual de um sistema. Ela facilita a comunicação e cria modelos precisos sem ambigüidades e completos. Os modelos podem ser conectados a uma série de linguagens de programação. Além disso, ela provê mecanismos de documentação da arquitetura do sistema e de todos os seus detalhes.

[4] Em Port., modelo.

O sucesso de utilização da UML está relacionado com o método de desenvolvimento utilizado que descreve o que fazer, como fazer, quando fazer e porque deve ser feito.

A UML pode ser usada em sistemas de informação, sistemas financeiros, telecomunicações, serviços WEB distribuídos, etc. A UML, em sua forma atual, é esperada que seja a base para muitas ferramentas, incluindo para modelagem visual, simulação e desenvolvimento de ambientes. Como interessantes ferramentas de integração são desenvolvidas, padrões de implementação baseadas no UML vão se tornar amplamente disponibilizadas.

Todo o projeto implementado neste trabalho se apresente modelado em UML.

JAVA 2, MICRO EDITION

O *Java 2, Micro Edition* (J2ME) é a ferramenta que será utilizada para o desenvolvimento de toda parte cliente do aplicativo. Essa edição da plataforma Java possui algumas restrições em relação a seu uso que devem ser estudadas para se obter o desempenho esperado da linguagem para aparelhos móveis. Os livros referenciados nos itens [2], [3], [4] e [19] foram bastante utilizados para escrever todo esse capítulo.

3.1 Introdução ao J2ME

De acordo com a referência [29], a plataforma J2ME oferece o poder e os benefícios da linguagem Java para dispositivos móveis – incluindo uma interface gráfica flexível, modelo de segurança robusto, grande quantidade de protocolos de transmissão embutidos, e suporte para aplicativos usados pela rede ou localmente. Portanto, com o J2ME, os aplicativos podem ser escritos somente uma vez para uma grande quantidade de dispositivos móveis, podendo ser baixados dinamicamente da Internet e utilizando toda capacidade nativa dos dispositivos.

A Figura 3.1 apresenta a arquitetura J2ME com as configurações e perfis utilizados de acordo com o dispositivo móvel para o qual o aplicativo está sendo desenvolvido. Portanto, pode-se notar uma diferença na Máquina Virtual Java utilizada devido aos recursos disponibilizados por cada dispositivo.

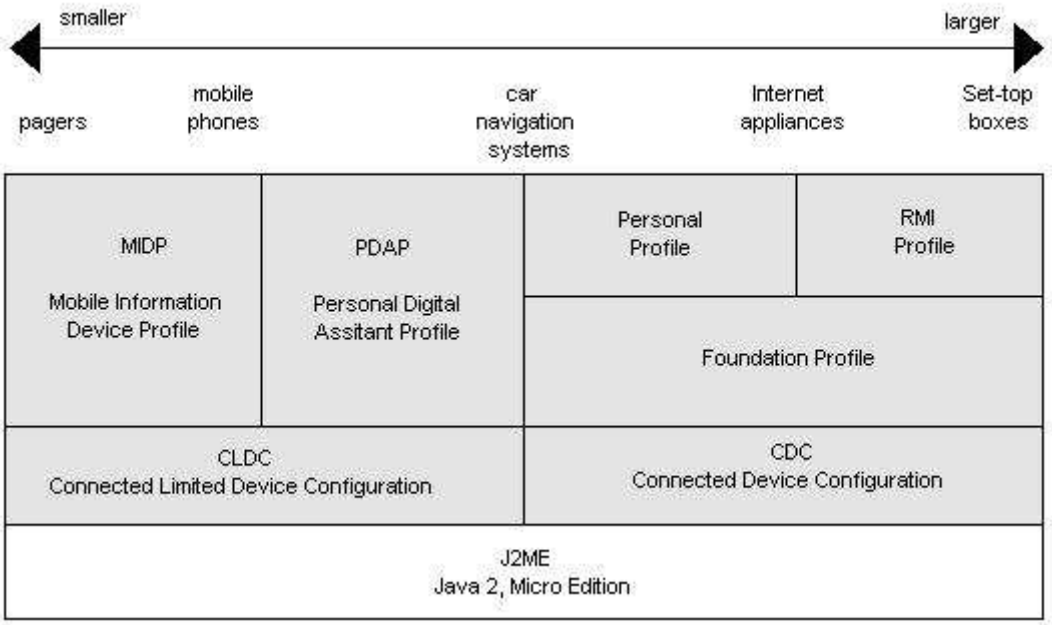

Figura 3.1: Arquitetura J2ME de acordo com os dispositivos.

Desde seu lançamento, essa plataforma tenta cobrir uma variedade de dispositivos móveis em uma variedade de segmentos de mercado, porém o J2ME é dividido em configurações direcionadas para um conjunto de dispositivos. A

configuração mais apropriada para telefones celulares é a *Connected Limited Device Configuration* (CLDC) que foi desenvolvida para dispositivos com 160 a 512 kB de memória disponível para a plataforma Java, alimentados por bateria, lentos, e com conexões possivelmente intermitente. Utilizando os recursos dados pelo CLDC está o *Mobile Information Device Profile* (MIDP) que especifica um conjunto de APIs apropriadas para dispositivos de informações móveis como telefones. A outra configuração disponível é a *Connected Device Configuration* (CDC) que foi desenvolvida visando PDAs avançados, sistemas automotivos e aparelhos televisivos. Será dado um enfoque maior para o CLDC, pois o sistema desenvolvido neste projeto é mais voltado para aparelhos celulares.

As principais vantagens oferecidas pela linguagem Java para sua implementação em dispositivos móveis são:

- Consistência embutida entre os produtos em termo de rodar em qualquer lugar, a qualquer hora e em qualquer dispositivo;
- Portabilidade do código;
- Utilização da mesma linguagem de programação Java;
- Envio pela rede com segurança;
- Aplicativos escritos com o J2ME são compatíveis para trabalhos com o J2SE e J2EE.

3.2 A arquitetura J2ME

A arquitetura J2ME define as configurações, perfis e pacotes opcionais como elementos para se implementar um ambiente *runtime* Java completo que se encaixe com os requerimentos dados por uma grande quantidade de dispositivos. Cada combinação é otimizada para uma capacidade de memória, poder de processamento, e entrada/saída de uma categoria de dispositivo. O resultado é uma plataforma Java comum que oferece uma rica experiência de uso para os usuários.

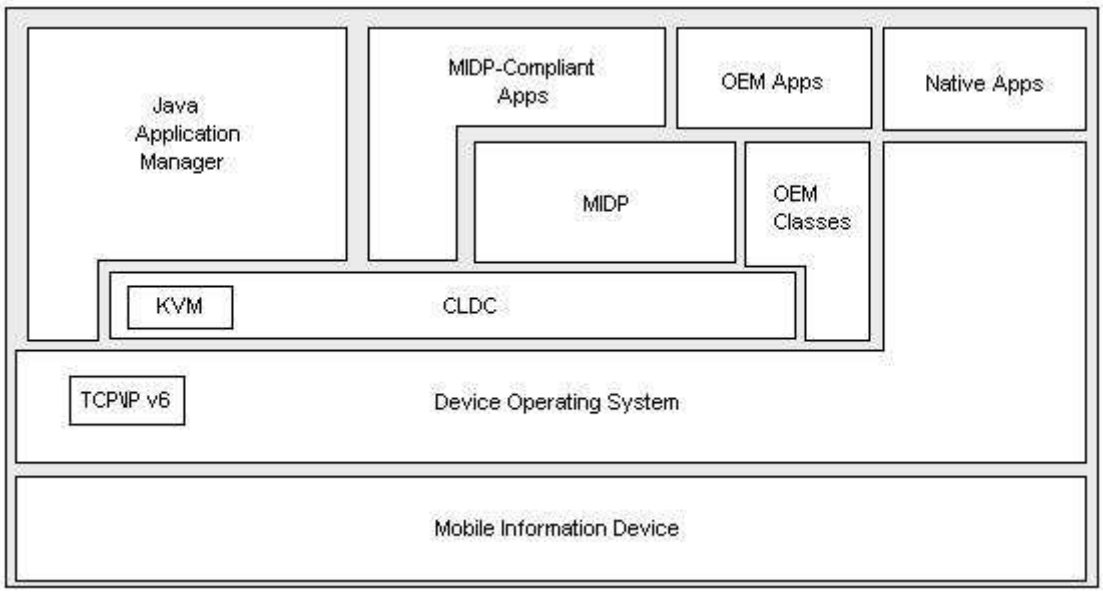

Figura 3.2: Arquitetura dos dispositivos rodando J2ME.

3.3 Configurações

Configurações são compostas de uma máquina virtual e um conjunto mínimo de bibliotecas de classes. Elas provêem a funcionalidade básica para um particular conjunto de dispositivos com uma mesma característica comum, como conectividade a uma rede e quantidade de memória. Um estudo

A configuração CLDC será discutida mais adiante devido a seu valor para o desenvolvimento de aplicativos para aparelhos celulares.

O CDC é feito para dispositivos com mais memória, processadores mais velozes e maiores quantidades de banda do que os dispositivos que utilizam o CLDC. Alguns exemplos seriam boxes de TV, *gateways* residenciais, sistemas automotivos e modernos PDAs. O CDC inclui uma máquina virtual Java completa, e um subconjunto do J2SE muito maior do que o CLDC. Como resultado, a maioria dos dispositivos utilizando o CDC possui processadores de 32 bits e uma quantidade mínima de memória de 2 MB disponível para a plataforma Java e aplicativos associados.

O CLDC é a menor das duas plataformas e inclui dispositivos como telefones móveis com uma conexão a rede intermitente, processador lento (16 ou 32 bits) e memória limitada (128 – 512 kB). O ambiente de desenvolvimento para o J2ME em dispositivos CLDC é o *Mobile Information Device Profile* (MIDP). Ele define as classes para a interface com o usuário, armazenamento persistente, conexão a rede e gerenciamento dos aplicativos. Combinando o CLDC e seu *Kilo Virtual Machine* (KVM), esse perfil fornece um ambiente *runtime*[5] Java completo para dispositivos móveis com memória e poder de processamento mínimos. O MIDP permite que programas sejam baixados, pelo ar, para o dispositivo pelo provedor de serviços em forma de MIDlets.

Perfis são mais específicos que configurações. Um perfil é baseado em uma configuração e adiciona APIs para a interface do usuário, armazenamento persistente, e qualquer outra coisa que seja necessário para o desenvolvimento de aplicativos executáveis onde o MIDP é o perfil utilizado pelo CLDC.

3.3.1 Pacotes opcionais

A plataforma J2ME pode ser estendida a partir da combinação de pacotes opcionais com o CLDC, CDC ou respectiva configuração. Criados para endereçar requerimentos de mercado mais específicos, pacotes opcionais oferecem APIs padrão para se utilizar tecnologias recentes como *Bluetooth*, *Web Services*, W*ireless Messaging*, multimídia e conexão a banco de dados. Devido a estes pacotes opcionais serem modulares, os fabricantes de aparelhos móveis podem incluir esses pacotes para caracterizar cada dispositivo sendo criado.

3.3.2 CLDC

As bibliotecas das APIs CLDC podem ser divididas na seguinte categoria:

[5] Esse termo será muito utilizado para definir um ambiente sendo executado em tempo real.

- **Classes que são um subconjunto das APIs J2SE**: Essas classes são localizadas nos pacotes `java.lang`, `java.io` e `java.util` e são derivadas das APIs J2SE.
- **Classes específicas ao CLDC**: Essas classes estão localizadas no pacote `javax.microedition` e em seus subpacotes.
- **Classes herdadas**: O CLDC herda um número de classes do sistema, de entrada/saída e de utilidades da plataforma J2SE.

Nota: Cada classe que possui o mesmo nome que o nome dado para o pacote da classe J2SE deve ser idêntica ou um subconjunto da classe J2SE correspondente. A semântica das classes e seus métodos não podem ser mudadas e essas classes não podem ter adicionadas a si qualquer método `public` ou `protected`, ou campos que não estejam disponíveis nas bibliotecas da classe J2SE correspondente. A tabela abaixo lista as classes e interfaces não-herdadas que não são provocadas por uma exceção na plataforma J2SE.

Tabela 3.1: Classes e interfaces do J2ME que são herdadas.

Package	Classes
`java.lang`	`Boolean, Byte, Character, Class, Integer, Long, Math, Object, Winnable, Runtime, Short, String, StringBuffer, System, Thread, Throwable`
`java.io`	`ByteArrayInputStream, ByteArrayOutputStream, DataInput, DataOutput, DataInputStream, DataOutputStream, InputStream, OutputStream, InputStreamReader, OutputStreamWriter, PrintStream, Reader, Writer`
`java.util`	`Calendar, Date, Enumeration, Hashtable, Random, Stack, TimeZone, Vector`

As classes de erro e exceção estão listadas abaixo:

Tabela 3.2: Classes de erro e exceção do J2ME.

Package	Classes
`java.lang`	`ArithmeticException, ArrayIndexOutOfBoundException, ArrayStoreException, ClassCastException, ClassNotFoundException, Error, Exception, IllegalAccessException, IllegalArgumentException, IllegalMonitorSateException, IllegalThreadstateException, IndexOutOfBoundException, InstantiationException, InterruptedException, OutOfMemoryError, NegativeArraySizeException, NumberFormatException, NullPointerException, RuntimeException, SecurityException, StringIndexOutOfBoundException, VirtualMachineException`
`java.io`	`EOFException, IOException, InterruptedException, UnsupportedEncodingException, UTFDataFormatException`
`java.util`	`EmptyStackException, NotSuchElementException`

As classes básicas que compõem o CLDC podem ser vistas na Figura 3.3.

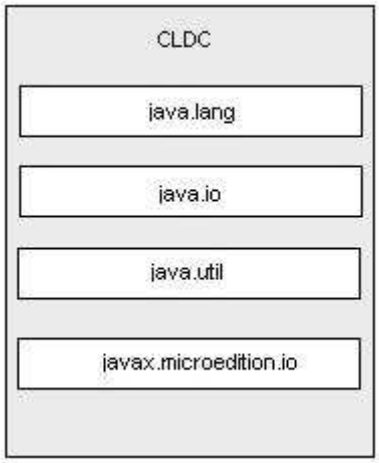

Figura 3.3: Classes do CLDC

3.3.2.1 Funcionalidades eliminadas no CLDC

Quando se criou o CLDC, um número de funcionalidades do J2SE VM foi eliminado onde o J2SE VM é a base do qual o CLDC foi derivado. Essas funcionalidades foram eliminadas porque elas ou eram muito caras para serem implementadas ou causariam um problema de segurança com sua presença. Portanto, a máquina virtual Java (KVM) suportando o CLDC possui as seguintes limitações:

- **Nenhum suporte a números reais**: Operações com números reais requerem que o hardware do dispositivo suporte essas computações. Como os dispositivos utilizados não possuem processadores potentes o suficiente para fazer essa computação, a configuração CLDC não tem suporte a números reais. Portanto, nenhum aplicativo baseado em CLDC pode usar números reais como o `float` ou `double`.
- **Nenhuma finalização**: A finalização de objetos não está disponível no CLDC (e, por extensão, MIDP). Finalização é o mecanismo pelo qual objetos podem limpar eles mesmos antes de serem coletados pelo lixo. Em J2SE, um método `finalize()` da classe Objeto é chamada antes que o objeto seja chamado pelo coletor de lixos. E esse mecanismo não existe para o CLDC. Se o desenvolvedor precisa limpar recursos, o desenvolvedor precisa fazê-lo explicitamente ao invés de utilizar o código de limpeza do método `finalize()`. Essa é uma boa idéia, em especial em pequenos dispositivos com recursos escassos. Explicitamente, limpando recursos significa que a memória e a quantidade de processamento que eles consomem podem ser reclamados o quanto antes. Os códigos de limpeza dos métodos `finalize()` não são executados até que o coletor de lixo seja executado, e nunca se sabe exatamente o que está acontecendo.
- **Correção de erros limitados**: Erros do tipo *Runtime* são lidados em uma maneira específica a implementação. O CLDC define somente três classes para erros: `java.lang.Error`, `java.lang.OutOfMemoryError`, e `java.lang.VirtualMachineError`. Todos erros fora do *runtime* são lidados de uma maneira dependente do dispositivo que pode envolver terminar o aplicativo ou reinicializar o dispositivo.

- **Sem suporte a Java Native Interface (JNI)**: Uma máquina virtual Java suportando o CLDC não implemente o *Java Native Interface* (JNI), primeiramente por razões de segurança. Também, a implementação do JNI também é considerada cara dado as restrições de memória dos dispositivos.

- **Nenhum *Classloading*[6] para o usuário**: Uma das forças da plataforma Java é a habilidade de carregar classes no *runtime*. Infelizmente, por que os recursos são restritos, CLDC\MIDP não permite que o usuário defina seu próprio *classloader*. O gerenciador de aplicativos que roda os MIDlets possui um *classloader*, mas o usuário não pode acessá-lo ou utilizá-lo de qualquer maneira.

- **Sem suporte a classe *Reflection***: O CLDC não suporta a API de Reflexão. Os dispositivos alvo da dupla CLDC\MIDP são simplesmente muito pequenos para permitirem tal fato. Sem reflexão, não é possível implementar o *Remote Method Invocation* (RMI) que permite que um objeto Java seja transferido para um aplicativo Java em outro dispositivo utilizando uma rede de comunicação de dados.
- **Sem suporte a grupos de *threads* ou *threads daemon***: Enquanto uma máquina virtual Java suportando o CLDC implementa *multithreading*, ele não suporta grupos de *thread* ou *thread daemon*. Quando são desejadas operações de *thread* para grupos, então se deve usar um objeto do tipo *Collections* para guardar os objetos *thread* no nível do aplicativo.
- **Sem referências fracas**: O pacote `java.lang.ref` do J2ME permite que aplicativos interajam com o coletor de lixo e permite mais flexibilidade ao se pegar a informação de um objeto. Entretanto, em dispositivos móveis com recursos limitados, esse tipo de flexibilidade não é um grande valor, portanto foi removido do CLDC. Ele também aumenta os requerimentos de memória do dispositivo.

3.3.2.2 Funcionalidades adotadas no CLDC

O CLDC também adota algumas novidades do J2SE com algumas modificações.

- **Verificação de classes**: Na J2SE VM, o verificador de classe é responsável por rejeitar arquivos de classe inválidos onde isso garante segurança de baixo nível para Máquina Virtual. Uma JVM suportando o CLDC deve garantir que também consiga rejeitar arquivos de classe inválidos. Dessa maneira, é garantido que as classes não terão referências para memória fora da pilha. Entretanto, o processo de verificação de classe é caro e consome tempo, logo, não é ideal para dispositivos pequenos com restrição de recursos.

  Os desenvolvedores do KVM decidiram mover grande parte do trabalho de verificação para fora do dispositivo, ou seja, para os *desktops* onde os arquivos de classe são compilados ou para o servidor de onde os aplicativos são baixados. Verificação de classe fora do dispositivo é chamado de pré-verificação. O dispositivo é simplesmente responsável por

[6] *Classloading* é a ação que ocorre quando o ambiente Java executa uma tarefa devido a uma nova classe ser carregada em tempo real – *runtime* – para ser utilizada.

rodar alguns testes no arquivo de classe pré-verificado para garantir que ele foi verificado e ainda é válido.

- **Formato das classes**: Uma implementação CLDC deve poder ler arquivos de classe Java padrão com as mudanças de pré-verificação mencionadas acima. Além disso, ele deve suportar arquivos comprimidos do formato *Java Archive* (JAR). A largura de banda de rede é muito importante para redes móveis com largura de banda escassa e o formato compresso JAR fornece 30 a 50 vezes maior compressão sobre arquivos do tipo classe sem qualquer perda de informação.
- **APIs**: As APIs J2SE requerem muitos megabytes de memória, portanto não são adequados para dispositivos pequenos com recursos limitados. Ao desenvolver APIs para o CLDC, o alvo é prover um conjunto mínimo de bibliotecas úteis para o desenvolvimento de aplicativos e definição de perfis para um conjunto de dispositivos pequenos.
- **Suporte a classe *Property***: No CLDC, não há implementação da classe `java.util.Properties`. Entretanto, o acesso as propriedades é suportado ao chamar o método `System.getProperty(String key)`. As propriedades padrão podem ser vistas na tabela abaixo.

Tabela 3.3: Propriedades padrão suportadas pelo J2ME.

Classe	Explicação	Valor
`microedition.platform`	A plataforma utilizada	Default: null
`microedition.encoding`	A codificação de caracteres utilizada	Default: ISO8859_1
`microedition.configurations`	Versão e configuração do J2ME	Default: CLDC-1.0
`microedition.profiles`	Nome dos perfis suportados	Default: null

- **Internacionalização**: O CLDC suporta a tradução de/para uma seqüência de bytes de uma maneira limitada. Ele especifica as classes `InputStreamReader` e `OutputStreamWriter` idênticas aos `Reader` e `Writer` do J2SE. Se qualquer codificação não é suportada pela implementação, então uma `UnSupportedEncodingException` é lançada. Funcionalidades de localização não são especificadas pelo CLDC. Então, não há suporte embutido para conversão de moedas, datas ou horários.
- **Ordem de procura do arquivo de classe**: A especificação da Java VM e linguagem Java não especificam a ordem no qual os arquivos classe são procurados quando novas classes são carregadas no VM. J2SE usa a variável de ambiente *classpath* para definir a ordem de procura. O CLDC não suporta o conceito de *classpath*, deixando esse problema para a implementação da VM do dispositivo com duas restrições:
  - O programador do aplicativo não pode sobrescrever as classes do sistema (CLDC e Perfis). Classes do sistema são implementadas para um dispositivo específico e mudar elas levaria a um comportamento imprevisível. Também, mudando essas classes de sistema seria um compromisso a segurança do dispositivo.
  - O programador do aplicativo não pode sobrescrever a ordem procura. Sobrescrever a ordem de procura pode provocar um comportamento imprevisível.

3.3.2.3 Funcionalidades e classes adicionadas ao CLDC

A CLDC adiciona funcionalidades e classes para suprir as necessidades para todos seus dispositivos:

- **Classes específicas MIDP**: Em adição as classes específicas MIDP nos pacotes `javax.microedition.rms`, `javax.microedition.midlet`, e `javax.microedition.lcdui`, as seguintes classes, interfaces e classes de exceção estão disponíveis:
  - A classe `IllegalStateException` no pacote `java.lang`.
  - As classes `Timer` e `TimeTask` no pacote `java.util`.
  - A interface `HTTPConnection` para acesso com o protocolo HTTP usando o pacote `javax.microedition.io`.
- **Timers**: Essas classes são usadas pelos aplicativos MIDP para receber e criar notificações de tempo. As seguintes duas classes são disponibilizadas para atrasar ou agendar atividades para execução uma ou repetidas vezes em intervalos regulares:
  - `java.util.Timer`
  - `java.util.TimerTask`
- **Generic Connection *Framework***: Os dispositivos alvo do CLDC como telefones celulares não provêem todo suporte para rede e entrada/saída dos pacotes `java.io` e `java.net`. Como é impossível prover todas funcionalidades definidas pelos pacotes `java.io` e `java.net` do J2SE, CLDC define o *Generic Connection Framework* para suportar a entrada/saída e toda conexão a rede em geral. A idéia é prover um conjunto de abstrações no nível da API, ao invés de prover abstrações para diferentes dispositivos específicos. Para expandir ainda mais essa idéia, ele provê uma API genérica para fazer a conexão.

Essas interfaces, classes e classes de exceção foram introduzidas pelo CLDC e são parte do pacote `javax.microedition.io` que podem ser mapeadas para classes do J2SE.

Tabela 3.4: Classes utilizadas pelo GCF.

Package	Classes
`java.microedition.io`	`Connection, ConnectionNotFoundException, Connector, ContentConnector, Datagram, DatagramConnection, InputConnection, OutputConnection, StreamConnection, StreamConnectionNotifier`

Um estudo mais detalhado do CLDC pode ser encontrado na referência [31].

3.4 MIDP

Aplicativos MIDP fornecem a fundação para aplicativos com uma interface gráfica avançada e intuitiva. A interface gráfica do usuário é otimizada para um display de tamanho pequeno, métodos de entrada específicos e outras funcionalidades nativas de dispositivos móveis. O MIDP fornece uma navegação intuitiva e entrada de dados a partir de um total mapeamento das teclas do telefone. Os aplicativos MIDP são instalados e executados localmente, podendo operar em modo desconectado ou através

de uma rede e têm a habilidade para armazenar e gerenciar seguramente toda informação local. A Figura 3.4 fornece as classes implementadas no MIDP. A especificação do MIDP pode ser visto nas referências [28] e [30], sendo que as duas foram utilizadas como base para essa seção.

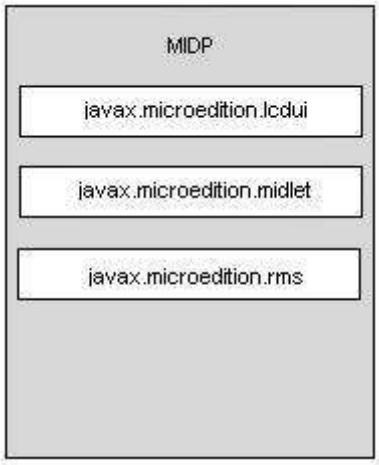

Figura 3.4: Classes do MIDP.

3.4.1 Requisitos para os *Mobile Information Devices* (MIDs)

Dispositivos para o qual este perfil foi projeto podem ser definidos a seguir:

1. Um *display* com, pelo menos, 96 por 54 pixels com uma profundidade de 1-bit pelo menos;
2. Uma interface para entrada de dados com uma mão, teclado de duas mãos ou *touch screen*;
3. A memória é feita de 32 kB de espaço volátil para a pilha *runtime* e 128 kB de espaço não-volátil para os componentes MIDP e 8 kB de memória não-volátil para as informações persistentes criadas pelos aplicativos;
4. Uma conexão a rede intermitente *full-duplex* deve estar presente.

3.4.2 Interface com o usuário móvel

O MIDP oferece uma API de interface gráfica de alto nível que retira dos desenvolvedores toda a complexidade de construir aplicativos portáteis. Essa API de alto nível permite que os desenvolvedores construam aplicativos fáceis de usar, altamente gráficos e portáteis, otimizados para dispositivos móveis, além de reduzir a duração de desenvolvimento do aplicativo.

As funcionalidades desta interface gráfica incluem telas pré-definidas para mostrar e selecionar listas, editar texto, mostrar diálogos de alerta e adicionar barras de rolagem onde *Forms*[7] são telas que podem incluir qualquer número de itens pré-definidos – imagens, caixas de texto somente para leitura, caixas de texto editáveis, caixas de tempo, *charts* e grupos de escolha – assim como quaisquer itens customizados adicionados pelo desenvolvedor para prover funcionalidades e gráficos únicos. Todas

[7] Classe Java utilizada pelo MIDP para gerenciar os objetos gráficos no *display*.

telas e itens estão de acordo com os recursos do dispositivo, ou seja, com suporte nativo para o tamanho do *display*, entrada e capacidades de navegação. Isso permite que os desenvolvedores definam uma interface altamente portátil e flexível que pode mudar seu *layout* e navegação para utilizar todos recursos de cada dispositivo.

3.4.3 Funcionalidades para jogos e multimídia

O MIDP é ideal para construir jogos e aplicativos multimídia portáteis. Uma API de interface de baixo nível complementa a API de alto nível para fornecer aos desenvolvedores maior controle sobre os gráficos e entrada que eles precisam. Uma API para jogos adiciona funcionalidades específicas para jogos como camadas de *sprites* que possuem vantagens em relação a capacidade gráfica nativa do dispositivo. Uma API de áudio embutida fornece suporte para tons, seqüências de tons e arquivos WAV. Em adição, desenvolvedores podem usar a *Mobile Media API* (MMAPI), um pacote adicional do MIDP, para adicionar *video streaming* e outros conteúdos multimídia aos seus aplicativos.

3.4.4 Conectividade extensiva

MIDP permite que os desenvolvedores utilizem toda capacidade de troca de mensagens de um dispositivo móvel. Este perfil suporta padrões líderes de transmissão de dados, incluindo HTTP, HTTPS, datagramas, *sockets*, *Server sockets* e comunicação serial. MIDP também suporta *Short Message Service* (SMS) e *Cell Broadcast Service* (CBS) de redes GSM ou CDMA utilizando a *Wireless Messaging* API (WMA), pacote opcional do MIDP.

A conectividade MIDP e o suporte para troca de mensagens permitem aplicativos totalmente conectados e orientados para eventos. MIDP suporta um modelo de servidor do tipo *push* onde um registrador *push* mantém uma lista dos aplicativos registrados para receber informação da rede. Quando a informação chega, o dispositivo decide se deve começar um aplicativo baseado nas preferências do usuário. Essa arquitetura *push* permite que os desenvolvedores possam incluir em seus aplicativos alertas, troca de mensagens e *broadcasts*, e usar ao máximo as capacidades baseadas em evento dos dispositivos e operadoras.

3.4.5 Provisionamento Over-the-Air

Um grande benefício do MIDP é sua habilidade de instalar e atualizar aplicativos dinamicamente *over-the-air* (OTA). A especificação MIDP define como aplicativos MIDP são descobertos, instalados, atualizados e removidos de dispositivos móveis e o MIDP também permite que um provedor de serviço identifique qual aplicativo MIDP vai funcionar em qual dispositivo e obter uma indicação de status do dispositivo após uma instalação, atualização ou remoção. O modelo de provisionamento MIDP OTA foi definido e adotado por fabricantes líderes de dispositivos para permitir uma solução de provisionamento segura e confiável. Uma explicação de como esse provisionamento funciona será dada adiante neste capítulo.

3.4.6 Segurança fim-a-fim

MIDP provê um modelo de segurança robusto – construído através de padrões abertos – que protege a rede, os aplicativos e os dispositivos móveis. O uso de HTTPS maximiza padrões tais como o SSL e o WTLS para permitir a transmissão de informação cifrada. Domínios seguros protegem contra acesso não-autorizada a informação, aplicativos ou outros recursos da rede ou dispositivo por aplicativos MIDP no dispositivo. Por padrão, aplicativos MIDP não são confiáveis e são designados a domínios não-confiáveis que previnem o acesso a qualquer funcionalidade privilegiada. Para ganhar acesso privilegiado, o aplicativo MIDP deve ser designado para um domínio específico que é definido no dispositivo móvel e são assinados utilizando o padrão de segurança x.509 PKI. Para que um aplicativo MIDP assinado possa ser baixado, instalado e permitido suas devidas permissões, ele deve ser autenticado com sucesso.

3.4.7 Pacotes

A Tabela 3.5 apresenta todos pacotes implementados para o perfil MIDP.

Tabela 3.5: Pacotes do MIDP.

Pacotes de Interface do Usuário	
`javax.microedition.lcdui`	A API UI define as funcionalidades para implementação de uma interface gráfica para aplicativos MIDP.
`javax.microedition.lcdui.game`	O pacote da API de jogos fornece um conjunto de classes que permite o desenvolvimento de um conteúdo rico para jogos em ambiente *wireless*.
Pacotes de Tempo de vida dos Aplicativos	
`javax.microedition.midlet`	O pacote MIDlet define as interações entre os aplicativos e o ambiente no qual eles são executados.
Pacotes de persistência	
`javax.microedition.rms`	O MIDP fornece um mecanismo para que os MIDlets armazenem e busquem informações persistentemente.
Pacotes de conexão	
`javax.microedition.io`	O perfil MID inclui suporte a *networking* baseado no *Generic Connection Framework* do CLDC.
Pacotes de chave pública	
`javax.microedition.pki`	Certificados são utilizados para autenticar informação de conexões seguras.
Pacotes de Som e Tons	
`javax.microedition.media`	A Media API do MIDP 2.0 é um bloco compatível com a MMAPI (JSR 135).
`javax.microedition.media.control`	Esse pacote define os tipos de Controle específicos que podem ser usados pelo *Player*.
Pacotes centrais	

`java.lang`	Classes MID da linguagem incluídas a partir do J2SE.
`java.util`	Classes MID de utilitários incluídas a partir do J2SE.

3.5 O ciclo de vida de MIDlets

Aplicativos MIDP são representados por instâncias da classe `javax.microedition.midlet.MIDlet` onde MIDlets tem um ciclo de vida específico que é refletido nos métodos e comportamentos da classe MIDlet.

Um pedaço de software fora da linguagem Java, o gerenciador de aplicativos, controla a instalação e execução dos MIDlets. Um MIDlet é instalado ao mover seus arquivos de classe para um dispositivo. Os arquivos de classe são empacotados em arquivos do tipo *Java Archive* (JAR), enquanto um arquivo descritor acompanhante (com uma extensão .jad) descreve o conteúdo do JAR.

3.5.1 Gerenciamento do ciclo de vida dos aplicativos

O perfil MIDP provê o pacote `javax.microedition.midlet` que contém o *framework* para o desenvolvimento de MIDlets. Ele fornece as classes e métodos para começar, terminar, pausar e destruir aplicativos no ambiente Java.

Um MIDlet vai pelos seguintes estados:

1. Quando o MIDlet está quase pronto para ser executado, uma instância é criada. O construtor do MIDlet é executado e o MIDlet está no estado *Paused*.
2. A seguir, o MIDlet entra no estado *Active* depois que o gerenciador do aplicativo chama `startApp()`.
3. Enquanto que o MIDlet está em *Active*, o gerenciador de aplicativos pode suspender sua execução ao chamar `pauseApp()` onde isso coloca o MIDlet de volta no estado *Paused*. Um MIDlet pode colocar si mesmo no estado *Paused* ao chamar `notifyPaused()`.
4. O gerenciador de aplicativos pode terminar a execução do MIDlet ao chamar `destroyApp()` no qual o MIDlet entra no estado *Destroyed* e pacientemente espera pelo coletor de lixo. Um MIDlet pode destruir a si ao chamar `notifyDestroyed()`.

A interação entre os diferentes estados pelo qual um MIDlet passa e a troca de mensagens feitas pode ser vista na Figura 3.5.

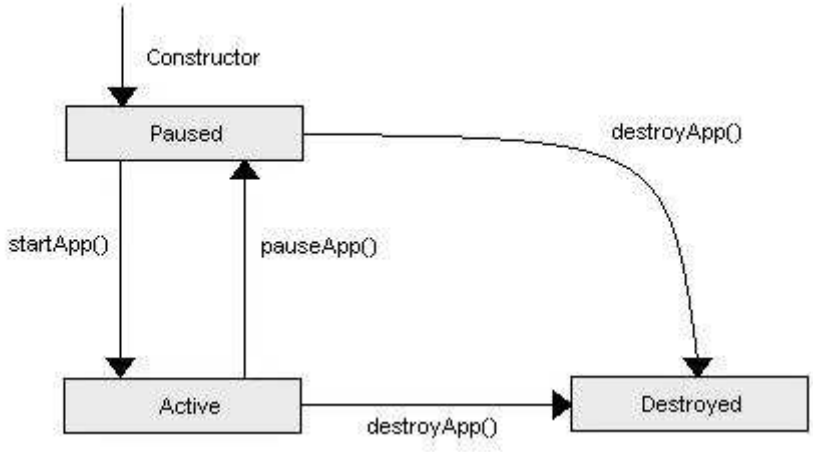

Figura 3.5: Ciclo de vida dos MIDlets.

3.6 Programação para redes

Java é a linguagem de programação preferida para construir aplicativos que se liguem a alguma rede. É bastante conhecido que Java fornece funcionalidades poderosas para construir aplicativos ligados a rede para *desktops* (aplicativos Java), para *enterprise* (usando *Servlet*s, JSPs e EJBs) e para a Internet (usando *Servlet*s, JSPs e *Applets*).

Em geral, os requisitos para as bibliotecas de *networking* e armazenamento variam significantemente de um dispositivo para outro. Por exemplo:

- Dispositivos baseados em uma rede celular comutada por circuito funcionam melhor em conexões baseadas em rajadas (TCP\IP);
- Dispositivos baseados em redes de dados comutadas por pacotes funcionam melhor em mecanismos de comunicação baseados em datagramas;
- Alguns dispositivos possuem sistemas de arquivos tradicionais, enquanto que muitos outros possuem mecanismos específicos para o dispositivo;
- Alguns dispositivos possuem suporte para tecnologias como 802.11 ou Bluetooth.

Devido as limitações de memória, dispositivos suportando certos tipos de entrada/saída, conexão com determinados tipos de rede, e capacidades de armazenamento não suportam mecanismo múltiplos, em geral. Além das classes de *networking* do J2SE não possuírem suporte a uma variedade de protocolos de redes *wireless*. Então, ao invés, de reutilizar as APIs da `java.net` ou `java.io`, o CLDC define seu próprio conjunto de APIs que podem ser facilmente mapeadas na implementação J2SE.

Como programador do aplicativo, deve-se querer escrever MIDlets que sejam capazes de rodar pela rede e que sejam suportados por todos dispositivos com MIDP. Também não deve ser necessário lidar com os detalhes de baixo nível das redes e protocolos. Em vista a separar o programador dos detalhes de baixo nível, CLDC fornece uma abstração limpa de diferentes tipos de conexão. Essa camada de abstração é

conhecida como *Generic Connection Framework* (GCF). MIDP estende essa abstração e fornece a implementação de algumas interfaces provenientes desse *framework*.

3.6.1 Generic Connection Framework

A *Generic Connection Framework* define uma hierarquia de setes interfaces que agrupam juntas classes de protocolos com a mesma semântica. Ao nível de implementação, define o mínimo que uma classe precisa implementar para cada protocolo suportado.

Essa coleção de interfaces forma uma hierarquia que se torna progressivamente mais avançada quando a hierarquia progride a partir da interface raiz.

- A interface `Connection` é o tipo de conexão mais básico. Ela só pode ser aberta e fechada.
- A interface `InputConnection` representa o dispositivo do qual informação pode ser lida.
- A interface `OutputConnection` represente o dispositivo para o qual informação pode ser escrita.
- A interface `StreamConnection` combina as conexões de entrada e saída.
- A interface `ContentConnection` é uma sub-interface da `StreamConnection`. Ela provê acesso a algumas informações básicas de *metadata* providenciadas pelas conexões HTTP, como os cabeçalhos HTTP, tipos de conteúdo, e tamanho do conteúdo.
- A interface `StreamConnectionNotifier` espera que uma conexão seja estabelecida. Ela retorna uma `StreamConnection` no qual um *link* de comunicação foi estabelecido.
- A interface `DatagramConnection` representa um *endpoint* de datagrama.
- A interface `HTTPConnection` representa um *endpoint* HTTP.

Esse arranjo permite que os desenvolvedores escolham o nível ótimo de portabilidade para o protocolo para o aplicativo que estão escrevendo. Essas interfaces são definidas na configuração CLDC e sua implementação é providenciada pelo perfil. Entretanto, perfis não são obrigados a providenciar implementação para todas as interfaces. Os perfis podem escolher providenciar implementação de algumas delas ou podem até estender essas interfaces quando providenciarem a implementação. O perfil MIDP, por exemplo, provê uma interface `HTTPConnection` e sua implementação, mas não provê implementação para a interface `DatagramConnection`.

Outra classe importante que o GCF provê é a classe `Connector`. Nenhuma das interfaces acima provê um método aberto para se estabelecer uma conexão. A classe `Connector` provê métodos estáticos que permitem que um aplicativo estabeleça uma conexão. Por exemplo, para abrir uma conexão, a sintaxe se parece com:

```
Connector.open("<protocol>:<address>;<parameters>");
```

O parâmetro que é passado é uma String que armazena o *Uniform Resource Indicator* (URI). Como pode ser visto, o formato do URI é bem genérico em representar qualquer tipo de conexão.

A vantagem principal dessa sintaxe é que somente a string parâmetro determina o protocolo que será utilizado pela conexão. Grande parte do código do aplicativo se mantém o mesmo, não importando o tipo de protocolo utilizado.

A implementação do protocolo é providenciada pelo perfil. Quando o método aberto é chamado, a implementação do protocolo é determinada pelo identificador do protocolo na string (como HTTP: ou file:) e carregado no *runtime*. Isso é análogo a maneira como drivers de dispositivos são usados por aplicativos em *desktops*.

3.6.2 A interface HTTPConnection

O perfil MIDP providencia uma interface chamada `HTTPConnection` que estende a interface `ContentConnection` definida pelo CLDC como:

```
Public interface HTTPConnection
       extends javax.microedition.io.ContentConnection
```

Além disso, o MIDP providencia somente essa interface para toda programação para a rede. Ele não fornece suporte para outros tipos de conexão como datagramas ou *sockets* onde isso significa que todos fabricantes que suportem MIDP precisam providenciar uma implementação para essa interface. Alguns fabricantes podem escolher em prover suporte para outros tipos de conexão como datagramas. Entretanto, programar para essas conexões faria seu aplicativo não-portável. Em outras palavras, programar para a `HTTPConnection` (e suas super interfaces) torna seus MIDlets portáteis entre todos dispositivos que suportem MIDP.

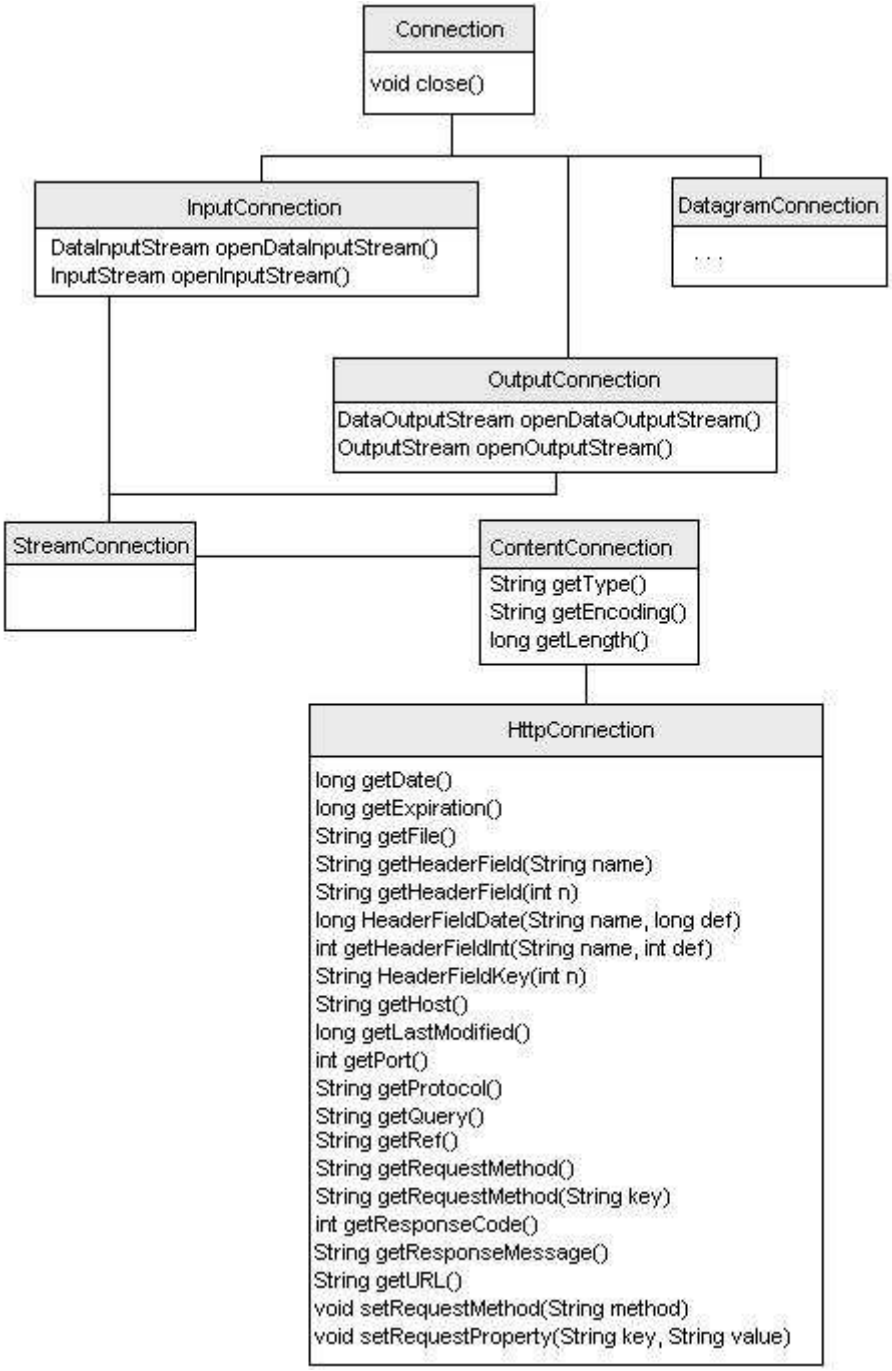

Figura 3.6: Diagrama de classes para HTTPConnection.

3.7 XML no MIDP

O perfil MIDP junto com a configuração CLDC formam uma plataforma J2ME completa que provê capacidade para conexão com a rede (*networking*) e armazenamento. Assim, os aplicativos usando o perfil MIDP podem interagir com outros aplicativos ligados a rede em algum servidor ou outro dispositivo. Essa interação, tipicamente, envolve a troca e transferência de informação. Quando os aplicativos dos dois lados da interação são controlados pelo desenvolvedor do MIDlet, então é fácil definir o formato da informação para algum formato que seja compatível. Entretanto, na maioria dos casos, o formato das informações usadas pelos componentes do servidor

foram previamente definidos. Se os componentes do lado do servidor usam XML como uma linguagem para troca de informação, o desenvolvedor do MIDlet precisa de um meio para interagir com o componente do servidor usando XML. Assim, o MIDlet precisa processar a informação XML diretamente ou converter o documento XML no servidor para um formato pelo qual o MIDlet entende. A segunda opção não é sempre possível quando o componente do servidor não é facilmente modificado. Portanto, quando se quer usar a primeira opção, é necessário se processar documentos XML nos aplicativos MIDP. Um estudo mais detalhado da utilização de XML em J2ME pode ser visto na referência [2].

Ser possível de traduzir documentos XML nos MIDlets permite uma interação com muitos aplicativos que estão fora do limite para MIDlets que não suportem XML. Tais aplicativos poderiam incluir:

- **Integração de aplicativos/Web Services**: Ser possível construir e entender mensagens SOAP (que são documentos XML) permite que os MIDlets participem do mundo dos *Web Services*.
- **Sincronização**: sincronização de informação de *desktops* para dispositivos móveis usando um formato comum de troca de informações como *SyncML*.
- **Armazenamento de informação**: XML se tornou popular como um formato nativo de armazenamento de informação.

3.7.1 Baseado em árvore x Baseado em evento

Todos tradutores são baseados em tradução baseada em árvores ou tradução baseada em eventos. O primeiro tipo de tradutor, o tradutor baseado em árvores, escreve todo o documento na memória e constrói uma representação em forma de árvore do conteúdo deste documento. Isso permite que o aplicativo ande pela arvora armazenada na memória e tome ações baseadas nos elementos encontrados. O segundo tipo de tradutor, tradutor baseado em eventos, lê o documento elemento por elemento, chamando métodos de *callback* em seu aplicativo baseado em quais elementos foram encontrados.

Ao menos que se saiba de antecedência que os documentos serão pequenos, geralmente faz mais sentido usar um tradutor baseado em eventos em um ambiente com pouca memória que é a característica típica de dispositivos MIDP. Quando se usa um tradutor baseado em árvore, não existe limite para a quantidade de memória que o documento irá utilizar, podendo levar para instabilidade ou *crash* dos MIDlets.

3.7.2 kXML

O kXML foi o tradutor XML utilizado no cliente elaborado neste projeto, logo será feita uma pequena introdução sobre este tradutor.

O projeto kXML providencia um tradutor XML do tipo *pull* que pode ser utilizado em todas plataformas Java, incluindo o J2ME (CLDC\MIDP\CDC). Devido a seu pequeno rastro (tanto no tamanho do JAR quanto na utilização de memória), ele é especialmente utilizado em *Applets* e aplicativos Java rodando em dispositivos móveis.

O tradutor kXML fornece:

- Suporte a *Namespace* em XML;
- Um modo para tradução de HTML ou outros formatos SGML (pois estes documentos podem não aderir a estrita sintaxe do XML);
- Um ambiente com pequeno *overhead* (pequeno rastro na memória);
- Suporte para escrita (saída) de documentos XML, incluindo suporte para manipulação de *namespaces* durante a escrita.;
- Suporte ao kDOM opcional (provê as APIs DOM para seus MIDlets);
- Suporte WAP opcional (WBXML/WML);

A mais nova versão do kXML é o kXML2 que provê um tradutor baseado em *pull*. Um tradutor baseado em *pull* é uma forma especializada de tradutor baseado em eventos. Quando se utiliza um tradutor baseado em eventos padrão, o tradutor emite eventos ao progredir pelo documento. Seu aplicativo pega esses eventos assim que o tradutor passa pelo documento, fazendo as ações apropriadas baseadas no evento recebido. Nesse cenário, o tradutor empurra eventos para seu aplicativo. Um tradutor do tipo *pull* pede para o tradutor por eventos quando são necessários. O aplicativo puxa os eventos do tradutor, pedindo por cada evento quando necessário.

O tradutor kXML não suporte entidades externas. Além disso, o tradutor não traduz declarações <DOCTYPE> Para compensar isso, o tradutor emite um evento legado quando uma declaração DOCTYPE é encontrada, então o desenvolvedor por pegar esse evento e manipular essa declaração da maneira desejada.

3.8 Desenvolvimento de aplicativos MIDP

Existem algumas classes que todo aplicativo MIDP com interface gráfica geralmente deve utilizar. As classes mostradas são um subconjunto das classes necessárias para formar uma interface para o usuário em um aplicativo MIDP.

- **Classe MIDlet**: Essa classe fornece a classe básica para todos aplicativos MIDP. Como a classe *Applet*, a classe MIDlet provê a semântica para um ciclo de vida simples como começar, pausar, e destruir para os aplicativos MIDP, assim como alguns métodos de *callback* para as classes MIDlet.
- **Classes para interface gráfica de baixo nível**: Essas classes provêem os elementos de interface com usuário mais brutos. Algumas classes são as caixas de texto, strings, alertas, botões de escolha, entre outros. Classes básicas como `Screen` e `Displayable` provêem *containe*rs básicos e funções de *layout*. A classe `Canvas` provê funções gráficas primitivas e eventos de baixo nível.
- **Classes de comando**: Essas classes provêem as classes necessárias para manipular ações associadas com os botões ou itens da interface com o usuário.

3.8.1 Desenvolvimento MIDP e lançamento dos aplicativos

O desenvolvimento de programas Java MIDP não é diferente do desenvolvimento de qualquer outro aplicativo Java. Arquivos Java são criados, compilados e então são empacotados. Existem alguns passos que são diferentes para arquivos MIDP. Eles são:

1. Criar o arquivo Java (.java) usando um editor.
2. Compilar o arquivo `.java` com o comando `javac`. Esse é o mesmo comando `javac` que o JDK J2SE fornece. O comando `javac` produz arquivos `.class`.
3. Preverificar o arquivo `.class`. O arquivo classe produzido pelo `javac` não é utilizável em dispositivos J2ME. O passo da preverificação simplifica o passo de carregamento da classe no dispositivo MIDP. Um novo arquivo `.class` com informações de preverificação é produzido.
4. Empacotar o arquivo `.class` verificado em um arquivo JAR (`.jar`)
5. Lançar o arquivo JAR e o arquivo JAD no dispositivo. O arquivos JAD descreve a maneira como o programa MIDP deve ser instalado e também é obrigatório.

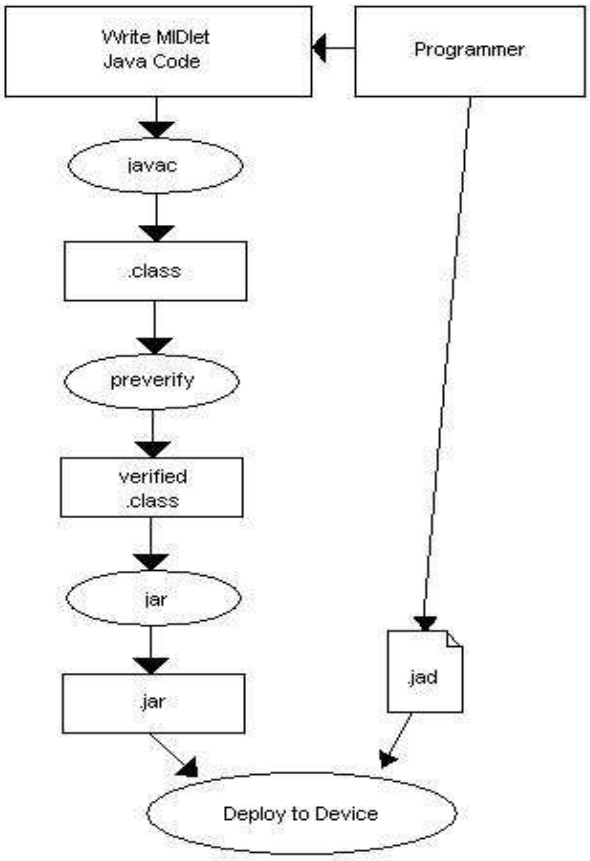

Figura 3.7: Lançamento de um aplicativo MIDP.

SERVLETS

4.1 Introdução aos Servlets

Servlets[8] podem ser definidos como: as classes Java que implementam a interface `javax.Servlet.Servlet` tanto diretamente ou indiretamente, processam pedidos HTTP vindos de um cliente Web, e estendem a classe `HTTPServlet`. Além disso, pode-se afirmar que *Servlet*s que processam outros tipos de pedidos devem estender a classe `GenericServlet`. Porém, para o funcionamento de servidor Web utilizando a tecnologia de *Servlets*, é necessário um certo número de classes de suporte que provêem um ambiente completo para se programar e executar os *Servlet*s.

As funcionalidades oferecidas pela classe `Servlet` são: a inicialização e o controle do ciclo de vida do *Servlet*, processamento e manipulação de pedidos, criação de respostas para os pedidos, manutenção dos clientes Web através de sessões, manutenção da informação contextual, filtragem dos pedidos e respostas, o chamado para outros componentes Web, e a troca de informações com outros componentes Web.

O material utilizado neste capítulo pode ser visto, com maior quantidade de detalhes, na referência [9].

4.2 Inicialização e ciclo de vida dos *Servlet*s

Instâncias de *Servlet*s são criados quando um pedido do cliente é recebido e uma instância do *Servlet* ainda não existe. Quando um *Servlet* é criado, suas classes são carregadas e uma instância do *Servlet* é criada. Em seguida, o método `init()` do *Servlet* é chamado para que seja possível a inicialização dos métodos de serviço como `doGet()` ou `doPost()`.

Quando o *container*[9] decide remover uma instância do *Servlet*, ele chama o método `destroy()` do *Servlet* para executar essa tarefa, então o *Servlet* e toda informação contida no mesmo é perdida.

Pode-se definir algumas classes do tipo `listener` para o *Servlet* onde estas classes fazem a captura de eventos do ciclo de vida do *Servlet* que ocorrem durante a execução do *Servlet*. Eventos do ciclo de vida incluem a criação e destruição de *Servlet*s, a manipulação de atributos, além da criação, invalidação e *timeout* de sessões.

A Figura 4.1 apresenta todas as classes e interfaces que fazem parte da implementação de um *Servlet* Java.

[8] A palavra *Servlet* tenta denominar o aplicativo que roda no Servidor em uma arquitetura Cliente-Servidor.
[9] Objeto que faz parte do servidor Web e responsável por controlar todas instâncias da classe *Servlet*.

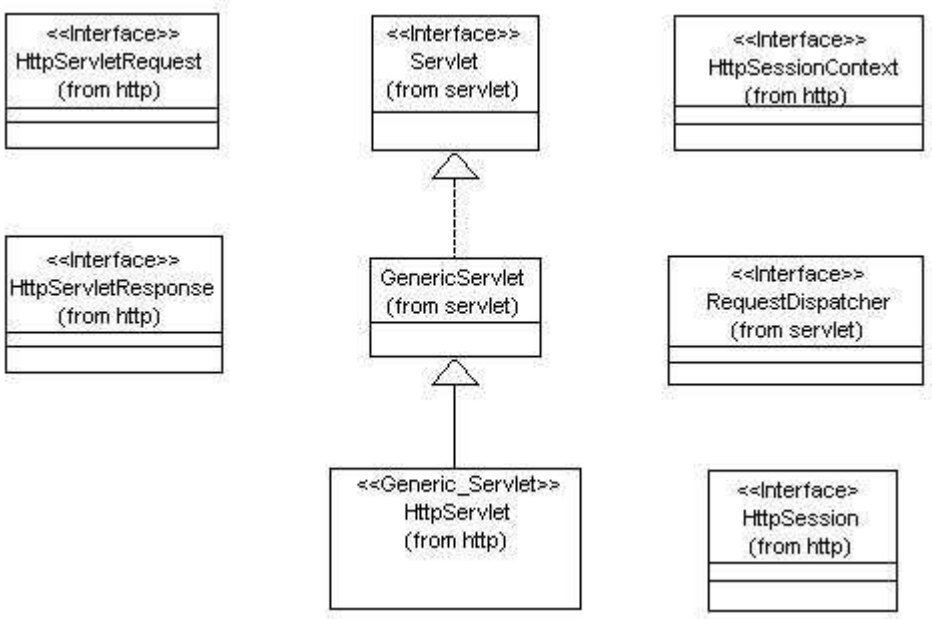

Figura 4.1: Classes e interfaces de um Servlet.

4.3 Escrevendo funções para aplicativos *Servlet*s

A maneira como um *Servlet* funciona pode ser definida nos seguintes passos:

1. O *container* recebe os pedidos e passa estes pedidos para o *Servlet* ao invocar o método de serviço que faz parte do *Servlet*;
2. Para *Servlet*s HTTP, este método é tipicamente os métodos `doGet()` ou doPost(). Para um *Servlet* genérico, o método `service()` é chamado;
3. Os métodos de serviço definem os objetos de pedido e resposta como seu parâmetro e o *container* passa esses objetos para o *Servlet* quando o método é invocado.

A seguir, pode-se ver a assinatura do método de serviço `doGet()` para um *Servlet* HTTP:

```
Public void doGet (HTTPServletRequest request,
                   HTTPServletResponse response)
 throws ServletException, IOException
```

4.4 Estado do Cliente

Os clientes Web várias vezes precisam manter seu estado através de múltiplas chamadas HTTP para o *Servlet*. O protocolo HTTP não possui estados, ou seja, é definido como *stateless*[10], então os *Servlet*s utilizam um objeto `HTTPSession` para manter o estado dos clientes. O objeto `Session` de cada cliente é mantido ao se passar um identificador entre o cliente e o servidor. Isso pode ser feito através de cookies ou através de uma técnica chamada de *URL rewriting* onde um *Servlet* insere o identificador da sessão como uma string de parâmetro no URL quando o cliente não habilitou a utilização de cookies.

[10] Palavra em inglês que denota um objeto que não possui nenhum estado.

Uma nova sessão pode ser criada ou uma sessão já existente encontrada usando o método `getSession()` no objeto que fez o chamado. Assim que uma sessão é criada ou encontrada, os métodos `getAttribute()` e `setAttribute()` podem ser chamados para acessar ou modificar o estado da sessão do cliente.

CONCEITOS DA LINGUAGEM XML

A linguagem XML costuma ser muito utilizada em um contexto onde não se sabe muito sobre os elementos e o conteúdo de um documento, mas se conhece a estrutura do documento. Neste caso, todos elementos precisam começar e terminar, além disso, cada atributo pode possuir somente um único valor.

O conteúdo do documento, assim como os elementos e atributos utilizados estão a critério do desenvolvedor. É possível se desenvolver a formatação, o conteúdo e as especificações para a representação da sua informação, assim permitindo uma interoperabilidade entre diferentes sistemas, pois o meio como as mensagens são trocadas são definidas pelo desenvolvedor do documento XML. O material utilizado para escrever da seção 5.1 à 5.4 foi retirado da referência [1].

5.1 XML 1.0

O arquivo abaixo é um exemplo típico de um documento XML definindo um objeto `Sessao` onde toda `Sessao` possui um `Contexto` e uma `Transicao`, e toda `Transicao` é formada por `IDTransicao`.

```
<?xml version=”1.0”?>
<!DOCTYPE Sessao SYSTEM “DTD/Sessao.dtd”>

<!-- Comentários -->
<Sessao xmlns=”http://www.xxx.com/sessao”
        xmlns:ora=”HTTP://www.xxx.com”
>

<IDSessao ora:series=”1234”>Sessao 1</IDSessao>

<!-- Lista de Contextos -->
<Contexto>
      <Transicao number=”1”>
            <IDTransicao number=”567” />
            <IDTransicao number=”890” />
      </Transicao>
</Contexto>

<Contexto>
      <Transicao number=”2”>
            ...
      </Transicao>
</Contexto>

<ora:copyright>&MarcosDytz;</ora:copyright>
</Sessao>
```

Grande parte da especificação XML 1.0 descreve o que é intuitivo na maioria dos casos. Para qualquer um que já tenho desenvolvido algum arquivo em HTML, ou

SGML, ou esteja familiarizado com estas linguagens, então é fácil pegar os conceitos por trás dos elementos e atributos que formam um documento XML. A grande diferença é que em XML, existem mais definições de como se devem utilizar esses itens, e como um documento deve ser estruturado, portanto a linguagem XML gasta mais tempo definindo certos detalhes como *whitespace* do que introduzindo novos conceitos que não seja de familiaridade para desenvolvedores HTML.

Um documento XML pode ser quebrado em dois pedaços básicos: o cabeçalho onde é selecionado o tradutor[11] XML e outras informações do aplicativo XML como a maneira correta para lidar com os documentos, e seu conteúdo que consiste da informação XML em si. Embora esta seja uma divisão bastante solta, ela ajuda a diferenciar as instruções para os aplicativos dentro de um documento XML do conteúdo, sendo essa é uma distinção bastante importante. O cabeçalho – *header* – é simplesmente a declaração do XML e pode incluir a codificação, e se o documento é um documento só ou se o mesmo requer outro documento para que o mesmo seja referenciado, assim sendo possível uma total compreensão do seu significado. O restante do cabeçalho é feito de itens como as declarações DOCTYPE.

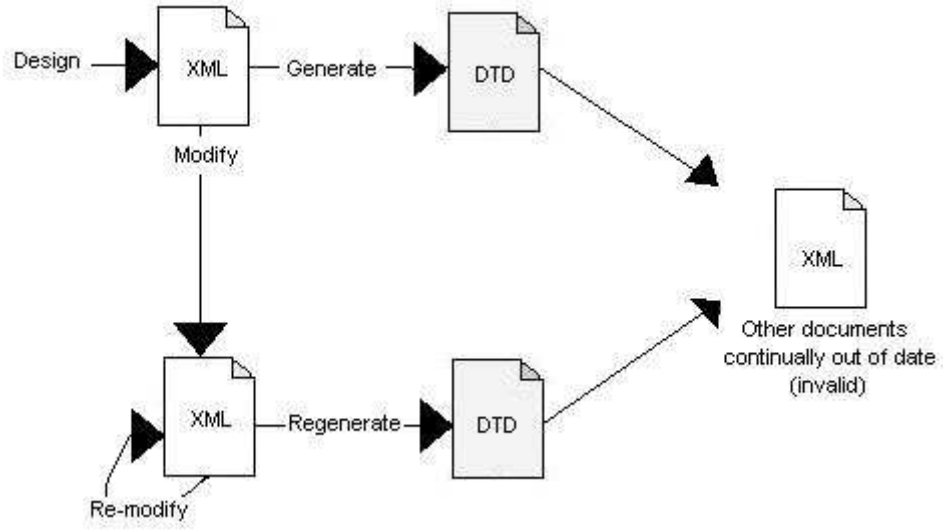

Figura 5.1: Etapas para a geração de um documento XML.

5.1.1 O elemento raiz

O elemento raiz é o elemento de mais alto nível em um documento XML e deve ser a primeira *tag*[12], a abertura, e a última *tag*, o fechamento, dentro de um documento. Esse elemento fornece um ponto de referência que permite que o tradutor XML ou aplicativos com suporte a XML façam o reconhecimento de um início e um fim de um documento XML.

A linguagem XML especifica que somente pode haver um único elemento raiz no documento, ou seja, o elemento raiz deve encobrir todos outros elementos dentro de um documento. Além deste requisito, o elemento raiz não se difere de qualquer outro

[11] Optou-se por traduzir o termo *parser* em tradutor, e não analisador.
[12] O termo *tag* foi mantido em seu formato original na língua inglesa devido a sua grande utilização nessa forma na literatura técnica em língua portuguesa.

elemento XML. É importante entender isso, por que documentos XML podem referenciar e incluir outros elementos XML dentro de si. Nesses casos, o elemento raiz do documento referenciado se torna um elemento encoberto no documento referente, e deve ser lidado normalmente por um tradutor XML. Definir elementos raiz como elementos XML padrão, sem qualquer propriedade ou comportamento especial, permite que a inclusão destes documentos funcione sem problemas.

5.1.2 Elementos

Até o momento, se falou sobre como definir um elemento como pode ser visto no exemplo, mas também é importante se estudar o que seria um elemento, sabendo que os mesmos podem ser representados por nomes arbitrários e estão encobertos entre sinais de maior e menor.

A primeira regra em se criar elementos é que o nome dos mesmos deve começar com uma letra ou *underscore*[13], e então podem conter qualquer número de letras, número, *underscores*, hífens ou períodos. Os nomes não podem conter espaços entre as palavras que os compõem.

Além disso, o nome de elementos XML é sensível a caixa, ou seja, utilizando as mesmas regras que norteiam o nome de variáveis Java resultará em um nome para elementos XML coerente. Portanto manter uma boa documentação a partir de uma boa escolha para o nome dos elementos é essencial para o desenvolvimento de qualquer aplicativo utilizando a linguagem XML para troca de dados.

Outra regra que precisa ser seguida para criação de documentos XML é que todo elemento aberto precisa ser fechado em seguida. Não existe qualquer exceção a esta regra como existe em outras linguagens do tipo *markup*, como HTML. Um elemento de fechamento consiste de uma barra e o nome do elemento: `</Sessao>`. Entre um elemento de abertura e um de fechamento, pode haver qualquer número de elementos ou informações textuais. Entretanto, não se pode misturar a ordem de *tag*s entrelaçadas: o primeiro elemento aberto deve sempre ser o último elemento a ser fechado. Se qualquer uma dessas regras para sintaxe XML não é seguida no documento XML, então o documento não é considerado do tipo bem-formado.

Um documento bem-formado é um no qual todas regras da sintaxe XML são seguidas, e todos elementos e atributos estão posicionados corretamente. Entretanto, um documento bem-formado não é necessariamente válido que significaria que o mesmo segue as restrições impostas ao documento por um DTD ou *Schema*. Existe uma diferença significativa entre um documento bem-formado e outro válido.

A aderência bem estrita de XML para as regras de ordenamento e entrelaçamento permite que a informação seja traduzida e manipulada de uma maneira bem mais veloz comparada com outras linguagens do tipo *markup* que não possuem essas restrições.

[13] Utilizou-se o termo *underscore* devido a grande utilização do mesmo na literatura em língua portuguesa.

5.1.3 Atributos

Em adição ao texto contido dentre de um *tag* de elementos, um elemento também pode possuir atributos. Atributos são inclusos com seu respectivo valor dentro da declaração de abertura de um elemento.

Os nomes de atributos devem seguir as mesmas regras dos nomes de elementos XML, e os valores de atributos devem estar entre apóstrofes. Embora tanto apóstrofes duplas quanto simples são permitidas, apóstrofes duplas são mais utilizadas, pois permitem que os documentos XML mapeiem diretamente para práticas de programação Java. Além disso, marcação com apóstrofes duplas ou simples podem ser utilizadas em valores de atributos, cercando o valor em apóstrofes simples permite que as apóstrofes duplas sejam parte do valor. Essa não é uma boa prática de programação, pois tradutores e processadores XML quase sempre convertem as apóstrofes em torno do valor de um atributo para apóstrofe toda dupla (ou toda simples), assim introduzindo resultados inesperados.

Além da necessidade de saber como usar um atributo, também é importante saber quando usar um atributo. Por que XML permite uma grande variação na formatação da informação, é raro um atributo não poder ser representado por um elemento, ou que um elemento não possa facilmente ser convertido em um atributo. Embora não exista nenhum especificação ou padrão largamente aceito para determinar quando usar um atributo ou quando usar um elemento, existe uma boa regra: utilize elementos para informações com valores múltiplos e atributos para informações com valores únicos. Se a informação possui valores múltiplos ou é muito extensa, a informação provavelmente pertence a um elemento, logo ela pode então ser tratada primariamente como uma informação textual e é facilmente localizável e utilizável. Entretanto, caso a informação é primariamente representada por um valor único, então ela seria melhor representada por um atributo. Se depois de toda essa análise ainda existe certa dúvida, então é melhor utilizar um elemento para manter uma margem de segurança.

Em XML, raramente existe somente uma maneira para representar a informação utilizada, além de existirem várias boas maneiras para se concluir a mesma tarefa. Mais de uma vez, o aplicativo e o uso da informação ditam o que faz mais sentido.

5.1.4 Referências a entidades[14]

Uma referência a uma entidade é um tipo de informação especial em um documento XML e é usado para referenciar outro pedaço de informação. A entidade consiste de um nome único precedido de um ampersand e seguido por um ponto-e-vírgula: `&[nome da entidade];`. Quando um tradutor XML vê uma dessas referências, o valor especificado é substituído e nenhum processamento desse valor ocorre. Ver-se-á que ao tradutor XML é dito o que referenciar somente quando o mesmo vê a referência a entidade, sendo que a referência a entidade é feita somente quando se constrói o arquivo DTD ou *Schema*.

[14] Em inglês, *Entity references*.

5.1.5 Informação não-traduzível

A último construtor a ser visto para documentos XML é o marcador de sessão CDATA. Uma sessão CDATA é usada quando uma quantidade significante de informação deve ser passada para o aplicativo que fez a chamada sem qualquer tradução pelo tradutor XML. Ele é utilizado quando uma quantidade enorme de caracteres deve ser enviada utilizando referências a entidades, ou quando o espaçamento deve ser preservado.

A informação dentro de uma sessão CDATA não deve utilizar referências ou outro mecanismo para alertar o tradutor que caracteres reservados estão sendo utilizado, ao invés, o tradutor XML passa esta informação sem qualquer mudança para o aplicativo.

5.2 Restrições

Sem restrições aos documentos, é praticamente impossível (na maioria dos casos) dizer o que a informação significa, pois este arquivo que irá definir os elementos e atributos que formam um documento, assim como a ordem com que estes itens aparecem em um documento XML.

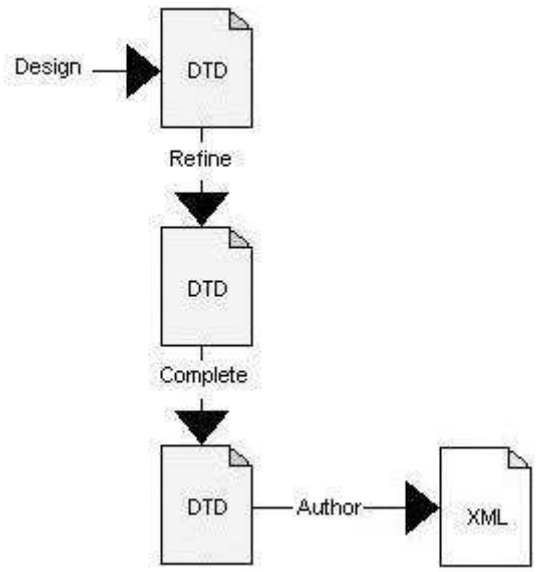

Figura 5.2: Criação de restrições para um documento XML.

5.2.1 DTDs

Um documento XML não é muito utilizável sem um arquivo DTD ou *Schema* que o acompanhe. Assim como XML pode eficientemente descrever informação, o DTD faz com que essa informação seja utilizável por muitos programas diferentes com uma variedade de jeitos ao definir uma estrutura para a informação.

O DTD define como a informação é formatada. Ele deve definir cada elemento permitido em um documento XML, os atributos permitidos e possivelmente os valores

aceitáveis para esses atributos para cada elemento, a ocorrência e encadeamento de cada elemento, e qualquer entidade externa.

```
<!ELEMENT Sessao (IDSessao, Contexto, ora:copyright)>
<!ATTLIST Sessao
            xmlns       CDATA #REQUIRED
            xmlns:ora   CDATA #REQUIRED
>
<!ELEMENT <#PCDATA)>
<!ATTLIST   IDSessao
            ora:series (1 | 12 | 123 | 1234)   #REQUIRED
>

<!ELEMENT Contexto (Transicao+)>
<!ELEMENT Transicao(IDTransicao+)>
<!ATTLIST Transicao
            number      CDATA #REQUIRED
>
<!ATTLIST IDTransicao
            number      CDATA #REQUIRED
>
```

5.2.1.1 Elementos em DTDs

A maior parte de um DTD é composta de definições do tipo ELEMENT e definições do tipo ATTRIBUTE. Uma definição de elemento começa com a *keyword* ELEMENT, antecedida da abertura padrão <! da *tag* DTD, e então o nome do elemento. Seguindo esse nome está o modelo de conteúdo do elemento. O modelo de conteúdo fica entre parênteses geralmente, e especifica qual o conteúdo que pode ser incluso dentro de um elemento.

Deve-se manter cuidado, pois nesse caso padronizado, a ordem especificada do conteúdo no modelo é a ordem no qual os elementos devem aparecer dentro do documento. Além disso, cada elemento deve aparecer, uma e somente uma única vez, quando nenhum modificador é utilizado. Se estas regras são quebradas, o documento não é considerado válido (porém ele ainda poderia ser bem-formado).

Em muitos casos vai ser necessário especificar a ocorrência múltipla de um elemento, ou ocorrências opcionais. Pode-se fazer isso com o uso de modificadores de ocorrência que podem ser vistos na Tabela 5.1.

Tabela 5.1: Modificadores para elementos em um arquivo DTD.

Operador	Descrição
[default]	Deve aparecer uma e somente uma única vez (1)
?	Deve aparecer uma vez ou nenhuma (0..1)
+	Deve aparecer ao menos uma vez até um número infinito (1..N)
*	Pode aparecer qualquer quantidade de vezes, incluindo nenhuma (0..N)

Se um elemento possui um caracter de informação dentro dele, a *keyword* #PCDATA é usada no modelo de conteúdo.

Se um elemento deve sempre ser um elemento vazio, a palavra EMPTY é usada.

5.2.1.2 Atributos em DTDs

Assim que você tiver criado todas definições de elementos, deve-se definir os atributos onde eles são definidos através da *keyword* ATTLIST. O primeiro valor é o nome do elemento, então abaixo se tem a definição dos vários atributos. Essas definições envolvem dar o nome do atributo, o tipo de atributo, e então se o atributo é requerido ou implícito (que significa que ele não é requerido, essencialmente). A maioria dos atributos com valores textuais irão simplesmente ser do tipo CDATA.

Também se pode especificar um conjunto de valores que um atributo pode ter para que o documento seja considerado válido.

5.3 Transformações

Transformações XML são bastante úteis, porém não são simples de serem implementadas. De fato, ao invés de tentar especificar a transformação do XML na especificação original XML 1.0, três recomendações diferentes foram feitas para definir como transformações deveriam ocorrer. Embora uma dessas (*Xpath*) também é usada em diversas outras especificações XML, a maior utilidade dos componentes listados nesse resumo será para a transformação de um arquivo XML de um formato para outro.

Porque essa três especificações estão ligadas de maneira bem forte e quase sempre são usadas em conjunto, raramente existe uma distinção clara entre as mesmas. Isso pode levar a uma discussão que é fácil de entender, mas não é tecnicamente correta. Em outras palavras, o termo XSLT que refere, especificamente, a *Extensible Stylesheet Transformations*, é aplicado tanto para *Extensible Stylesheets* (XSL) quanto para *Xpath*. No mesmo caminho, XSL é usado para se referir as três tecnologias muitas vezes.

XSL é definido como uma linguagem para expressar *Stylesheets*[15]. Essa definição extensa pode ser quebrada em duas partes:

- XSL é uma linguagem para transformar documentos XML;
- XSL é um vocabulário XML para especificar o formato de documentos XML.

As definições são similares, mas uma é lida como a mudança do formato de um documento XML para outro formato, enquanto que a outra foca na apresentação do conteúdo dentro de cada documento. Talvez uma definição mais clara seria dizer que XSL lida com a especificação de como transformar um documento do formato A para o formato B. Os componentes da linguagem lidam com o processamento e a identificação dos construtores usados para fazer isso.

O conceito mais importante para entender em XSL é que toda informação dentro dos estágios de processamento XSL estão em estruturas do tipo árvore. De fato, as regras que definem o XSL são elas próprias guardadas em estruturas de árvore. Isso

[15] Optou-se por manter o nome em inglês para esta palavra onde a mesma tenta representar um marcador de estilo.

permite um processamento simples da estrutura hierárquica de documentos XML, ou seja, modelos são usados para igualar o elemento raiz do XML sendo processado, então as regras “folha” são aplicadas aos elementos “folha”, filtrando até o elemento mais entrelaçado. Em qualquer ponto da progressão, elementos podem ser processados, estilizados, ignorados, copiados ou ter uma variedade de outras funções feitas sobre eles.

Uma boa vantagem dessa estrutura de árvores é que ela permite que o agrupamento de documentos XML seja mantido. Se o elemento A contém os elementos B e C, e o elemento A é movido ou copiado, os elementos contidos dentro dele recebem o mesmo tratamento.

XSLT é a linguagem que especifica a conversação de um documento de um formato para outro (onde XSL definiu os meios para a especificação). A sintaxe usada dentro do XSLT é geralmente preocupada com as transformações textuais que não resultam em uma saída de informação binária. Por exemplo, XSLT é instrumental em gerar HTML ou WML de um arquivo XML.

Xpath provê o mecanismo para se referir a uma larga variedade de nomes para elementos e atributos, além dos valores, em um arquivo XML. Com essa estrutura complexa que um documento XML pode ter, localizar um elemento específico ou conjunto de elementos pode ser difícil. È mais difícil por que o acesso ao DTD ou conjunto de restrições que resumem a estrutura do documento não podem ser assumidos; documentos que não são validados devem ter a possibilidade de serem transformados assim como documentos válidos podem. Para alcançar esse endereçamento de elementos, *Xpath* define a sintaxe em linha com a estrutura de árvore do XML e os processos XSLT para construir o documento usando essa sintaxe.

Referenciando qualquer elemento ou atributo dentro de um documento XML não é facilmente realizável ao especificar o caminho para o elemento relativo ao elemento atual sendo processado. Em outras palavras, se o elemento B é o elemento corrente e o elemento C e D estão dentro dele (inferiores), um caminho relativo encontra estes dois mais facilmente. O *Xpath* fornece essa possibilidade ao transformar um documento XML.

5.4 Tradutor

Para se utilizar a linguagem XML é necessário um tradutor XML, portanto uma das camadas mais importantes para qualquer aplicativo que utilize XML é o tradutor XML. Esse componente é o responsável por pegar um documento XML cru como entrada e fazer esse documento ter um sentido, ele irá garantir que o documento é bem formado, e se um DTD ou *Schema* for referenciado, ele será capaz de garantir que o documento é válido. O que resulta de um documento XML ser traduzido é tipicamente uma estrutura de dados que pode ser manipulada e utilizada por outras ferramentas XML ou APIs Java. Por agora, é importante ter uma idéia de que o tradutor é um dos blocos centrais de construção para a utilização de informação XML.

Devido a todos essas variáveis, a escolha de um tradutor XML não é simples. Não existem regras rápidas e fáceis para essa escolha, mas dois critérios são os principais na hora da escolha. A primeira é a velocidade do tradutor. Como documentos

XML são usados mais vezes e sua complexidade aumenta, a velocidade do tradutor XML se torna extremamente importante para a performance de todo o aplicativo. O segundo fator é a conformidade a especificação XML. Porque a performance é uma prioridade maior em relação a algumas funcionalidades obscuras do XML, alguns tradutores não implementam todos os pequenos pontos da especificação XML para que seja possível obter uma redução do custo de processamento. É tarefa do desenvolvedor fazer a escolha entre os dois fatores de acordo com as necessidades de seu aplicativo. Além disso, a maioria dos tradutores XML possui a opção de validação, o que significa que eles oferecem a opção de validar seu arquivo XML através de um DTD ou XML *Schema*, mas alguns não oferecem essa opção. É importante se utilizar um tradutor com esta opção de validação se essa capacidade é necessária para o aplicativo sendo desenvolvido.

5.5 APIs Java de baixo nível

O material nesta seção foi obtido da referência [5]. A mesma referência foi utilizada nas seções 5.6 e 5.7.

Uma API é uma interface de programação de um aplicativo e uma API de baixo nível para XML permite que o desenvolvedor interaja diretamente com o conteúdo do documento XML. Em outras palavras, praticamente não existe pré-processamento e se obtém o conteúdo XML cru para manipular. Além disso, esse é o meio mais eficiente de interagir com XML, além do mais poderoso. Ao mesmo tempo, se requer a maior quantidade de conhecimento de XML para trabalhar com essa API, e geralmente envolve a maior quantidade de trabalho para tornar o conteúdo de um documento em algo utilizável.

As duas APIs de baixo nível mais comum atualmente são a SAX, *Simple API for XML*, e a DOM, *Document Object Model*. Além disso, JDOM tem obtido muita força atualmente. Todas essas três APIs possuem alguma forma de padronização (SAX como a API de fato, DOM através da W3C e JDOM pela Sun), e são boas apostas para tecnologias de longo prazo. Todas três oferecem acesso a documentos XML de forma diferente, e permitem que se faça praticamente tudo o que se desejar com o documento. Outra utilidade importante é o JAXP, *Java API for XML Parsing*, que fornece uma camada de abstração bem fina para o processamento de documentos XML utilizando o SAX e DOM, ou seja, a mesma não provê um novo meio para traduzir XML, como SAX, DOM ou JDOM, e nem fornece uma nova funcionalidade em lidar com Java e XML. Ao invés disso, ele facilita lidar com algumas tarefas difíceis com DOM e SAX. Ele também torna possível lidar com tarefas específicas de cada vendedor encontradas quando se utiliza as APIs DOM e SAX que permite que essas APIs sejam utilizadas em um ambiente neutro a implementação dos fabricantes.

Um estudo um pouco mais aprofundado do SAX e do DOM auxiliará na compreensão da maneira como ela serão utilizadas no sistema criado, pois elas são os blocos básicos para todas outras APIs.

SAX foi a primeira grande API feita para tradução de XML para Java e permanece como o bloco básico de construção para quase todas outras APIs. SAX é

baseado em uma entrada do tipo rajada[16] que lê a informação de uma fonte de entrada XML pedaço por pedaço. Para se utilizar o SAX para tradução, se registram várias implementações para o manipulador[17] para que seja possível manipular o conteúdo, erros, entidades e assim por diante. Cada interface é feita de vários métodos de *callback*[18] de manipulação que recebem informação sobre os dados específicos sendo enviados para o tradutor, como os caracteres de informação, o início de cada elemento e o fim de cada elemento. Seu aplicativo baseado no SAX então pode usar essa informação para executar tarefas dentro dos métodos de implementação de *callback* de acordo com a demanda. A grande vantagem oferecida pelo SAX é a velocidade na conversão da informação, entretanto essa velocidade é paga com um certo grau de complexidade da API.

O DOM apresenta uma curva de aprendizado bem menor que o SAX devido a maneira como foi planejada sua arquitetura. Essa API fornece um modelo completo em memória do documento XML. DOM não é um tradutor (assim como o SAX também não é); ele requer um tradutor XML que faça uma implementação da API DOM para ser executado. Quando o tradutor finaliza sua leitura do documento XML, o resultado é uma árvore DOM e esse modelo em árvore de um documento XML é composto por elementos-pai que têm filhos, nós textuais, comentários e outras construções XML. Pode-se facilmente transitar pela árvore DOM utilizando uma API DOM e geralmente se movendo pelas folhas com certa facilidade. Porque é necessário se esperar pela tradução completa antes de se utilizar a árvore DOM, o mesmo é geralmente mais lento que o SAX, pois este primeiro precisa criar objetos para cada estrutura XML e isso gasta muito memória até se ter uma árvore em operação. Entretanto, essas desvantagens são equilibradas com um modelo de programação relativamente mais simples, um meio para transformar o conteúdo em uma árvore DOM, além de várias implementações oferecerem várias outras opções.

5.6 APIs Java de alto nível

O próximo passo no processamento de documentos XML são as APIs de alto nível. Ao invés de oferecer acesso direto a um documento, estas APIs depende das APIs de baixo nível para fazer o trabalho por eles. Além disso, essas APIs apresentam o documento em um formato diferente, de uma maneira mais fácil para o usuário, modelada de uma maneira específica, ou em alguma outra forma diferente da estrutura típica de um documento XML. Enquanto essas APIs são mais fáceis de usar, além de serem mais rápidas para se desenvolver com as mesmas, há um custo adicional de processamento enquanto a informação é convertida para um formato diferente. Outro ponto é que se gastará algum tempo para se aprender todas as funcionalidades da API, além de ser necessário o estudo da API de baixo nível também.

[16] Utilizará-se o termo rajada para representar o termo *streaming*, pois este termo em inglês representa o envio de uma larga quantidade de dados com certos períodos de pausa.
[17] O termo *handler* será traduzido para manipulador durante todo este documento.
[18] *Callback* é um tipo especial de método e não se encontrou uma tradução boa para este termo, então se optou por utilizá-lo na sua forma original em inglês. Este termo denota uma comunicação onde o provedor se conecta com o usuário depois que o usuário confirmou sua decisão de se conectar ao servidor, ou seja, um método de confirmação da transferência de dados.

Data binding[19] permite que se pegue um documento XML e forneça esse documento como um objeto Java para o desenvolvedor. Não um objeto baseado em uma árvore, mas um objeto Java no qual o desenvolvedor pode especificar o tipo de objeto. Por exemplo, se existem elementos chamados "pessoa" e "primeiroNome", seria possível um objeto com métodos do tipo `getPessoa()` e `setPrimeiroNome()`. Obviamente, essa é uma maneira simples para começar a utilizar XML, pois nenhum conhecimento mais aprofundado seria necessário. Porém, não é possível mudar a estrutura do documento facilmente, portanto *data binding* só pode ser utilizada em certos tipos de aplicativos.

5.7 Processos para o Data Binding

O Data Binding de XML para Java ou vice-versa pode ser definido pelo fluxo dado pela Figura 5.3.

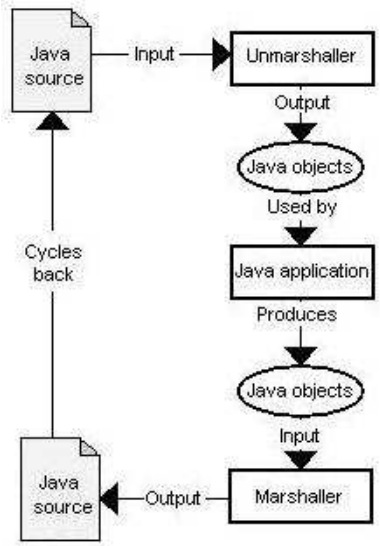

Figura 5.3: Fluxo para o Data Binding.

5.7.1 Geração de classes

5.7.1.1 Fluxo do processo

Primeiro, deve-se dar uma olhada no fluxo de processos envolvidos com a geração do arquivo de restrições, as *constraints*. Isso auxiliará na compreensão de todo esse processo. Também será útil para formar um *checklist* mental que deve ser seguido para a geração de classes onde se um dos passos abaixo foi esquecido, então problemas podem surgir, logo é bom se executar os passos na ordem dada.

1. Crie um conjunto de restrições para a informação XML;
2. Crie um binding *Schema* para converter as restrições para Java;
3. Gere as classes usando o *framework* de binding;
4. Compile as classes e garante que elas estão prontas para serem utilizadas.

[19] Optou-se por manter o nome dessa função em inglês, pois não se encontrou um termo em português para representar essa função que agradasse ao autor.

O processo acima é resumido na Figura 5.4.

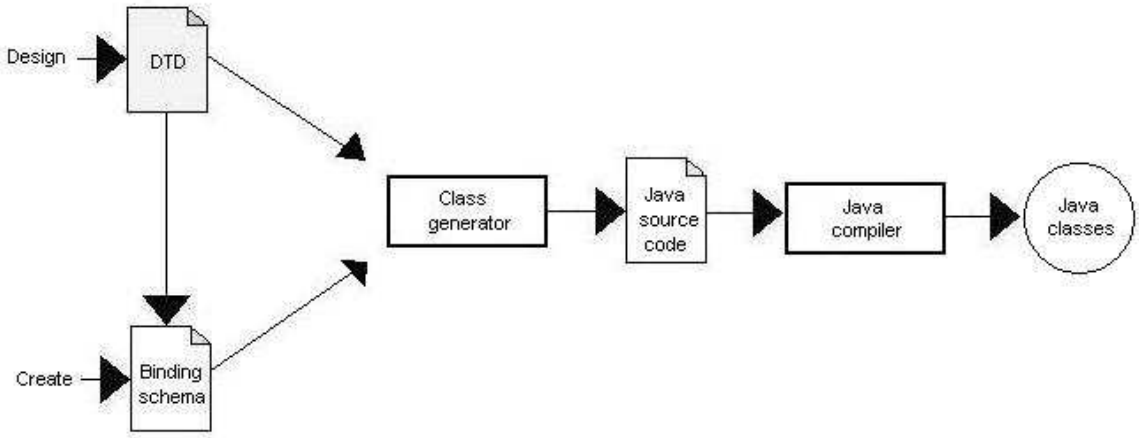

Figura 5.4: Fluxo para a geração de classes.

5.7.1.2 Definição

O primeiro processo, a geração de classes, fornece o meio para se converter um documento XML em um documento Java. Quando o *data binding* converte um documento XML em uma representação Java, ele procura providenciar acesso somente para a informação no documento. Ele faz isso ao criar a representação Java com o método de acesso (`get`) e o método de mutação (`set`) como `getItem()` ou `setPosition()`, ao invés dos métodos `getElement()` e `setAttribute()` usados nas APIs de baixo nível. Isso faz com que lidar com esses documentos seja algo menos voltado para programação Java e algo mais voltado para a lógica de negócios do sistema, ou seja, é uma boa coisa, obviamente. Entretanto, essas classes Java devem existir antes que o documento XML seja transformando em uma instância de uma dessas classes, esta é a função da geração de classes.

A geração de classes é o processo de pegar um conjunto de restrições XML e gerar classes Java (e talvez interfaces) desta restrição. As restrições XML, arquivos DTD ou XML *Schema*, seriam equivalentes as definições da classe Java, ou seja, eles definem o meio como a informação é representada. Em outra mão, um documento XML é equivalente a uma instância dessas classes, então as classes são simplesmente um conjunto de informações que contempla o contrato definido pelas restrições do documento.

Esse processo é muito mais útil do que escrever centenas de linhas de *callback*s SAX. Portanto o trabalho com o documento se torna algo bem simples, ao invés de um exercício das habilidades do programador de Java e da API XML.

5.7.2 *Unmarshalling*[20]

5.7.2.1 Fluxo de processos

Assim como na geração de classes, é importante se ver rapidamente o fluxo de processos do processo de *unmarshalling* – desordenamento - de informação XML em objetos Java. Isso é útil na hora de entender perfeitamente o que está ocorrendo quando se invoca o método `unmarshal()` ou qualquer que seja o nome dado ao método no *framework*. A Figura 5.5 mostra o mesmo fluxo de processo.

1. Construa a informação XML para ser *unmarshal* em objetos Java;
2. Converta a informação XML em instâncias dos objetos Java gerados;
3. Utilize as instâncias dos objetos Java resultantes.

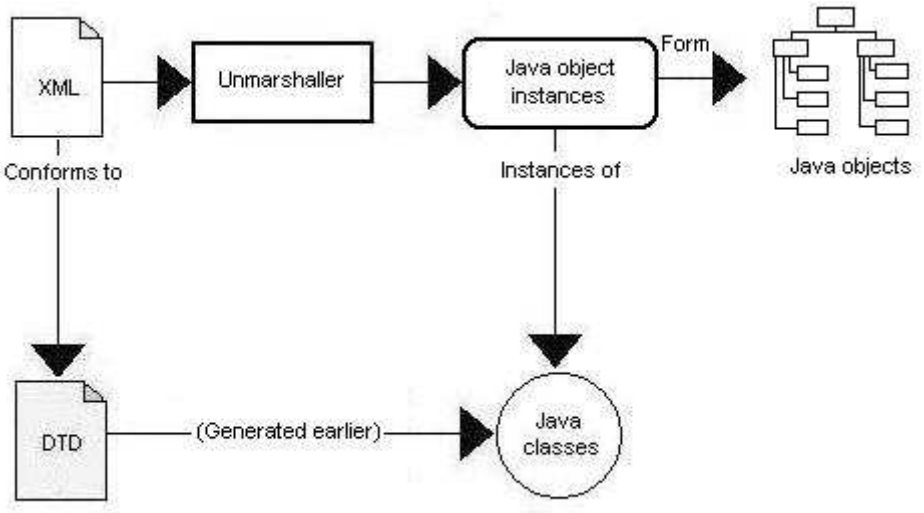

Figura 5.5: Fluxo para o unmarshalling.

5.7.2.2 Definição

É bom se ter uma noção que a partir do momento que se tem um conjunto de classes foram geradas, ainda não se tem uma grande utilidade para elas. Poderia-se usar uma API XML de baixo nível já existente para ler o documento XML, abstrair as informações, ou criar novas instâncias da classe gerada e popular a classe com a informação obtida do XML. Mas o *data binding* fornece tudo isso. De fato, os *frameworks* de *data binding* provêem exatamente esse processo. E nesse contexto, *unmarshalling* é o processo de converter um documento XML em uma instância de uma classe Java.

Esse é um processo relativamente simples: obtém-se um documento XML, se executa o mesmo em uma ferramenta ou instância de classe do *framework* de *data binding*, assim se obtém os objetos Java. O resultado é, geralmente, uma instância da classe do objeto Java representando o documento. Também será necessário,

[20] Optou-se por manter esse termo na sua forma original em inglês onde o melhor termo para *unmarshalling* seria algo ligado com desordenar a informação.

tipicamente, uma troca[21] do objeto `Java.lang.Object` para o tipo de objeto da classe específica que se está esperando, pois o *framework* não saberá nada sobre as classes usadas, pois elas foram geradas. Depois de feita a mudança para o tipo de objeto correto, então se está pronto para trabalhar com o objeto normalmente, e não como um documento XML. Então se podem usar os vários métodos de acesso (`get`) e mutação (`set`) para lidar com essa informação, e quando se estiver pronto para mandar o documento de volta para o formato XML, se executa um processo de *marshal* do documento.

5.7.3 *Marshalling*[22]

5.7.3.1 Fluxo do processo

Assim como no processo de *unmarshalling*, existem três processos básicos ao converter objetos Java (ou um conjunto de objetos) em XML. Eles são os seguintes:

1. Valide os objetos Java para garantir a validade da informação;
2. Converta os objetos Java com as informações em documentos XML;
3. Utilize ou armazene os documentos XML resultantes.

A Figura 5.6 define o fluxo de processo para o *marshalling*.

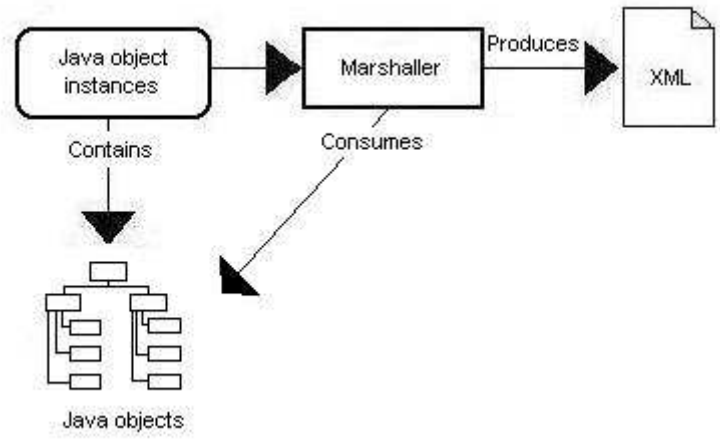

Figura 5.6: Fluxo para o marshalling.

5.7.3.2 Definição

Marshalling é o oposto de *unmarshalling*. Ele é o processo de converter um objeto Java e seus objetos dependentes em uma representação XML. Em muitos casos, *marshalling* é parte de um ciclo repetido de transformações de e para XML, e está em conjunto com o *unmarshalling*.

Existem duas maneiras distintas para fazer o processo de *marshal* de um objeto Java. A primeira é invocar o método `marshal()` em um objeto; esse método é

[21] Essa troca do tipo do objeto para um outro tipo de objeto se chama de *cast*.
[22] Processo oposto ao *unmarshalling* e ligado com o ordenamento da informação em um documento XML.

geralmente gerado junto com os métodos de acesso e mutação durante a geração da classe. O método recursivamente chama o método `marshal()` em cada um dos objetos dependentes, até que se termine em um documento XML. Nota-se que o documento XML alvo não precise ser o mesmo que o documento XML original; pode-se facilmente terminar com um grande número de documentos XML armazenados ao se utilizar diferente nomes de arquivo no mesmo processo de *marshalling*.

Uma abordagem diferente para o processo de *marshalling* é criar uma classe *standalone* que execute o *marshalling*. Ao invés de invocar o `marshal()` no objeto gerado, se invoca o `marshal()` na classe *standalone*, e é passada para a mesma o objeto a ser ordenado. Esse processo é mais útil, pois executa as mesmas tarefas ilustradas anteriormente e também faz as classes que não foram originalmente desordenadas do XML serem convertidas para o XML. *Data binding* usada dessa maneira se torna em uma arquitetura persistente, pois qualquer objeto com propriedades *bean-like* (objetos que possuam métodos do tipo `setXXX()` e `getXXX()`) poderia ser facilmente convertido para XML. Obtém-se o poder do *data binding* com a flexibilidade da persistência.

SISTEMA WIRELESS VILLAGE

6.1 Arquitetura do sistema WV

Embora o aplicativo sendo desenvolvido não faça parte do *Wireless Village* Server, o mesmo utiliza essa arquitetura na sua implementação, pois as classes criadas para o funcionamento do WVS são a base para a implementação do *PhoneBook*.

É interessante um estudo mais detalhado do sistema que irá ser implementado como base, ou seja, o *Wireless Village*. A arquitetura deste sistema é utilizada para se descrever o sistema IMPS e sua relação com as redes celulares já existentes e a Internet. A base do sistema é uma arquitetura cliente-servidor, onde o cliente pode ser um aparelho celular, um PDA ou um computador. Só se implementaram as duas primeiras alternativas desse projeto até o momento.

O *Wireless Village* Server é o ponto principal no projeto, é o ponto no qual se congregam todos os elementos que formam o sistema. Os quatro módulos que podem ser implementados nesse serviço são:

- **Presença**: responsável por definir a localização do usuário em uma área urbana, assim como, armazenar toda a lista de contato deles, seus humores (*moods*) e estados, ou seja, guardar suas definições de presença. Estas informações podem ser manipuladas explicitamente pelo usuário ou implicitamente pelo sistema;
- **Mensagens instantâneas**: responsável pela troca de mensagens em formato SMS ou XML/HTTP entre os diferentes usuários do sistema. Não existe um tamanho específico para o tamanho das mensagens usando o XML/HTTP. Além disso, diferentes tipos de mensagem podem ser enviadas como texto, imagens e som;
- **Grupo**: provêem habilidades para a manipulação de grupo de usuários que podem ser privados ou públicos. No fundo, um grupo nada mais é do que uma sala de chat;
- **Conteúdo**: permite a troca de conteúdo/informação entre os usuários do WV. O conteúdo compartilhado permite a troca de conteúdo entre usuários trocando mensagens ou em um grupo qualquer.

Para se implementar um WVS não é necessário a implementação dos quatro elementos de rede, porém uma implementação parcial já seria razoável. A Figura 6.1 mostra todos os serviços que cada módulo possui.

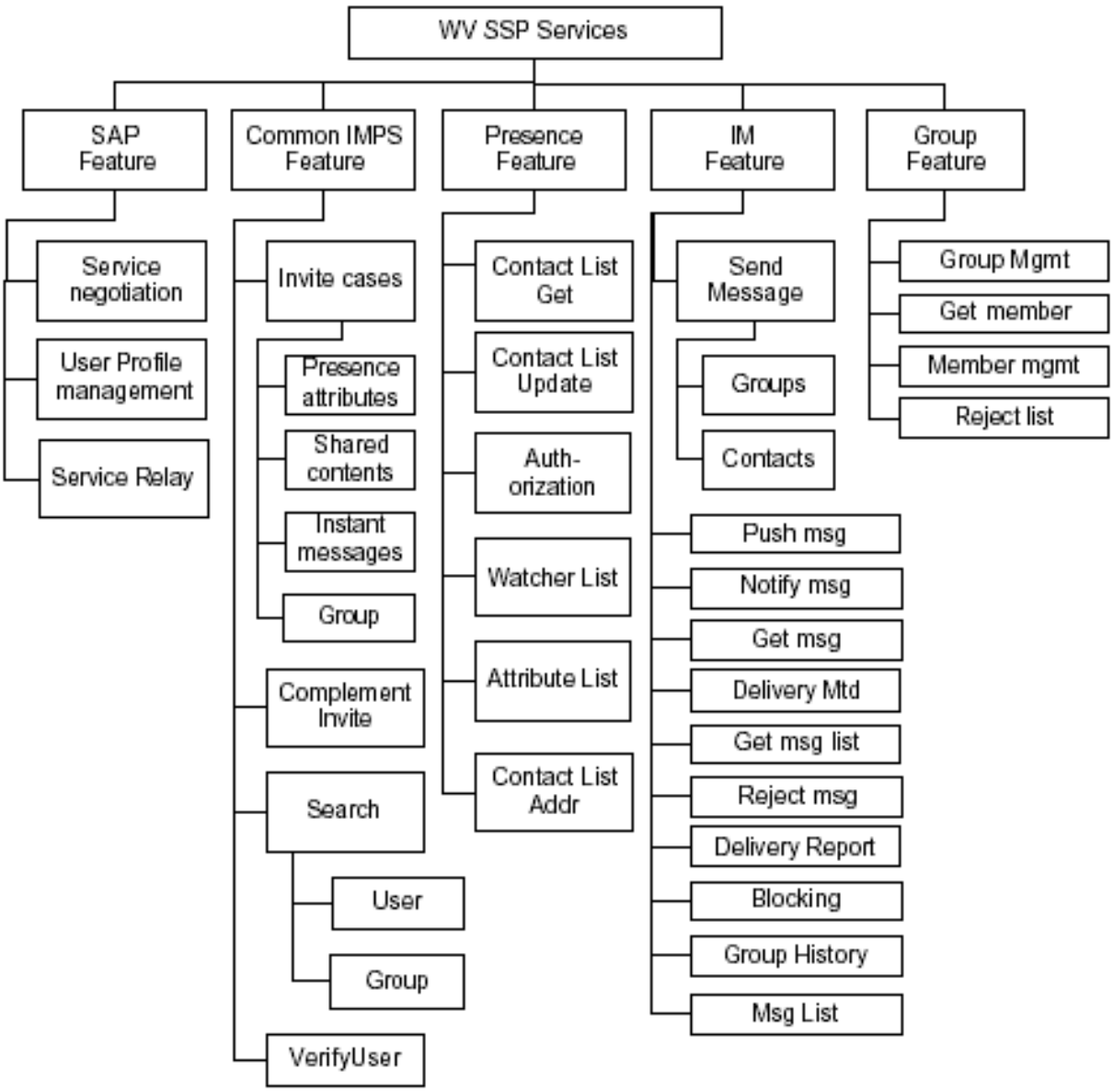

Figura 6.1: Serviços prestados pelo Wireless Village.

Observa-se que todos estes módulos são ligados ao SAP que é ligado a MCN conseqüentemente, assim fazendo toda a integração do WVS com a MCN a partir do SAP.

No caso do elemento de Presença, se torna necessário a existência de um iGMLC que será o responsável por buscar a localização do usuário a partir da infra-estrutura da operadora celular.

O papel do SAP no sistema é fazer a interligação entre os quatro elementos listados e o restante da rede pertencente a operadora. Ele possui interface para todos os elementos da rede: o WV client, o WVS, a MCN e gateways proprietários não-pertencentes ao WVS. As funções exercidas pelo SAP são:

1. **Autenticação e Autorização**: são os serviços utilizados para se verificar a identidade de alguma entidade e verificar o que uma entidade autenticada

tem direito de fazer. Os modos de autenticação/autorização são: Application-Network, User-Application, Application-Application, User-Network;

2. **Descoberta de um serviço e acordo para este serviço**: possibilita que a aplicação no celular identifique a capacidade total da coleção de serviços disponíveis. Esse processo ocorre a partir de um registro pelo qual o cliente envia uma notificação pelo servidor, então um acordo ocorre entre os servidores caso o serviço não esteja disponível neste servidor ou entre o servidor e o cliente para que o cliente possa interagir com a este serviço;
3. **Gerenciamento dos perfis de usuário**: descreve como um usuário deseja gerenciar ou interagir com os serviços de comunicação. A informação de perfil consiste de várias interfaces com o usuário, assim como, os serviços dos quais o usuário tem assinatura, suas preferências para estes serviços, o status destes serviços (ativo/inativo), privacidade de sua informação na rede, capacidade do terminal e preferência de interface com o terminal. Essa funcionalidade permite a busca e mudança das informações de perfil;
4. **Service Relay**: permite o roteamento de todas os pedidos ou respostas entre diferentes WVS usando o SSP.

6.2 Serviços de Presença

Para este item, se manterá um foco mais detalhado em um dos serviços que foi implementado no projeto, ou seja, o serviço de presença.

Algumas características do servidor de presença são: definição da informação de presença, mecanismos de transferência de informações de presença, e gerenciamento das listas de contatos.

Informação de presença não pode ser facilmente definida devido a vasta quantidade de informação disponível. Para garantir a interoperabilidade, a especificação decidiu manter certos padrões de localização, logo existem classes que guardam o status do cliente na rede e outras classes guardam o status do usuário onde os dois conjuntos de classe guardam todo o conjunto de informações de presença do usuário.

Atributos relacionados ao status do cliente descrevem o status do software rodando no WV cliente, assim como, o hardware do dispositivo. Entre estes se incluem atributos que descrevem o status do WV Client e seu dispositivo em relação a rede móvel, além de possuir informações mais detalhadas sobre o cliente como a sua versão e capacidades. O status na rede do cliente inclui o registro e o status online do cliente na rede, assim como, a localização do usuário e sua informação de endereço.

Atributos relacionados ao status do usuário incluem informações como a disponibilidade do usuário, suas informações de contato e a forma de contato preferida pelo usuário. Também incluem informações relacionadas à forma textual, status do conteúdo disponibilizado e o estado emocional do cliente, ou seja, seu humor de acordo com a escolha feita pelo usuário.

A lista de contato é uma lista mantida sobre o usuário WV e suas funções são: uma lista de distribuição quando se enviar mensagens instantâneas ou informações de presença e uma maneira para se fazer autorização pró-ativa de presença.

Um usuário pode manipular diversas listas de contato com diferentes propósitos. O gerenciamento das listas de contato deve possuir possibilidades como criação, remoção e modificação da lista de contatos, além de ser possível a obtenção de uma lista de contatos de outro usuário. Usuários também podem modificar o conteúdo de uma lista de contatos e recuperar o conteúdo de uma lista de contatos a partir do elemento de presença.

A autorização de atributos de presença é dividida em autorização pró-ativa no qual o usuário WV autoriza os atributos de presença antes que qualquer pessoa tenha feito um pedido por estes atributos e a resposta pode ser enviada para um usuário ou um grupo a partir de uma lista de contatos, e autorização reativa no qual o usuário autoriza de acordo com um pedido e funciona para um usuário somente

Na autorização pró-ativa, os atributos que podem ser enviados são definidos pela lista de atributos. Na autorização reativa, o usuário pedindo a informação declara quais atributos ele deseja receber.

O termo *Publisher* define o usuário que gerencia sua própria lista de contatos e será utilizado o próprio termo em inglês no restante deste projeto.

Na autorização pró-ativa, o *Publisher* cria sua lista de contatos e adiciona usuários manualmente na mesma, portanto, os membros desta lista estão pró-ativamente autorizados a acessar as informações de presença do *Publisher* contidas na lista de atributos padrão. Logo, ao criar uma lista de contatos, o *Publisher* pode definir uma lista de atributos que será associada a lista de contatos onde estes atributos podem ser visualizados pelos membros da lista.

Além disso, o *Publisher* também pode especificar um conjunto de atributos de presença que serão associados para um usuário em uma lista, mesmo que a lista possua seus próprios atributos. Caso ocorra um caso onde tanto o usuário quanto a lista de contatos possuam os mesmos atributos com diferentes restrições, então os atributos do usuário sempre sobrescreverão os atributos da lista de contato.

Quando um usuário está em diferentes listas de contatos com atributos diferentes, então o usuário tem direito ao conjunto de atributos contidos em todas listas.

O *Publisher* gerencia todo esse processo a partir de funções do tipo *create Contact List, delete Contact List, get contact list, add contact list member* e *remove contact list member*. Outras funções de gerenciamento incluem o *update* de atributos de presença de uma lista de contatos ou usuário, assim como, a remoção de atributos de presença de um usuário.

Mais algumas funções de suporte podem ser criadas para facilitar a vida do *Publisher* como *create attribute lists, delete attribute lists, get attribute lists, update attribute lists* e *attach and/or detach attribute lists to users and/or contact lists*.

Na autorização reativa, o usuário utilizando o IM pode fazer o pedido de alguns atributos ou todos os atributos de outro usuário. O serviço de presença envia um pedido de autorização para o usuário IM e caso autorizado, coloca o usuário que fez o pedido na *Watcher List* ou lista de observação. Não existe uma autorização parcial para a lista

de atributos e caso nenhuma lista seja criada no pedido, então se envia a *Default Public Attribute List* ao usuário que fez o pedido.

A Lista de Observação é uma lista de usuários definida pelo sistema com funcionalidades limitadas a guardar os usuários que foram inscritos para as informações de presença de outro usuário. Portanto, essa lista guarda todos os usuários que foram inscritos para um ou mais atributos de presença, mesmo que ele não tenha ganhado acesso aos atributos pelo qual se inscreveu. Entretanto, ser parte desta lista não garante que a autorização é válida ou que o usuário pedindo o acesso tenha acesso a qualquer informação de presença. Observa-se que cada usuário deve ter uma Lista de Observação no servidor. O *Publisher* tem direito de receber sua Lista de Observação com as informações de quem está inscrita nela a qualquer momento, não importando se é uma autorização pró-ativa ou reativa.

Tanto a autorização pró-ativa quanto a reativa podem ser canceladas pelo *Publisher* a qualquer momento. Um usuário inscrito também pode pedir para ser removido de uma lista a qualquer momento, ou seja, a inscrição deste usuário seria cancelada e seu nome removido da Lista de Observação.

Outro ponto importante é que a autorização pró-ativa tem maior prioridade em relação a autorização reativa. Portanto, se alguns atributos são autorizado pró-ativamente para o usuário, então nenhuma autorização reativa seria necessária.

Por último, o “reenvio” de valores de presença para um usuário cadastrado para um usuário já cadastrado para estar informações também um fator importante dentro do sistema. O sistema funciona da seguinte maneira, um usuário pode se inscrever para informações de presença de outro usuário por um período de tempo indeterminado. O usuário que fez o pedido irá inicialmente receber novas informações de presença e irá receber novas informações sempre quando os contatos do usuário forem sendo modificados. Além disso, um usuário pode pegar as informações de presença de outro usuário caso seja autorizado pelo dono desta informação. Esta utilidade é usada quando, por exemplo, um usuário quer ocasionalmente conferir o status de outro usuário. Um usuário também pode modificar seus atributos que são enviados aos usuários que possuem esse usuário em sua lista. Portanto, estas são todas as maneiras como as informações de contato ou localização podem ser trocadas entre os usuários do sistema de forma explicita ou implícita.

6.3 Mensagens Instantâneas

Um material mais elaborado sobre essa parte do sistema pode ser estudado na referência [13].

As características de troca instantânea de mensagens são realizadas pelo *Instant Messaging Service Element* no servidor. Dois diferentes tipos de mecanismo de entrega são suportados: o mecanismo de entrega direta e o *Notification/Pull*.

O destinatário de uma mensagem pode ser um ou mais usuários individuais, um grupo, ou usuários dentro de uma *Contact List*.

No tipo de entrega direta, o *Instant Messaging Service Element* envia a mensagem ao destinatário logo que ela chega. Este método de entrega é recomendado para mensagens de texto pequenas. Já no tipo *Notification/Pull*, a mensagem fica no servidor, e este manda apenas uma notificação de nova mensagem para o destinatário.

Na hora do envio de uma mensagem, o destinatário pode estar *offline*, não logado no sistema WV. Isso pode ser indicado por um atributo de presença, que o usuário pode verificar antes de enviar uma mensagem. O servidor pode implementar uma funcionalidade do tipo *store-and-forward*. Não sendo este o caso, a mensagem é simplesmente descartada. Caso seja, a mensagem fica armazenada no servidor por um certo período de tempo esperando o *login* do destinatário. Quando o *login* acontece, o servidor pode escolher entre entregar as mensagens uma a uma ou esperar que o usuário peça uma lista de mensagens não entregues.

Usuários têm a opção de bloquear mensagens utilizando-se de uma lista de bloqueio assim como também podem ter uma lista de usuários permitidos. Esta funcionalidade ainda não está implementada no projeto.

O *Wireless Village* prevê que a mensagem tenha qualquer tipo de conteúdo, incluindo conteúdo multimídia. Para este projeto, se limitou ao envio somente de mensagens de texto.

6.4 Acesso ao sistema

Todo cliente WV é obrigado a fazer sua autenticação e autorização em um servidor WV para poder utilizar os serviços WV e nesta etapa ocorre a criação de uma sessão. Além disso, o cliente pode fazer *Logout* do sistema a qualquer momento onde a sessão é desconectada no lado do servidor.

Um ponto muito importante ao se utilizar estes serviços é que as capacidades e serviços disponíveis a um cliente são negociados com o servidor antes que os serviços WV possam ser utilizados pelo cliente.

Na negociação de capacidades, o cliente WV indica ao servidor suas capacidades. A comunicação é adaptada e o conteúdo transmitido de acordo com estas capacidades, alguns termos negociados são: melhor método de envio (push/pull), tipos de conteúdo aceito, cifragem de transferência aceita, tamanho máximo de conteúdo aceito, *bindings* de transporte suportados.

Na fase de negociação, o cliente WV indica as funções que ele planeja usar durante a sessão e as quais o servidor WV precisa suportar. O servidor responde indicando as funções que ele aceita em suportar baseado na disponibilidade e perfil do usuário.

Outro fator importante é o envio de uma mensagem do tipo *Keep Alive* regulada por um timer quando não houver comunicação durante a sessão. O propósito desta mensagem é indicar para o servidor WV que o cliente está online e pronto para se comunicar.

Outro serviço disponível é uma mensagem enviada ao celular do cliente contendo o provedor do serviço WV, ou seja, seu nome, logotipo, texto descritivo e um URL. Este serviço é utilizado somente para a diferenciação da marca dos provedores de serviço.

6.5 Serviços comuns a todo sistema

Serviço de procura. Este serviço é utilizado para que um usuário localize outros usuários ou informações contidas em um serviço WV. Por enquanto, a procura será limitada a usuário e grupos.

Os critérios que podem ser utilizados pela busca são o username, o primeiro nome do usuário, seu sobrenome, seu email ou seu alias.

Grupos podem ser procurados se utilizando o id do grupo, seu nome ou tópico. Também é possível se procurar grupos que foram criados por um determinado usuário ou grupos no qual determinado usuário esteja cadastrado.

Convite geral. A característica de convite geral permite que um cliente WV convide um ou mais usuários para atividades como: convite para um chat em grupo, convite para troca de informações de presença e convite para trocar conteúdo compartilhado.

No pedido de convite, o usuário pode escrever o motivo para este convite e o receptor deve responder se aceita ou não o convite, assim como mandar um texto explicando sua escolha. O usuário que fez o pedido também pode cancelá-lo caso mude de idéia com o tempo.

Cada uma dessas habilidades é especificada como um tipo de mensagem contendo um código que identifica os erros, assim como, os diferentes parâmetros utilizados para seu pedido e resposta. As variáveis e as primitivas utilizadas em cada transação.

IMPLEMENTAÇÃO DO PHONEBOOK

7.1 Sistema utilizado

O sistema *Wireless Village* foi implementado utilizando o *Red Hat Linux Kernel* 2.4.20-8 em sua instalação completa e utilizando o J2SE SDK 1.4.1-02, o MySQL 3.23.1 e o WTK 2.0.

Como o *Wireless Village* foi implementado usando a linguagem Java, então o mesmo foi transferido para uma implementação baseada em Windows 2000 devido a portabilidade entre sistemas operacionais desta linguagem e a implementação do aplicativo *PhoneBook* sendo feita a partir do Windows 2000.

Utilizou-se o J2SE SDK 1.4.1-02 na implementação de todo código na parte do servidor, o *Wireless ToolKit*, WTK, 2.0_0.1 da Sun foi mantido como emulador para o ambiente J2ME, o MySQL 4.0.16 foi utilizado nesta nova implementação, porém nenhuma novidade desta nova versão chegou a ser utilizada no projeto, e utilizou-se o ambiente de desenvolvimento Eclipse 3.0 para se desenvolver o código do servidor. Para fazer toda transferência de mensagens XML/HTTP com o servidor foi instalado o Jakarta Tomcat 5 a partir da instalação do *Java Web Services Developer Pack* 1.3 da Sun e todos diagramas UML foram criados com o UMLStudio 7.1 ou o Microsoft Visio.

Para o funcionamento do sistema, foi necessária a adição dos diretórios contendo arquivos executáveis do Java e MySQL às *Variáveis de Ambiente* do Windows. Todas classes que compõem o servidor desenvolvido foram colocadas em um mesmo projeto no Eclipse para facilitar os testes do sistema utilizando o JUnit e facilitar na importação de outras classes necessárias para a tradução XML e a conexão com o banco de dados. A conexão com o banco de dados foi feita a partir da API *mysql-connector-java-3.0.9-stable* fornecida pela MySQL AB.

A instalação do MySQL foi a padrão e os testes feitos com ele resultaram na Tabela 7.1 onde se pode ver os valores utilizados na implementação do sistema.

Tabela 7.1: Variáveis utilizadas pelo MySQL.

Variable Name	Value
back_log	50
Basedir	/
binlog_cache_size	32768
character_set	latin1
character_sets	latin1 big5 czech euc_kr gb2312 gbk sjis tis620 ujis dec8 dos german1 hp8 koi8_ru latin2 swe7 usa7 cp1251 danish hebrew win1251 estonia hungarian koi8_ukr win1251ukr greek win1250 croat cp1257 latin5
concurrent_insert	ON
connect_timeout	5

Datadir	/var/lib/mysql/
delay_key_write	ON
delayed_insert_limit	100
delayed_insert_timeout	300
delayed_queue_size	1000
Flush	OFF
flush_time	0
have_bdb	NO
have_gemini	NO
have_innodb	NO
have_isam	YES
have_raid	NO
have_openssl	NO
init_file	
interactive_timeout	28800
join_buffer_size	131072
key_buffer_size	8388600
language	/usr/share/mysql/english/
large_files_support	ON
log	OFF
log_update	OFF
log_bin	OFF
log_slave_updates	OFF
log_long_queries	OFF
long_query_time	10
low_priority_updates	OFF
lower_case_table_names	0
max_allowed_packet	1048576
max_binlog_cache_size	4294967295
max_bilong_size	1073741824
max_connections	100
max_connect_errors	10
max_delayed_*threads*	20
max_heap_table_size	16777216
max_join_size	4294967295
max_sort_length	1024
max_user_connections	0
max_tmp_tables	32
max_write_lock_count	4294967295
myisam_max_extra_sort_file_size	256
myisam_max_sort_file_size	2047
myisam_recover_options	0
myisam_sort_buffer_size	8388608
net_buffer_length	16384
net_read_timeout	30
net_retry_count	10
net_write_timeout	60
open_files_limit	0

pid_file	/var/lib/mysql/Capella.pid
port	3306
protocol_version	10
record_buffer	131072
record_rnd_buffer	131072
query_buffer_size	0
safe_show_database	OFF
server_id	0
slave_net_timeout	3600
skip_locking	ON
skip_networking	OFF
skip_show_database	OFF
slow_launch_time	2
Socket	/var/lib/mysql/mysql.sock
sort_buffer	2097144
sql_mode	0
table_cache	64
table_type	MYISAM
*thread*_cache_size	0
*thread*_stack	65536
transaction_isolation	READ-COMMITTED
timezone	BRT
tmp_table_size	33554432
Tmpdir	/tmp/
Version	3.23.56
wait_timeout	28800

7.2 Modo de operação do PhoneBook

A operação do sistema passa por diversas etapas que são divididas da seguinte maneira:

1. O usuário do aparelho móvel entra com a informação a ser procurada no sistema onde esta informação pode ser o nome ou o telefone de outra pessoa;
2. O aparelho móvel faz a tradução do objeto `String` contendo a informação a ser procurada em um documento XML;
3. O documento XML é enviado para o servidor do sistema utilizando a execução de um método `doGet()` no *container* Tomcat;
4. O documento XML recebido do celular é traduzido para forma de objetos Java contendo a informação a ser procurada (no servidor);
5. O servidor faz uma consulta ao banco de dados usando a informação do usuário como a restrição à busca;
6. O resultado da busca é traduzido de objeto Java em um documento XML;
7. O documento XML contendo o resultado da consulta é enviado ao aparelho móvel que traduz o documento XML para forma de `Strings` Java;
8. O resultado é visualizado no *display* do aparelho móvel.

Observa-se que tanto para a consulta de um nome quanto para a de um telefone, se utiliza o mesmo procedimento listado acima. As únicas diferenças estão na classe que faz o pedido no aparelho móvel e no manipulador que irá receber este pedido no servidor. A Figura 7.1 mostra, através de um diagrama de seqüências UML, a maneira como ocorre esse procedimento.

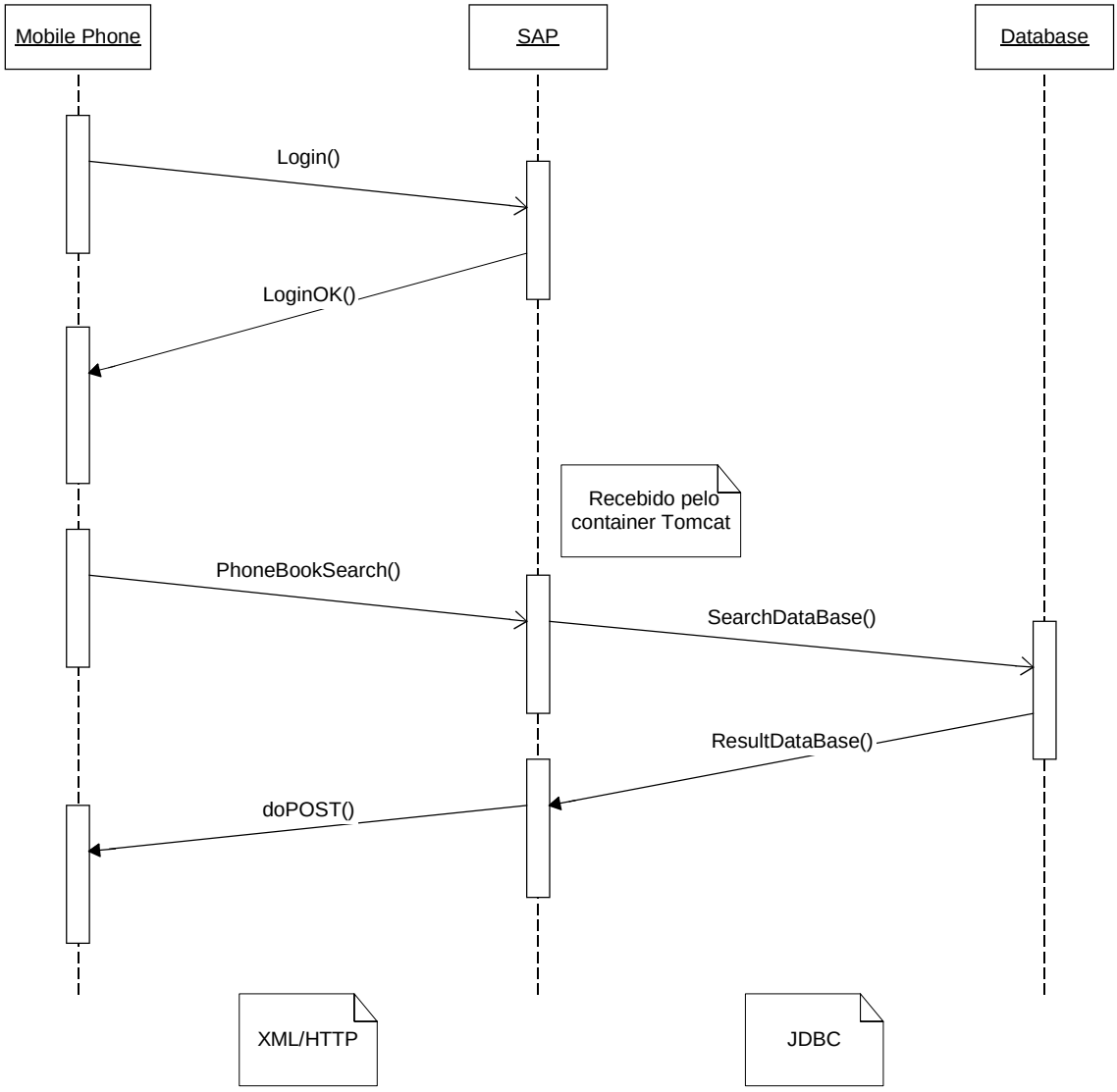

Figura 7.1: Diagrama de seqüência para a procura por nome ou telefone.

Além disso, é necessário que o usuário faça seu *login* a rede *Wireless Village* antes que qualquer utilização dos serviços possa ocorrer, esta função é responsabilidade do sistema *Wireless Village*, logo não precisou ser modificada para a implementação deste aplicativo.

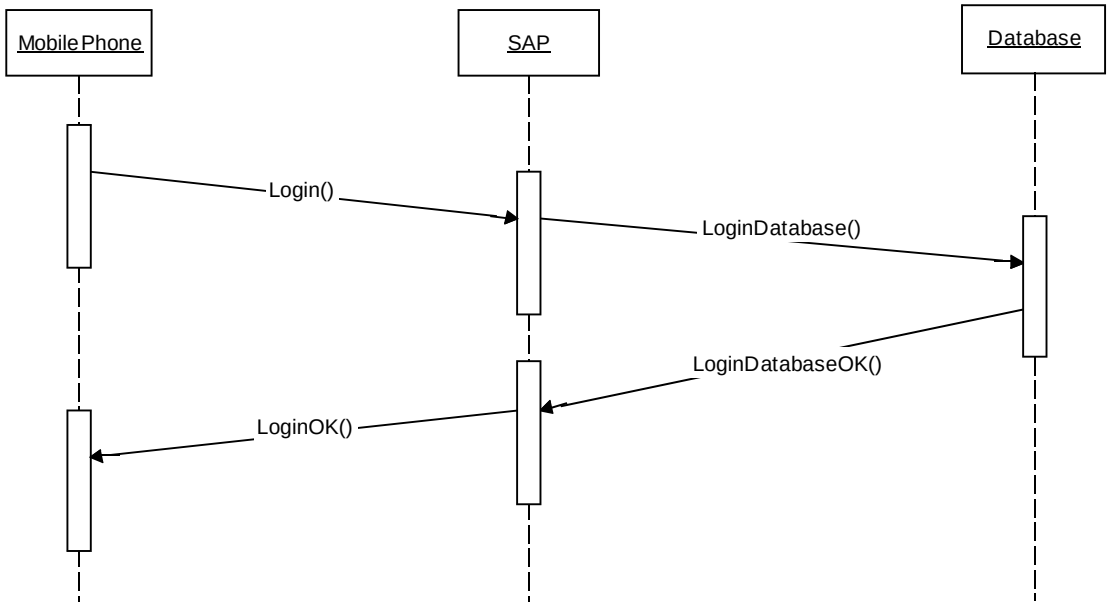

Figura 7.2: Diagrama de seqüência para o Login do usuário.

Os diagramas de seqüência para as outras funções realizadas pelo Wireless Village Server podem ser vistas nas referências [13] e [14].

7.3 Tabelas no MySQL

Pode-se notar que os dois diagramas acima fazem uma interação com o banco de dados do sistema, porém diferentes tabelas são usadas para armazenar os dados necessários.

A tabela que armazena as informações necessárias para o *Login* tais como o *ID*, U*serID*, P*assword*, *MIN*, ou S*essionID* se chama wv.*User*. A Tabela 7.2 apresenta todas colunas presentes nessa tabela, assim como os valores implementados para testar a interação entre as diferentes entidades do projeto.

Nota-se que a coluna *Last_update* possui valores que serão definidos a partir da utilização do programa, então somente se indicou o tipo de informação que é armazenada nessa coluna. A coluna ID é do tipo auto incrementável, logo não é necessário que o administrador atualize essa coluna, pois o banco de dados executa essa tarefa automaticamente. Por fim, a tabela abaixo foi projetada para que não seja possível que as colunas *UserID*, *SessionID*, *Serial* e *MIN* possuam valores iguais em dois campos diferentes.

Tabela 7.2: Informações de Login no Banco de Dados.

UserID	Password	Session ID	dDeliveryMetho	LengthAcceptedContent	Serial	MIN	Last_update
teste	teste	wv.usr.com#9999999@server.com	N	2048	1234567	9999999	<Date>
marcos	marcos	wv.usr.com#9998888@server.com	P	1024	1234566	9998888	<Date>
lemom	lemom	wv.usr.com#9997777@server.com	N	512	1234565	9997777	<Date>
Suporte	Suporte	wv.usr.com#9996666@server.com	P	4096	1234564	9996666	<Date>
Admin	Admin	wv.usr.com#9995555@server.com	N	2048	1234563	9995555	<Date>

ID	1	2	3	4	5

A segunda tabela necessária para o funcionamento deste aplicativo foi a *wv.PhoneBook* que armazena os dados *Name* e *Phone* de cada usuário, além de uma coluna *AllowSearch* que permite que cada usuário do sistema escolha se seus dados podem ser repassados ou não (1 = Sim e 0 = Não). A Tabela 7.3 mostra essa tabela com os valores de testes utilizados.

Tabela 7.3: Informações guardadas na tabela PhoneBook.PhoneBook.

Serial	Name	Phone	AllowSearch	Last_update
123456789	Eduardo Sirqueira	99823045	1	<Date>
123456788	Tatiana Martins	99611313	1	<Date>
123456787	Daniel Almeida	99758412	1	<Date>
123456786	Rafael Moraes	96268814	0	<Date>
123456785	Antonio da Silva	96198652	1	<Date>
123456784	Izabela Goes	9237410	0	<Date>
123456783	Marcela Santos	9968810	0	<Date>
123456782	Ana Paula Ferreira	9215697	1	<Date>
123456781	Joao Oliveira	99884103	1	<Date>
123456780	Barbara Valente	96984520	0	<Date>

As demais tabelas presentes no sistema Wireless Village podem ser vistas nas referências [13] e [14]. Os dois scripts utilizados para se criar as tabelas se encontram nos apêndices A e B.

Por fim, como este projeto foi separado em diferente módulos, logo foi necessária a realização de testes com o Banco de Dados. Para isto, se executou o comando *mysqld --console* para executar o servidor do MySQL 4.0.16, foi feito um teste com o comando *mysql –u root* para se conectar ao Banco de Dados como um cliente e foi executado o comando *telnet <IP> 3306* para testar a conexão com o banco de dados através de uma rede.

7.4 Implementação do Servlet

A aplicação servidor utiliza 12 classes para seu funcionamento onde uma classe é o *Servlet* implementado, cinco classes são utilizadas para o funcionamento do serviço de presença através da tecnologia RMI, uma classe implementa o serviço de mensagens instantâneas, duas classes são responsáveis pela autorização do cliente, uma classe implementa a interface gráfica utilizada pelo cliente RMI e uma classe implementa o aplicativo *PhoneBook*.

Neste projeto, foram feitas modificações nas classes `Sap_CSPservlet`, `BD` e `SAP_Authentication` para prover os novos serviços criados. Além disso, foi criada a classe `PhoneBookSearch` que implementa as funcionalidades da lista telefônica direta/reversa. Estas quatro classes se encontram nos apêndices C, D, E e F, respectivamente. O diagrama de classes para o Servlet pode ser visto na Figura 7.3.

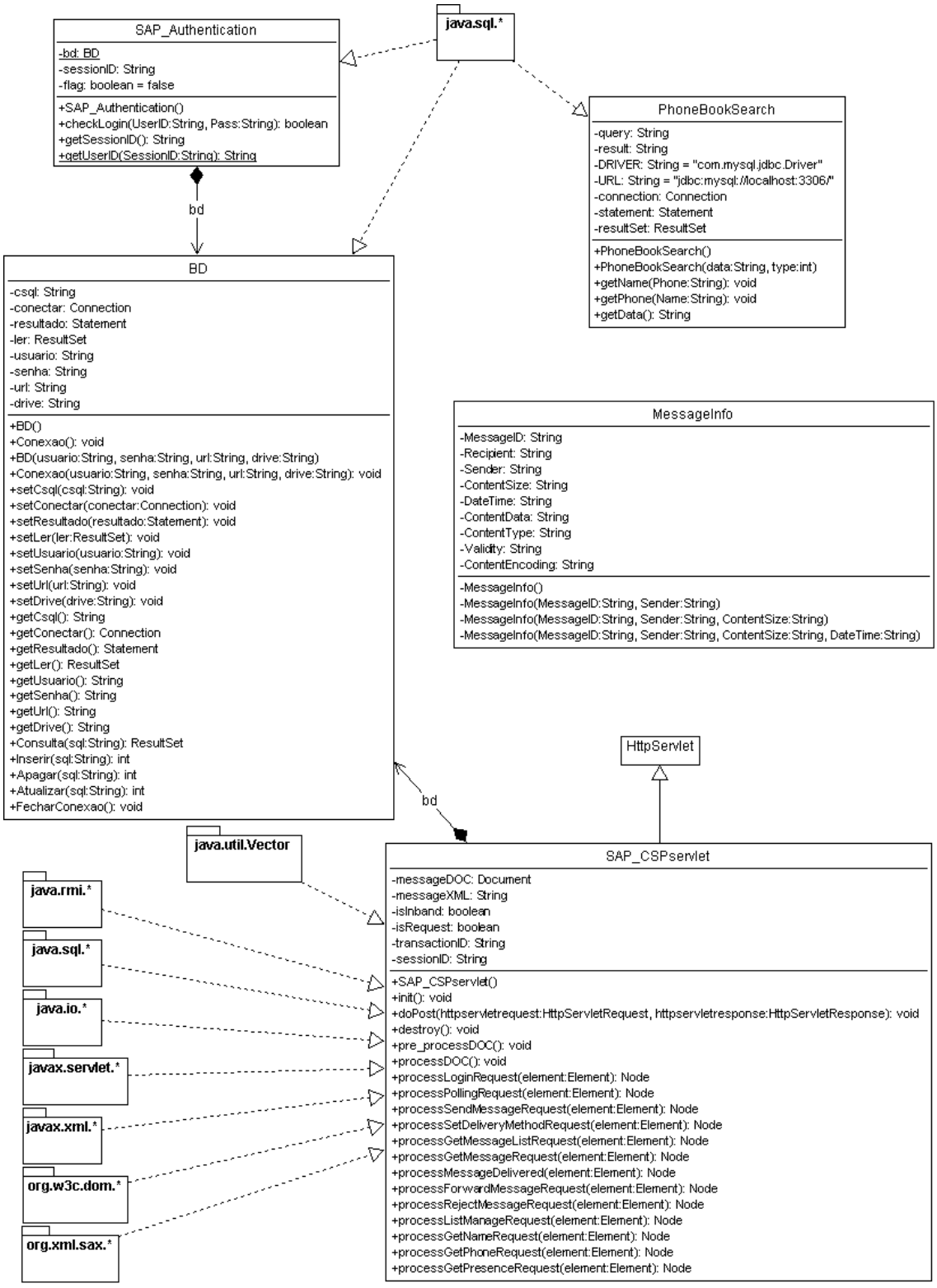

Figura 7.3: Diagrama de classes para o Servlet.

A classes `Sap_CSPservlet` é a classe principal na parte do servidor e implementa a classe `HttpServlet`. A classe `SAP_Authentication` faz a autenticação do usuário junto a *Wireless Village* e possui um método estático usado para identificar um usuário a partir de um *SessionID*. A classe `BD` possui os métodos necessários para

efetuar as consultas ao banco de dados. A classe `PhoneBookSearch` executa as consultas para o banco de dados referentes ao aplicativo de lista telefônica.

7.5 Implementação do Cliente

A implementação do cliente é composta por dois pacotes e uma classe que utiliza os recursos disponibilizados pelos dois pacotes.

A classe WVClient é a classe principal do MIDlet e é apresentada na Figura 7.4. Ela é a responsável por capturar as entradas do usuário, exibir as telas no *display* do celular de acordo com a escolha do usuário, iniciar e destruir o MIDlet, criar a conexão com o servidor, chamar o tradutor XML no pacote `lemom.wv.client.xml`, e chamar o serviço requisitado pelo cliente no pacote `lemom.wv.client.services`.

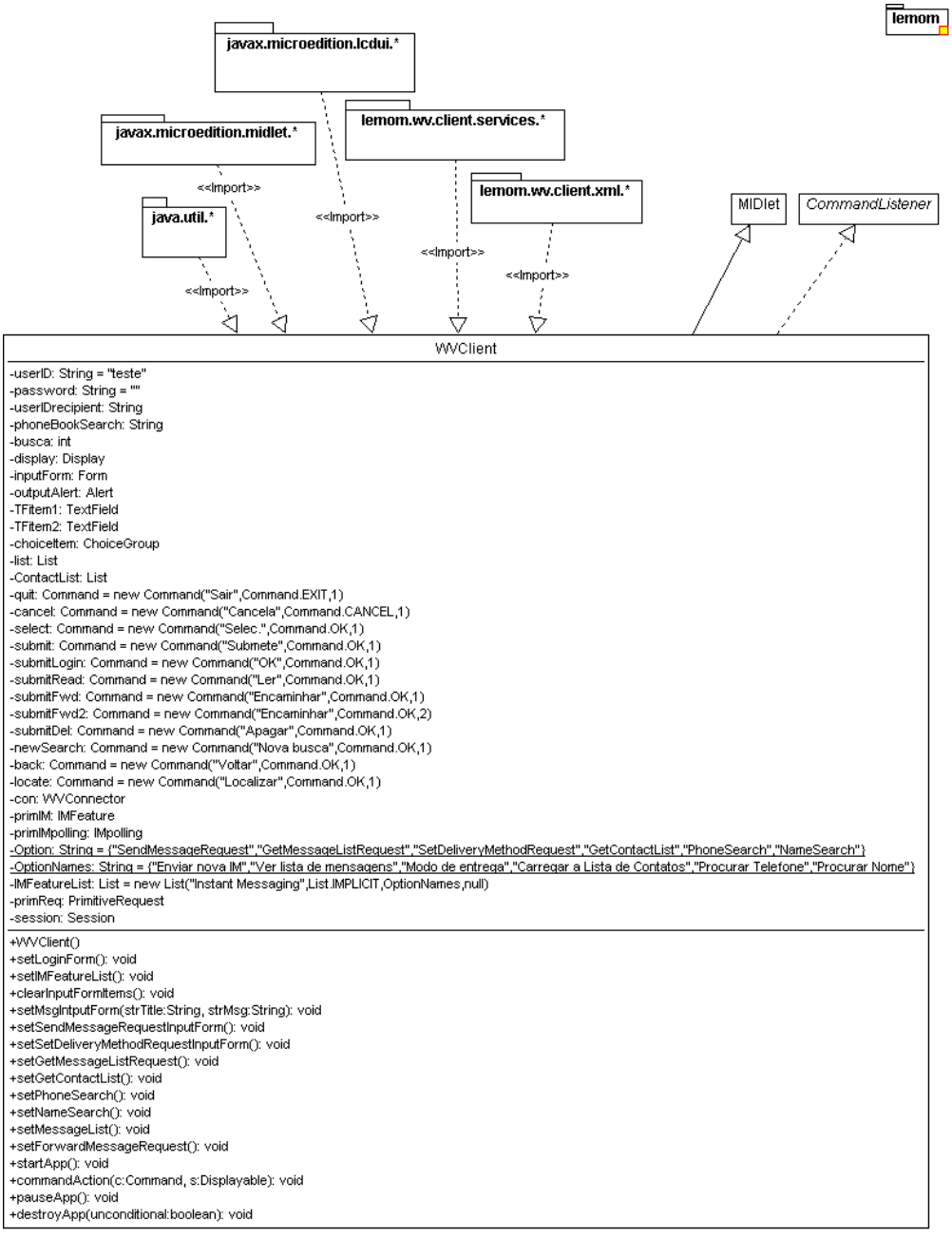

Figura 7.4: Diagrama de classe para WVClient.

O pacote lemom.wv.client.services oferece todos os serviços disponíveis para os usuários do Wireless Village e o diagrama de classes para este pacote pode ser visto na Figura 7.5.

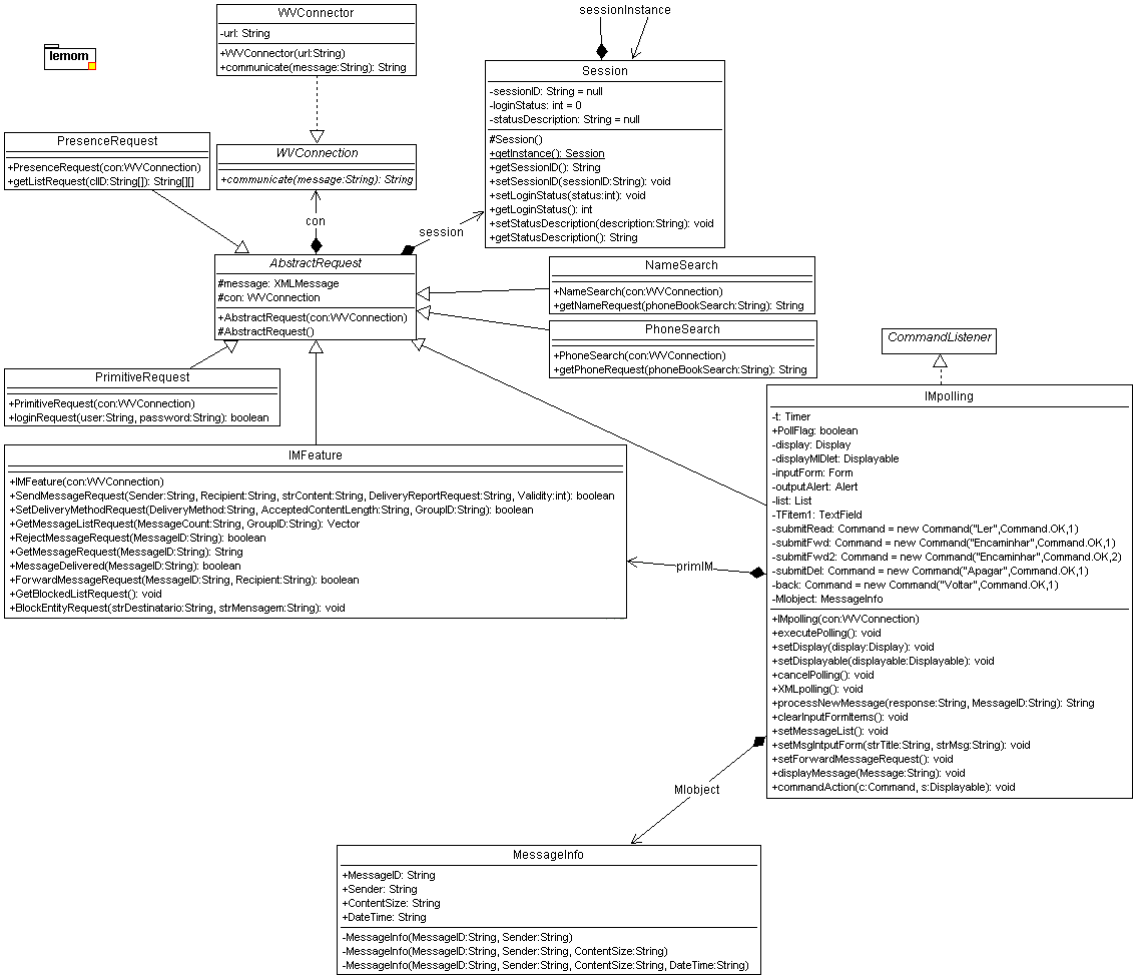

Figura 7.5: Diagrama de classes para o pacote `lemom.wv.client.services`.

As classes `WVConnector` e `WVConnection` efetuam as conexões entre o cliente e o servidor usando o protocolo HTTP. A classe `Session` é usada para garantir o *SessionID* do cliente. A classe `PrimitiveRequest` efetua todas transações relacionadas com a autenticação do usuário. A classes `IMFeature`, `IMPolling` e `MessageInfo` são usadas pelo serviço de mensagem instantânea. Por fim, as classes `NameSearch` e `PhoneSearch` efetuam toda lógica relacionada com o aplicativo *PhoneBook*. É possível se verificar a relação de herança e polimorfismo entre todas essas classes através do diagrama apresentado na figura acima.

Por fim, o pacote lemom.wv.client.xml é usado para se montar as mensagens XML que serão enviadas para o servidor Wireless Village e o diagrama de classe pode ser visto na Figura 7.6.

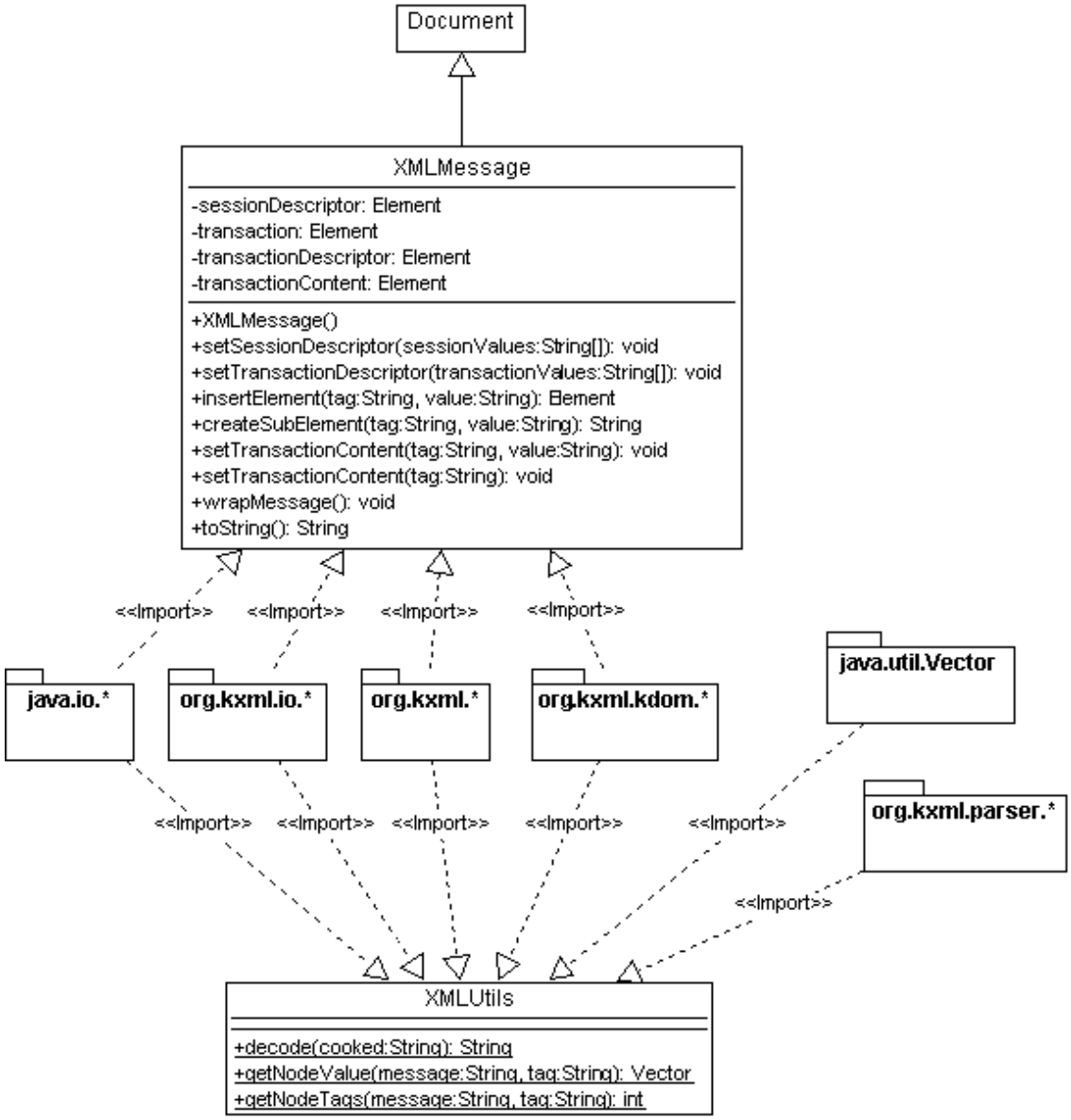

Figura 7.6: Diagrama de classes para o pacote `lemom.wv.client.xml`.

A classe `XMLMessage` é usada na montagem das mensagens que serão enviadas para o servidor, a montagem dessas mensagens segue a especificação *Wireless Village* para mensagens XML para os serviços de presença e mensagem instantânea e segue um padrão própria para troca de mensagens XML entre o cliente e o servido quando se utiliza o aplicativo PhoneBook. A classe `XMLUtil` possui três métodos estáticos que são usados na manipulação das respostas XML recebidas do servidor no qual a resposta vinda do servidor também segue a especificação *Wireless Village* para as respostas XML para os serviços de presença e mensagem instantânea e segue um padrão próprio para as respostas XML para o aplicativo *PhoneBook*.

7.6 Funcionamento do aplicativo

Neste trecho se estudará a maneira como o sistema funciona, ou seja, serão apresentadas as telas que aparecem para o usuário do sistema no momento de *login* do mesmo ao sistema *Wireless Village*, assim como as telas utilizadas na busca pelo nome

ou telefone de um outro usuário. O trecho referente à autenticação do cliente foi retirado, em partes, da referência [13].

7.6.1 Autenticação do Cliente

Para o sistema funcionar corretamente é necessário que o Servidor saiba identificar os Clientes que acessam o serviço. Como a identificação do Cliente junto ao Servidor é feita no ato da autenticação e, além disso, todas as transações entre Cliente e Servidor são efetuadas dentro de uma sessão que é identificada pelo *SessionID* obtido na autenticação, foi necessário acrescentar esta funcionalidade ao projeto.

A primeira tela do MIDlet é a seguinte:

Figura 7.7: Tela de login do sistema.

Onde o usuário entra com o seu *UserID*, que é seu identificador único na *Wireless Village*, e sua senha. Ao selecionar o comando *OK* a seguinte tela é apresentada:

Figura 7.8: Tela de espera da autenticação.

Neste momento o Cliente envia a mensagem (no padrão XML) *LoginRequest* ao Servidor.

Ao receber a *LoginRequest*, o Servidor identifica a mensagem, acessa o banco de dados e, caso o usuário tenha entrado com *userID* e senha válidos, autentica o Cliente. No caso de sucesso o Servidor envia uma mensagem de resposta (no padrão XML), *LoginResponse*, de código 200 (*Successfully logged in.*) contendo o *SessionID* que, de agora em diante, será usado para todas as transações entre Cliente e Servidor. Caso o

Cliente receba esta mensagem, *LoginResponse*, corretamente a seguinte tela é apresentada ao usuário:

Figura 7.9: Tela de login bem sucedido.

Esta primeira versão não trata os diferentes tipos de erro de autenticação de usuários. No caso de o Servidor não autenticar o usuário a mensagem de resposta, *LoginResponse*, será sempre de código 531 (*Unknown user.*).

Nenhuma das outras funcionalidades oferecidas pela implementação do *Wireless Village* será estudada em mais detalhe, pois o estudo deste servidor não é o enfoque deste trabalho.

7.6.2. Busca através do aplicativo PhoneBook

Como o aplicativo desenvolvido não faz parte da proposta *Wireless Village*, não foi necessária a utilização das especificações para estabelecer os critérios de troca de mensagens entre o cliente e o servidor, mas se tentou manter um foco similar para o nome dos pedidos e respostas trocados pelos sistemas em linguagem XML.

Esse aplicativo consiste na busca do telefone de uma pessoa qualquer a partir de seu nome ou da busca do nome de uma pessoa a partir de seu telefone. A primeira tela que o sistema apresenta para o usuário é a escolha do serviço ligado ao *Wireless Village* que ele quer usar, esta tela pode ser vista na Figura 7.10. Após essa escolha, as telas para a busca por nome ou telefone são bem análogas, onde a única diferença está na String que será utilizada na busca ao banco de dados, portanto no resultado apresentado, mas a forma para todo esse processo permanece a mesma para os dois casos.

Figura 7.10: Escolha do serviço no Wireless Village.

A Figura 7.11 apresenta a tela pela qual o usuário faz a entrada de um telefone no aplicativo para a busca do nome referente a esse telefone. A seguir, aparece uma tela de espera, enquanto a busca é feita no banco de dados, como pode ser visto na Figura 7.12.

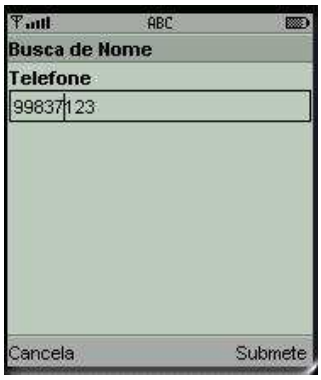

Figura 7.11: Busca por nome no aplicativo.

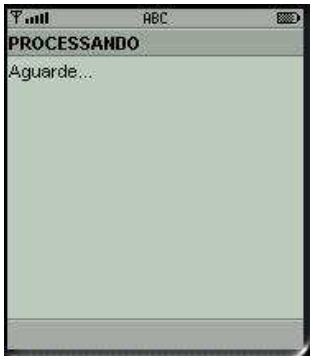

Figura 7.12: Tela de espera da busca.

Neste momento o cliente envia a mensagem (no padrão XML) *NameSearch* ao Servidor. Para o caso da busca de um telefone, seria enviada a mensagem *PhoneSearch*.

Ao receber a mensagem NameSearch, o servidor identifica a mensagem, acessa o banco de dados e, caso exista um resultado para o parâmetro usado no sistema, retorna esse resultado para o cliente. O resultado para uma busca por nome pode ser visto na Figura 7.13.

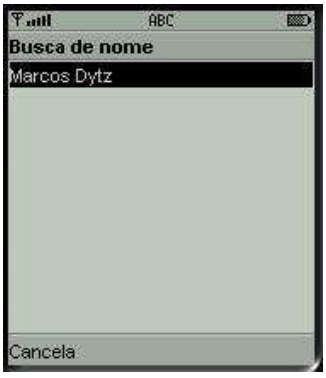

Figura 7.13: Resultado para uma busca por nome.

Caso o usuário não seja cadastrado na rede, uma resposta como a Figura 7.14 é enviada e, caso o usuário não permita que suas informações sejam enviadas para outros usuários, uma resposta como a Figura 7.15 é enviada.

Figura 7.14: Resposta para usuário não cadastrado ao sistema.

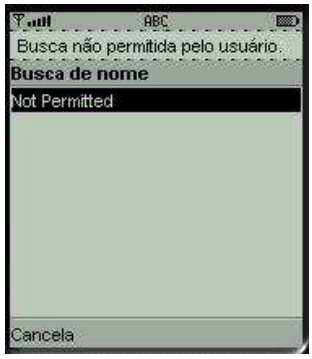

Figura 7.15: Resposta para usuário com "NotAllowed = 0".

CONCLUSÃO

Neste trabalho, começou-se por descrever todas tecnologias utilizadas para a implementação do sistema. Houve uma descrição dos principais elementos que norteiam a programação orientada a objetos e a linguagem Java. A seguir, se fez uma breve descrição do ambiente J2ME que foi utilizado na implementação da parte cliente do sistema. Os próximos temas abordados foram a tecnologia de *Servlets*, aplicativos-servidor, a linguagem de meta-informação XML e o sistema *Wireless Village*.

Partindo de todos essas tecnologias, então foi possível projetar o sistema a partir da interligação de todas essas tecnologias onde a linguagem Java e a linguagem XML foram os pilares que possibilitaram a troca de informação entre entidades tão diferentes como aparelhos móveis, aplicativos para servidor e banco de dados.

O programa apresentado permite a simulação de uma lista telefônica direta/reversa usando o sistema *Wireless Village*, ou seja, o usuário a partir de um aparelho móvel pode entrar com um telefone ou nome a ser procurado no sistema que retornaria o nome ou telefone da pessoa procurada, respectivamente. O usuário do qual a informação está sendo procurada teria o direito de não permitir que suas informações sejam rastreáveis, assim não invadindo a privacidade de qualquer usuário do sistema.

Com base no aplicativo desenvolvido, foram feitos testes para verificar a viabilidade e o funcionamento do sistema e o mesmo funcionou da maneira desejada e não se encontrou nenhuma anomalia nesse ponto, ou seja, a integração do aplicativo desenvolvido com o sistema *Wireless Village* ocorreu perfeitamente e todos os módulos deste servidor funcionaram da maneira projetada.

A implementação anexada neste trabalho apresenta todo o código-fonte Java desenvolvido para o projeto e os *scripts* SQL que criam as duas tabelas necessárias para o funcionamento do sistema. Os diagramas UML apresentam o comportamento do sistema exatamente como esse comportamento é visto no sistema.

Dentre as perspectivas para trabalhos futuros relacionados ao *Wireless Village Server* pode-se mencionar:

1. Implementação dos serviços que restam ao *Wireless Village Server*, ou seja, o serviço de grupo e o serviço de compartilhamento de conteúdo;
2. Implementação de novos aplicativos que utilizem o Wireless Village como sua base, assim gerando um maior valor para o sistema;
3. Transferência dos arquivos JAR e JAD referentes ao cliente do sistema a um aparelho móvel;
4. Utilização de uma interface gráfica mais sofisticada no aparelho móvel onde elementos coloridos e imagens gráficas poderiam ser utilizadas.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] MCLAUGHLIN B., *Java & XML*, O'Reilly, 2001

[2] HARKEY D., APPAJODU S., LARKIN, M., *Wireless Java Programming for Enterprise Applications*, Wiley Publishing Inc., 2002

[3] ASHRI R., ATKINSON S., AYERS D., HAGLIND M., RAY B., MACHIN R., NASHI N., TAYLOR R., WIGGERS C., *Java Mobile Programming*, Wrox Press, 2001

[4] KNUDSEN J., *Wireless Java: Developing with Java 2, Micro Edition*, Apress, 2001

[5] MCLAUGHLIN B., *Java & XML Data Binding*, O'reilly, 2002

[6] STALLINGS W., *Wireless Communications and Networks*, Prentice Hall, 2001

[7] STEELE R., LEE C. C., GOULD P., *GSM, cdmaOne and 3G Systems*, John Wiley & Sons Ltd., 2001

[8] LARMAN C., *Utilizando UML e padrões*, Bookman, 2000

[9] DEITEL H.M., DEITEL P. J., SANTRY S. E., *Advanced Java 2 Platform – How to Program*, Prentice Hall, 2001

[10] HORSTMANN C. S. , CORNELL G., *Core Java 2, Volume 1 – Fundamentos*, Makron Books, 2001

[11] HORSTMANN C. S., CORNELL G., *Core Java 2, Volume 2 – Recursos Avançados*, Makron Books, 2001

[12] OLIVEIRA A. H. C., CORREIA H. B., IIDA R. F., *Implementação de uma interface neutra cliente-servidor dedicada à simulação e dos módulos de sistemas móveis e circuitos elétricos*, Dissertação de Projeto Final de Graduação da Universidade de Brasília, 2002

[13] OLIVEIRA A. F., CARVALHO L. A., *Instant Messaging para Wireless Village em J2ME*, Dissertação de Projeto Final de Graduação da Universidade de Brasília, 2003

[14] CUSSIOLI T. J. S., RIBEIRO FILHO W., *Aplicações utilizando sistemas de localização em GSM*, Dissertação de Projeto Final de Graduação da Universidade de Brasília, 2003

[15] *Programação Orientada a Objeto*, http://www.unifieo.br/revista/rev1999/ProgOO.htm

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

[17] *XML 1.0 Recommendation*, http://www.w3.org/TR/REC-xml

[18] *JDBC Data Access API*, http://java.sun.com/products/jdbc/related.html

[19] *Wireless Java*, http://wireless.java.sun.com/

[20] ROUSELL T., *Java Technology: The Kingpin of Mobile Monetization.”*, http://wireless.java.sun.com/developers/business/articles/kingpin/

[21] http://www.openmobilealliance.org/wirelessvillage/

[22] *[WV] Wireless Village – System Architecture Model*, 2002, http://www.openmobilealliance.org/wirelessvillage/docs/WV_Architecture_v1.1.pdf

[23] *[WV-FF] Wireless Village – Features and Functions specification*, 2002, http://www.openmobilealliance.org/wirelessvillage/docs/WV_Features_Functions_v1.1.pdf

[24] *[WV-CSP] Wireless Village – Client-Server Protocol, Session and Transactions*, 2002, http://www.openmobilealliance.org/wirelessvillage/docs/WV_CSP_v1.1.pdf

[25] *[WV-PA] Wireless Village – Presence Attributes specification*, 2002, http://www.openmobilealliance.org/wirelessvillage/docs/WV_PA_v1.1.pdf

[26] *[WV-PA-DTD] Wireless Village – Presence Attributes, DTD and Examples*, 2002, http://www.openmobilealliance.org/wirelessvillage/docs/WV_PA_DTD_v1.1.pdf

[27] http://www.w3c.org

[28] *MIDP Datasheet*, 2003, http://java.sun.com/products/midp/midp-ds.pdf

[29] *J2ME Datasheet*, 2003, http://java.sun.com/j2me/j2me-ds.pdf

[30] VAN PEURSEM J., *Mobile Information Device Profile v2.0 (JSR-118) Final Specification*, 2003, http://jcp.org/aboutJava/communityprocess/final/jsr118/

[31] TAIVALSAARI A., *CLDC Specification V1.1 (JSR-139) Final Specification*, 2003, http://jcp.org/aboutJava/communityprocess/final/jsr139/index.html

APÊNDICES

Apêndice A: PhoneBook.sql

```
/*****************************************************************************
 *
 * PhoneBook.sql – LEMOM - UnB
 *
 *        Cria a base de dados que será utilizada para o aplicativo de lista
 *  telefônica e habita essa tabela com alguns valores hipotéticos
 *
*****************************************************************************/

# CREATE DATABASE wv;

CREATE TABLE wv.PhoneBook (
                        Serial VARCHAR(20) PRIMARY KEY NOT NULL,
                        Name VARCHAR(30) NOT NULL,
                        Phone VARCHAR(10) NOT NULL UNIQUE,
                        AllowSearch ENUM('0', '1'),
                        Last_update TIMESTAMP NULL);

CREATE INDEX user_idx ON wv.PhoneBook (Name);

INSERT INTO wv.PhoneBook (Serial, Name, Phone, AllowSearch) VALUES ("123456789", "Eduardo Sirqueira", "99823045", '1');

INSERT INTO wv.PhoneBook (Serial, Name, Phone, AllowSearch) VALUES ("123456788", "Tatiana Martins", "99611313", '1');

INSERT INTO wv.PhoneBook (Serial, Name, Phone, AllowSearch) VALUES ("123456787", "Daniel Almeida", "99758412", '1');

INSERT INTO wv.PhoneBook (Serial, Name, Phone, AllowSearch) VALUES ("123456786", "Rafael Moraes", "96268814", '0');

INSERT INTO wv.PhoneBook (Serial, Name, Phone, AllowSearch) VALUES ("123456785", "Antonio da Silva", "96198652", '1');

INSERT INTO wv.PhoneBook (Serial, Name, Phone, AllowSearch) VALUES ("123456784", "Izabela Goes", "9237410", '0');

INSERT INTO wv.PhoneBook (Serial, Name, Phone, AllowSearch) VALUES ("123456783", "Marcela Santos", "9968810", '0');

INSERT INTO wv.PhoneBook (Serial, Name, Phone, AllowSearch) VALUES ("123456782", "Ana Paula Ferreira", "9215697", '1');

INSERT INTO wv.PhoneBook (Serial, Name, Phone, AllowSearch) VALUES ("123456781", "Joao Oliveira", "99884103", '1');

INSERT INTO wv.PhoneBook (Serial, Name, Phone, AllowSearch) VALUES ("123456780", "Barbara Valente", "96984520", 1);

DESCRIBE wv.PhoneBook;

SELECT * FROM wv.PhoneBook;
```

Apêndice B: User.sql

```
/**************************************************************************
 *
 * User.sql - LEMOM - UnB
 *
 *         Cria a base de dados que será utilizada para o login ao sistema e
 *  habita essa tabela com alguns valores hipotéticos
 *
 **************************************************************************/

CREATE DATABASE wv;

CREATE TABLE wv.User (
                        ID TINYINT(3) UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
                        UserID VARCHAR(20)  NOT NULL UNIQUE,
                        Password VARCHAR(20) NOT NULL,
                        SessionID VARCHAR(50) NOT NULL UNIQUE,
                        DeliveryMethod ENUM('P', 'N') NOT NULL DEFAULT('N'),
                        AcceptedContentLength MEDIUMINT(3) UNSIGNED NOT NULL DEFAULT '0',
                        Serial VARCHAR(10) NOT NULL UNIQUE,
                        MIN VARCHAR(10) NOT NULL UNIQUE,
                        Last_update TIMESTAMP NULL);

CREATE INDEX login_idx ON wv.User (Login);

INSERT INTO wv.User (UserID, Password, SessionID, DeliveryMethod, AcceptedContentLength, Serial,
MIN) VALUES ("teste", "teste", "wv.usr.com#9999999@server.com", "N", "2048", "1234567", "9999999");

INSERT INTO wv.User (UserID, Password, SessionID, DeliveryMethod, AcceptedContentLength, Serial,
MIN) VALUES ("marcos", "marcos", "wv.usr.com#9998888@server.com", "P", "1024", "1234566",
"9998888");

INSERT INTO wv.User (UserID, Password, SessionID, DeliveryMethod, AcceptedContentLength, Serial,
MIN) VALUES ("lemom", "lemom", "wv.usr.com#9997777@server.com", "N", "512", "1234565",
"9997777");

INSERT INTO wv.User (UserID, Password, SessionID, DeliveryMethod, AcceptedContentLength, Serial,
MIN) VALUES ("admin", "admin", "wv.usr.com#9996666@server.com", "P", "4096", "1234564",
"9996666");

INSERT INTO wv.User (UserID, Password, SessionID, DeliveryMethod, AcceptedContentLength, Serial,
MIN) VALUES ("suporte", "suporte", "wv.usr.com#9995555@server.com", "N", "2048", "1234563",
"9995555");

DESCRIBE wv.User;

SELECT * FROM wv.User;
```

Apêndice C: Sap_CSPservlet.java

```
/*
 * @ 2003 - Lemom - UnB
 *
 * Servlet utilizado pelo servidor Wireless Village para capturar todos pedidos
 * vindos de aparelhos celulares
 */

import java.io.IOException;
import java.io.StringWriter;
import java.rmi.RemoteException;
import java.sql.SQLException;
import java.util.Vector;

import javax.servlet.ServletException;
import javax.servlet.ServletInputStream;
import javax.servlet.ServletOutputStream;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;

/**
 *
 * @author Lemom
 *
 * This class provides all the Servlet capabilities needed by the Wireless Village
 * clients
 */
public class SAP_CSPservlet extends HttpServlet {

    Document messageDOC;
    String messageXML;
    boolean isInband;
    boolean isRequest;
    String transactionID;
    String sessionID;

    BD bd;

/**
 * General constructor for this class
 *
 */
    public SAP_CSPservlet() {}
```

```
/**
 * Class responsible for initializing the Servlet
 *
 * @throws ServletException
 */
   public void init() throws ServletException {

         bd = new BD();
         bd.Conexao();
   }

/**
 * Method that will take care of all HTTP requests and responses
 *
 * @param httpservletrequest as <code>HttpServletRequest</code>
 * @param httpservletresponse as <code>HttpServletResponse</code>
 */
   public void doPost(HttpServletRequest httpservletrequest, HttpServletResponse httpservletresponse) {

         try {
            ServletInputStream servletinputstream = httpservletrequest.getInputStream();
            DocumentBuilderFactory documentbuilderfactory = DocumentBuilderFactory.newInstance();
            DocumentBuilder documentbuilder = documentbuilderfactory.newDocumentBuilder();
            messageDOC = documentbuilder.parse(servletinputstream);
            pre_processDOC();
            processDOC();
            TransformerFactory transformerfactory = TransformerFactory.newInstance();
            Transformer transformer = transformerfactory.newTransformer();
            transformer.setOutputProperty("omit-xml-declaration", "yes");
            transformer.setOutputProperty("indent", "yes");
            StringWriter stringwriter = new StringWriter();
            StreamResult streamresult = new StreamResult(stringwriter);
            DOMSource domsource = new DOMSource(messageDOC);
            transformer.transform(domsource, streamresult);
            messageXML = stringwriter.getBuffer().toString();
            httpservletresponse.setContentType("text/xml");
            ServletOutputStream servletoutputstream = httpservletresponse.getOutputStream();
            servletoutputstream.print(messageXML);
            servletinputstream.close();
            servletoutputstream.close();
         }
         catch(SAXParseException saxparseexception) { }
         catch(ParserConfigurationException parserconfigurationexception) { }
         catch(TransformerException transformerexception) { }
         catch(SAXException saxexception) { }
         catch(IOException ioexception) { }
   }

/**
 * Method that will be used when the Servlet is going to be destroyed
 *
 */
   public void destroy() {}

/**
 * Method used for pre processing the incoming request for the Servlet
 *
 */
   public void pre_processDOC() {
```

```
      NodeList nodelist = messageDOC.getElementsByTagName("SessionDescriptor");
      for(int i = 0; i < nodelist.getLength(); i++) {
         Element element = (Element)nodelist.item(i);
                Element sessiontype = (Element)element.getElementsByTagName("SessionType").item(0);
                String inband = " ";
                if (sessiontype.getFirstChild().getNodeValue() != null)
                   inband = sessiontype.getFirstChild().getNodeValue();
                if(inband.equalsIgnoreCase("Inband")) {
                   isInband = true;
                   Element sessionid = (Element)element.getElementsByTagName("SessionID").item(0);
                   sessionID = sessionid.getFirstChild().getNodeValue();
                } else if(inband.equalsIgnoreCase("Outband")) {
                   isInband = false;
                   sessionID = null;
                }
          }
    }

 /**
  * This is the most importante method of the Servlet. This method is responsible
  * for calling the method that will take care of the request according with the
  * XML tags used
  *
  */
    public void processDOC() {

      NodeList nodelist = messageDOC.getElementsByTagName("Transaction");
      for(int i = 0; i < nodelist.getLength(); i++) {
         Element element = (Element)nodelist.item(i);
         Element transactiondescriptor = (Element)
                element.getElementsByTagName("TransactionDescriptor").item(0);
         Element transactionmode = (Element)
                transactiondescriptor.getElementsByTagName("TransactionMode").item(0);
         if(transactionmode.getFirstChild().getNodeValue().equals("Request")) {
            org.w3c.dom.Text text = messageDOC.createTextNode("Response");
            transactionmode.removeChild(transactionmode.getFirstChild());
            transactionmode.appendChild(text);
         }
         Element transactioncontent = (Element)
                element.getElementsByTagName("TransactionContent").item(0);
         Element elementname = (Element) transactioncontent.getElementsByTagName("*").item(0);
         String tagname = elementname.getTagName();
         Element checkmessage = null;

         if(tagname.equalsIgnoreCase("Login-Request"))
           checkmessage = (Element)processLoginRequest(elementname);

         if(tagname.equalsIgnoreCase("Polling-Request"))
           checkmessage = (Element)processPollingRequest(elementname);

         if(tagname.equalsIgnoreCase("SendMessage-Request"))
           checkmessage = (Element)processSendMessageRequest(elementname);

         if(tagname.equalsIgnoreCase("SetDeliveryMethod-Request"))
           checkmessage = (Element)processSetDeliveryMethodRequest(elementname);

         if(tagname.equalsIgnoreCase("GetMessageList-Request"))
           checkmessage = (Element)processGetMessageListRequest(elementname);
```

```
            if(tagname.equalsIgnoreCase("GetMessage-Request"))
              checkmessage = (Element)processGetMessageRequest(elementname);

            if(tagname.equalsIgnoreCase("MessageDelivered"))
              checkmessage = (Element)processMessageDelivered(elementname);

            if(tagname.equalsIgnoreCase("ForwardMessage-Request"))
              checkmessage = (Element)processForwardMessageRequest(elementname);

            if(tagname.equalsIgnoreCase("RejectMessage-Request"))
              checkmessage = (Element)processRejectMessageRequest(elementname);

            if(tagname.equalsIgnoreCase("ListManage-Request"))
              checkmessage = (Element)processListManageRequest(elementname);

            if(tagname.equalsIgnoreCase("GetPresence-Request"))
              checkmessage = (Element)processGetPresenceRequest(elementname);

            if(tagname.equalsIgnoreCase("GetPhone-Request"))
              checkmessage = (Element)processGetPhoneRequest(elementname);

            if(tagname.equalsIgnoreCase("GetName-Request"))
              checkmessage = (Element)processGetNameRequest(elementname);

            if (checkmessage != null)
              transactionmode.appendChild(checkmessage);
            transactioncontent.removeChild(elementname);
        }
    }

/**
 * This method will take care of the login process
 *
 * @param element as <code>Element</code>
 * @return Node
 */
    public Node processLoginRequest(Element element) {

        Element element1 = (Element)element.getElementsByTagName("UserID").item(0);
        Element element2 = (Element)element.getElementsByTagName("Password").item(0);
        Element element3 = (Element)element.getElementsByTagName("URL").item(0);
        String s = element1.getFirstChild().getNodeValue();
        String s1 = element2.getFirstChild().getNodeValue();
        String s2 = element3.getFirstChild().getNodeValue();
        SAP_Authentication sap_authentication = new SAP_Authentication();
        String s3 = null;
        Element element4 = messageDOC.createElement("Login-Response");
        Element element7, element8, element9;
        org.w3c.dom.Text text1,text2;

        if(sap_authentication.checkLogin(s, s1)) {
                Element element5 = messageDOC.createElement("ClientID");
                Element element6 = messageDOC.createElement("URL");
                org.w3c.dom.Text text =
                    messageDOC.createTextNode("http://206.226.10.25:80/IMPSAPP");
                element6.appendChild(text);
                element5.appendChild(element6);
                element4.appendChild(element5);
                element7 = messageDOC.createElement("Result");
                element8 = messageDOC.createElement("Code");
```

```
                text1 = messageDOC.createTextNode("200");
                element8.appendChild(text1);
                element9 = messageDOC.createElement("Description");
                text2 = messageDOC.createTextNode("Successfully logged in.");
                element9.appendChild(text2);
                Element element10 = messageDOC.createElement("SessionID");
                org.w3c.dom.Text text3 =
                    messageDOC.createTextNode(sap_authentication.getSessionID());
                element10.appendChild(text3);
                element4.appendChild(element10);
                Element element11 = messageDOC.createElement("KeepAliveTime");
                org.w3c.dom.Text text4 = messageDOC.createTextNode("120");
                element11.appendChild(text4);
                element4.appendChild(element11);
                Element element12 = messageDOC.createElement("CapabilityRequest");
                org.w3c.dom.Text text5 = messageDOC.createTextNode("T");
                element12.appendChild(text5);
                element4.appendChild(element12);
              } else {
                 element7 = messageDOC.createElement("Result");
                 element8 = messageDOC.createElement("Code");
                 text1 = messageDOC.createTextNode("531");
                 element8.appendChild(text1);
                 element9 = messageDOC.createElement("Description");
                 text2 = messageDOC.createTextNode("Failed login.");
                 element9.appendChild(text2);
              }
      element7.appendChild(element8);
      element7.appendChild(element9);
      element4.appendChild(element7);
      return element4;
    }

/**
 * This method will take care of the polling for the IM service
 *
 * @param element as <code>Element</code>
 * @return Node
 */
    public Node processPollingRequest(Element element) {

      Element element1 = null;
      String UserID = SAP_Authentication.getUserID(sessionID);
      int NMensagens = 0;
      try {
                String csql;
                csql = "SELECT COUNT(MessageID) as NMensagens FROM message WHERE
Notified='N' AND Recipient = '"+ UserID.replaceAll("'", "''") +"'";
                bd.setCsql(csql);
                bd.setLer (bd.Consulta(bd.getCsql()));
                NMensagens = bd.getLer().getInt("NMensagens");
      } catch (SQLException e) {}

      if (NMensagens > 0) {
                String DeliveryMethod = "";
                try {
                        String csql;
                        csql = "SELECT DeliveryMethod FROM User WHERE UserID = '"+
UserID.replaceAll("'", "''") +"'";
                        bd.setCsql(csql);
```

```
                    bd.setLer (bd.Consulta(bd.getCsql()));
                    DeliveryMethod = bd.getLer().getString("DeliveryMethod");
               } catch (SQLException e) {}
               try  {
                    String csql;
                    csql = "SELECT * FROM message WHERE Notified='N' AND Recipient = '"+
UserID.replaceAll("'", "''") +"' ORDER BY MessageID";
                    bd.setCsql(csql);
                    bd.setLer (bd.Consulta(bd.getCsql()));
                    if (DeliveryMethod.equalsIgnoreCase("N")) {
                        element1 = messageDOC.createElement("MessageNotification");
                    } else if (DeliveryMethod.equalsIgnoreCase("P")) {
                        element1 = messageDOC.createElement("NewMessage");
                        Element element10 = messageDOC.createElement("ContentData");
                        org.w3c.dom.Text text10 =
                             messageDOC.createTextNode(bd.getLer().getString("ContentData"));
                        element10.appendChild(text10);
                        element1.appendChild(element10);
                    }
                    Element element2 = messageDOC.createElement("MessageInfo");
                    Element element3 = messageDOC.createElement("MessageID");
                    org.w3c.dom.Text text3 =
                         messageDOC.createTextNode(bd.getLer().getString("MessageID"));
                    element3.appendChild(text3);
                    element2.appendChild(element3);
                    Element element4 = messageDOC.createElement("ContentType");
                    org.w3c.dom.Text text4 =
                         messageDOC.createTextNode(bd.getLer().getString("ContentType"));
                    element4.appendChild(text4);
                    element2.appendChild(element4);
                    Element element5 = messageDOC.createElement("ContentEncoding");
                    org.w3c.dom.Text text5 =
                         messageDOC.createTextNode(bd.getLer().getString("ContentEncoding"));
                    element5.appendChild(text5);
                    element2.appendChild(element5);
                    Element element55 = messageDOC.createElement("ContentSize");
                    org.w3c.dom.Text text55 =
                         messageDOC.createTextNode(bd.getLer().getString("ContentSize"));
                    element55.appendChild(text55);
                    element2.appendChild(element55);
                    Element element6 = messageDOC.createElement("Recipient");
                    Element element601 = messageDOC.createElement("User");
                    Element element60101 = messageDOC.createElement("UserID");
                    org.w3c.dom.Text text60101 =
                         messageDOC.createTextNode(bd.getLer().getString("Recipient"));
                    element60101.appendChild(text60101);
                    element601.appendChild(element60101);
                    element6.appendChild(element601);
                    element2.appendChild(element6);
                    Element element7 = messageDOC.createElement("Sender");
                    Element element701 = messageDOC.createElement("User");
                    Element element70101 = messageDOC.createElement("UserID");
                    org.w3c.dom.Text text70101 =
                         messageDOC.createTextNode(bd.getLer().getString("Sender"));
                    element70101.appendChild(text70101);
                    element701.appendChild(element70101);
                    element7.appendChild(element701);
                    element2.appendChild(element7);
                    Element element8 = messageDOC.createElement("DateTime");
```

```
                        org.w3c.dom.Text text8 =
                            messageDOC.createTextNode(bd.getLer().getString("DateTime"));
                        element8.appendChild(text8);
                        element2.appendChild(element8);
                        Element element9 = messageDOC.createElement("Validity");
                        org.w3c.dom.Text text9 =
                            messageDOC.createTextNode(bd.getLer().getString("Validity"));
                        element9.appendChild(text9);
                        element2.appendChild(element9);
                        element1.appendChild(element2);
                        csql = "UPDATE message SET Notified='Y' WHERE MessageID=" +
bd.getLer().getInt("MessageID");
                        bd.setCsql(csql);
                        int x;
                        x = bd.Atualizar(bd.getCsql());
                    } catch (SQLException e) {}
            } else {
            element1 = messageDOC.createElement("Polling-Response");
            Element element3 = messageDOC.createElement("Result");
            Element element4 = messageDOC.createElement("Code");
            org.w3c.dom.Text text = messageDOC.createTextNode("200");
            element4.appendChild(text);
            element3.appendChild(element4);
            element1.appendChild(element3);
        }
        return element1;
    }

/**
 * This method will be used when a message needs to be sent
 *
 * @param element as <code>Element</code>
 * @return Node
 */
    public Node processSendMessageRequest(Element element) {

            MessageInfo MIobject = new MessageInfo();
            Element elementRequested1 = (Element)
               element.getElementsByTagName("MessageInfo").item(0);
            Element elementRequested2 = (Element)
               elementRequested1.getElementsByTagName("Sender").item(0);
            elementRequested1 = (Element)elementRequested2.getElementsByTagName("User").item(0);
            elementRequested2 = (Element)elementRequested1.getElementsByTagName("UserID").item(0);
            MIobject.Sender = SAP_Authentication.getUserID(sessionID);
            elementRequested1 = (Element)element.getElementsByTagName("MessageInfo").item(0);
            elementRequested2 = (Element)
               elementRequested1.getElementsByTagName("Recipient").item(0);
            elementRequested1 = (Element)elementRequested2.getElementsByTagName("User").item(0);
            elementRequested2 = (Element)elementRequested1.getElementsByTagName("UserID").item(0);
            MIobject.Recipient = elementRequested2.getFirstChild().getNodeValue();
            elementRequested1 = (Element)element.getElementsByTagName("MessageInfo").item(0);
              elementRequested2 = (Element)
                 elementRequested1.getElementsByTagName("DateTime").item(0);
            MIobject.DateTime = (new java.util.Date()).toString();
            Element elementRequested = (Element)
               element.getElementsByTagName("ContentData").item(0);
             MIobject.ContentData = elementRequested.getFirstChild().getNodeValue();
             MIobject.ContentSize = String.valueOf((MIobject.ContentData).length());

        try {
```

```
                String csql;
                csql = "SELECT MAX(MessageID) as MaxMessageID FROM message";
                        bd.setCsql(csql);
                bd.setLer (bd.Consulta(bd.getCsql()));
                MIobject.MessageID = String.valueOf(bd.getLer().getInt("MaxMessageID") + 1);
                csql = "INSERT INTO message
(MessageID,Sender,Recipient,DateTime,ContentSize,ContentEncoding,ContentType,Validity,ContentDat
a) ";
                csql = csql +  "VALUES ("+ Integer.parseInt((String)MIobject.MessageID) + ",'"+
MIobject.Sender.replaceAll("'", "''") +"','"+ MIobject.Recipient.replaceAll("'", "''") +"','"+
MIobject.DateTime.replaceAll("'", "''") +"','"+ MIobject.ContentSize.replaceAll("'", "''") +"','"+
MIobject.ContentEncoding.replaceAll("'", "''") +"','"+ MIobject.ContentType.replaceAll("'", "''") +"','"+
MIobject.Validity.replaceAll("'", "''") +"','"+ MIobject.ContentData.replaceAll("'", "''") + "')";
                        bd.setCsql(csql);
                int x;
                x = bd.Inserir(bd.getCsql());
        } catch (SQLException e) {}

        Element element2;
        element2 = messageDOC.createElement("SendMessage-Response");
        Element element3 = messageDOC.createElement("Result");
        Element element4 = messageDOC.createElement("Code");
        org.w3c.dom.Text text = messageDOC.createTextNode("200");
        element4.appendChild(text);
        Element element5 = messageDOC.createElement("Description");
        org.w3c.dom.Text text1 = messageDOC.createTextNode("Succesfully completed.");
        element5.appendChild(text1);
        Element element6 = messageDOC.createElement("MessageID");
        org.w3c.dom.Text text2 = messageDOC.createTextNode(MIobject.MessageID);
        element6.appendChild(text2);
        element3.appendChild(element4);
        element3.appendChild(element5);
        element2.appendChild(element3);
        element2.appendChild(element6);
        return element2;
    }

/**
 * This method will be used to set the delivery method
 *
 * @param element as <code>Element</code>
 * @return Node
 */
    public Node processSetDeliveryMethodRequest(Element element) {

        Element elementRequested = (Element)
          element.getElementsByTagName("DeliveryMethod").item(0);
        String s = elementRequested.getFirstChild().getNodeValue();
        Element elementRequested2 = (Element)
          element.getElementsByTagName("AcceptedContentLength").item(0);
        String s2 = elementRequested2.getFirstChild().getNodeValue();
        try {
            String csql;
            csql = "UPDATE User SET DeliveryMethod= '"+ s.replaceAll("'", "''") +"',
AcceptedContentLength = " + s2 + " WHERE UserID='" +
SAP_Authentication.getUserID(sessionID).replaceAll("'", "''") + "'";
            bd.setCsql(csql);
            int x;
            x= bd.Atualizar(bd.getCsql());
        } catch (SQLException e) {}
```

```
        Element element2 = messageDOC.createElement("Status");
        Element element3 = messageDOC.createElement("Result");
        Element element4 = messageDOC.createElement("Code");
        org.w3c.dom.Text text = messageDOC.createTextNode("200");
        element4.appendChild(text);
        Element element5 = messageDOC.createElement("Description");
        org.w3c.dom.Text text1 = messageDOC.createTextNode("Succesfully completed.");
        element5.appendChild(text1);
        element3.appendChild(element4);
        element3.appendChild(element5);
        element2.appendChild(element3);
         return element2;
    }

/**
 * This method will be used to get all the messages that a WV client has
 *
 * @param element as <code>Element</code>
 * @return Node
 */
    public Node processGetMessageListRequest(Element element) {

        Element elementRequested = (Element)
          element.getElementsByTagName("MessageCount").item(0);
        String s = elementRequested.getFirstChild().getNodeValue();
        int p = Integer.parseInt(s);
        int n = 0;
        String Recipient = SAP_Authentication.getUserID(sessionID);
        Element element1;
        element1 = messageDOC.createElement("GetMessageList-Response");
        Element element2 = null;
        try {
            String csql;
            csql = "SELECT COUNT(MessageID) as contador FROM message WHERE Recipient = ''' +
Recipient.replaceAll("'", "''") + "'";
            bd.setCsql(csql);
            bd.setLer (bd.Consulta(bd.getCsql()));
            if (bd.getLer().getInt("contador") > p)
                n = p;
            else
                n = bd.getLer().getInt("contador");
            csql = "SELECT * FROM message WHERE Recipient = ''' + Recipient.replaceAll("'", "''") + '''
ORDER BY MessageID";
            bd.setCsql(csql);
            bd.setLer (bd.Consulta(bd.getCsql()));
            Vector vector = new Vector(n, 5);
            for (int i = 0;i<n; i++) {
                bd.getLer().next();
                element2 = messageDOC.createElement("MessageInfo");
                Element element3 = messageDOC.createElement("MessageID");
                org.w3c.dom.Text text3 =
                  messageDOC.createTextNode(bd.getLer().getString("MessageID"));
                element3.appendChild(text3);
                element2.appendChild(element3);
                Element element4 = messageDOC.createElement("ContentType");
                org.w3c.dom.Text text4 =
                  messageDOC.createTextNode(bd.getLer().getString("ContentType"));
                element4.appendChild(text4);
                element2.appendChild(element4);
```

```
                Element element5 = messageDOC.createElement("ContentEncoding");
                org.w3c.dom.Text text5 =
                  messageDOC.createTextNode(bd.getLer().getString("ContentEncoding"));
                element5.appendChild(text5);
                element2.appendChild(element5);
                Element element55 = messageDOC.createElement("ContentSize");
                org.w3c.dom.Text text55 =
                  messageDOC.createTextNode(bd.getLer().getString("ContentSize"));
                element55.appendChild(text55);
                element2.appendChild(element55);
                Element element6 = messageDOC.createElement("Recipient");
                Element element601 = messageDOC.createElement("User");
                Element element60101 = messageDOC.createElement("UserID");
                org.w3c.dom.Text text60101 =
                  messageDOC.createTextNode(bd.getLer().getString("Recipient"));
                element60101.appendChild(text60101);
                element601.appendChild(element60101);
                element6.appendChild(element601);
                element2.appendChild(element6);
                Element element7 = messageDOC.createElement("Sender");
                Element element701 = messageDOC.createElement("User");
                Element element70101 = messageDOC.createElement("UserID");
                org.w3c.dom.Text text70101 =
                  messageDOC.createTextNode(bd.getLer().getString("Sender"));
                element70101.appendChild(text70101);
                element701.appendChild(element70101);
                element7.appendChild(element701);
                element2.appendChild(element7);
                Element element8 = messageDOC.createElement("DateTime");
                org.w3c.dom.Text text8 =
                  messageDOC.createTextNode(bd.getLer().getString("DateTime"));
                element8.appendChild(text8);
                element2.appendChild(element8);
                Element element9 = messageDOC.createElement("Validity");
                org.w3c.dom.Text text9 = messageDOC.createTextNode(bd.getLer().getString("Validity"));
                element9.appendChild(text9);
                element2.appendChild(element9);
                element1.appendChild(element2);
                vector.addElement(bd.getLer().getString("MessageID"));
            }
            for (int i = 0;i<n; i++) {
                csql = "UPDATE message SET Notified='Y' WHERE MessageID=" + vector.elementAt(i);
                bd.setCsql(csql);
                int x;
               x = bd.Atualizar(bd.getCsql());
            }
        } catch (SQLException e) {}

        return element1;
    }

/**
 * This method will be used to get a message
 *
 * @param element as <code>Element</code>
 * @return Node
 */
    public Node processGetMessageRequest(Element element) {

          Element elementRequested = (Element)element.getElementsByTagName("MessageID").item(0);
```

```
String MessageID = elementRequested.getFirstChild().getNodeValue();
Element element1;
element1 = messageDOC.createElement("GetMessage-Response");
Element element2 = null;
try {
     String csql;
     csql = "SELECT * FROM message WHERE MessageID = " + MessageID;
     bd.setCsql(csql);
     bd.setLer (bd.Consulta(bd.getCsql()));
     element2 = messageDOC.createElement("MessageInfo");
     Element element3 = messageDOC.createElement("MessageID");
     org.w3c.dom.Text text3 = messageDOC.createTextNode(bd.getLer().getString("MessageID"));
     element3.appendChild(text3);
     element2.appendChild(element3);
     Element element4 = messageDOC.createElement("ContentType");
     org.w3c.dom.Text text4 = messageDOC.createTextNode(bd.getLer().getString("ContentType"));
     element4.appendChild(text4);
     element2.appendChild(element4);
     Element element5 = messageDOC.createElement("ContentEncoding");
     org.w3c.dom.Text text5 =
        messageDOC.createTextNode(bd.getLer().getString("ContentEncoding"));
     element5.appendChild(text5);
     element2.appendChild(element5);
     Element element55 = messageDOC.createElement("ContentSize");
     org.w3c.dom.Text text55 =
        messageDOC.createTextNode(bd.getLer().getString("ContentSize"));
     element55.appendChild(text55);
     element2.appendChild(element55);
     Element element6 = messageDOC.createElement("Recipient");
     Element element601 = messageDOC.createElement("User");
     Element element60101 = messageDOC.createElement("UserID");
     org.w3c.dom.Text text60101 =
        messageDOC.createTextNode(bd.getLer().getString("Recipient"));
     element60101.appendChild(text60101);
     element601.appendChild(element60101);
     element6.appendChild(element601);
     element2.appendChild(element6);
     Element element7 = messageDOC.createElement("Sender");
     Element element701 = messageDOC.createElement("User");
     Element element70101 = messageDOC.createElement("UserID");
     org.w3c.dom.Text text70101 = messageDOC.createTextNode(bd.getLer().getString("Sender"));
     element70101.appendChild(text70101);
     element701.appendChild(element70101);
     element7.appendChild(element701);
     element2.appendChild(element7);
     Element element8 = messageDOC.createElement("DateTime");
     org.w3c.dom.Text text8 = messageDOC.createTextNode(bd.getLer().getString("DateTime"));
     element8.appendChild(text8);
     element2.appendChild(element8);
     Element element9 = messageDOC.createElement("Validity");
     org.w3c.dom.Text text9 = messageDOC.createTextNode(bd.getLer().getString("Validity"));
     element9.appendChild(text9);
     element2.appendChild(element9);
     element1.appendChild(element2);
     Element element10 = messageDOC.createElement("ContentData");
     org.w3c.dom.Text text10 =
        messageDOC.createTextNode(bd.getLer().getString("ContentData"));
     element10.appendChild(text10);
     element1.appendChild(element10);
  } catch (SQLException e) {}
```

```
        return element1;
    }

/**
 * This method will be used to get all the messages that a client sent to another
 * client
 *
 * @param element as <code>Element</code>
 * @return Node
 */
    public Node processMessageDelivered(Element element) {

        Element elementRequested = (Element)element.getElementsByTagName("MessageID").item(0);
        String MessageID = elementRequested.getFirstChild().getNodeValue();
        try  {
            String csql;
            csql = "DELETE FROM message WHERE MessageID=" + MessageID;
            bd.setCsql(csql);
            int x;
            x = bd.Inserir(bd.getCsql());
        } catch (SQLException e) {}

        Element element2 = messageDOC.createElement("Status");
        Element element3 = messageDOC.createElement("Result");
        Element element4 = messageDOC.createElement("Code");
        org.w3c.dom.Text text = messageDOC.createTextNode("200");
        element4.appendChild(text);
        Element element5 = messageDOC.createElement("Description");
        org.w3c.dom.Text text1 = messageDOC.createTextNode("Succesfully completed.");
        element5.appendChild(text1);
        element3.appendChild(element4);
        element3.appendChild(element5);
        element2.appendChild(element3);
        return element2;
    }

/**
 * This method will be used to get all the messages that a WV client has
 *
 * @param element as <code>Element</code>
 * @return Node
 */
    public Node processForwardMessageRequest(Element element){

        Element elementRequested1 = (Element)element.getElementsByTagName("MessageID").item(0);
        String MessageID = elementRequested1.getFirstChild().getNodeValue();
        Element elementRequested2 = (Element)element.getElementsByTagName("UserID").item(0);
        String UserID = elementRequested2.getFirstChild().getNodeValue();
//          Aki o SAP trataria este MessageID e o UserID do destinatario para encaminhar a mensagem

        try {
            String csql;
            csql = "UPDATE message SET Recipient ='" + UserID.replaceAll("'", "''") +"', Notified = 'N'
WHERE  MessageID=" + MessageID;
            bd.setCsql(csql);
            int x;
            x = bd.Atualizar(bd.getCsql());
         } catch (SQLException e) {}
```

```
        Element element2 = messageDOC.createElement("Status");
        Element element3 = messageDOC.createElement("Result");
        Element element4 = messageDOC.createElement("Code");
        org.w3c.dom.Text text = messageDOC.createTextNode("200");
        element4.appendChild(text);
        Element element5 = messageDOC.createElement("Description");
        org.w3c.dom.Text text1 = messageDOC.createTextNode("Succesfully completed.");
        element5.appendChild(text1);
        element3.appendChild(element4);
        element3.appendChild(element5);
        element2.appendChild(element3);
        return element2;
    }

/**
 * This method will be used to reject a message
 *
 * @param element as <code>Element</code>
 * @return Node
 */
    public Node processRejectMessageRequest(Element element) {

        Element elementRequested = (Element)element.getElementsByTagName("MessageID").item(0);
        String MessageID = elementRequested.getFirstChild().getNodeValue();
        try {
             String csql;
             csql = "DELETE FROM message WHERE MessageID=" + MessageID;
             bd.setCsql(csql);
              int x;
              x = bd.Inserir(bd.getCsql());
        } catch (SQLException e) {}

        Element element2 = messageDOC.createElement("Status");
        Element element3 = messageDOC.createElement("Result");
        Element element4 = messageDOC.createElement("Code");
        org.w3c.dom.Text text = messageDOC.createTextNode("200");
        element4.appendChild(text);
        Element element5 = messageDOC.createElement("Description");
        org.w3c.dom.Text text1 = messageDOC.createTextNode("Succesfully completed.");
        element5.appendChild(text1);
        element3.appendChild(element4);
        element3.appendChild(element5);
        element2.appendChild(element3);
        return element2;
    }

/**
 * This method will be used to list all the contact list information
 *
 * @param element as <code>Element</code>
 * @return Node
 */
    public Node processListManageRequest(Element element) {

        Element element1 = (Element)element.getElementsByTagName("ContactList").item(0);
        String s = element1.getFirstChild().getNodeValue();
        Element element2 = messageDOC.createElement("ListManage-Response");
        Element element3 = messageDOC.createElement("Result");
        Element element4 = messageDOC.createElement("Code");
        org.w3c.dom.Text text = messageDOC.createTextNode("200");
```

```
        element4.appendChild(text);
        element3.appendChild(element4);
        element2.appendChild(element3);
        Element element5 = messageDOC.createElement("NickList");
        String as[][] = null;
        Object obj = null;
        try {
           PresenceSAPInterfaceClient presencesapinterfaceclient = new
                PresenceSAPInterfaceClient("192.168.200.11");
           as = presencesapinterfaceclient.getContactList(s);
        }
        catch(RemoteException remoteexception) { }
        try {
           for(int i = 0; i < as.length; i++) {
              Element element6 = messageDOC.createElement("NickName");
              Element element7 = messageDOC.createElement("Name");
              Element element8 = messageDOC.createElement("UserID");
              org.w3c.dom.Text text1 = messageDOC.createTextNode(as[i][0]);
              org.w3c.dom.Text text2 = messageDOC.createTextNode(as[i][1]);
              element7.appendChild(text1);
              element8.appendChild(text2);
              element6.appendChild(element7);
              element6.appendChild(element8);
              element5.appendChild(element6);
           }
        }
        catch(NullPointerException nullpointerexception) { }
        element2.appendChild(element5);
        return element2;
     }

  /**
   * This method will be used to get the name of an user using his phone
   *
   * @param element as <code>Element</code>
   * @return Node
   */
     public Node processGetNameRequest(Element element) {

                        Element element1 = (Element)element.getElementsByTagName("Phone").item(0);
                        String s = element1.getFirstChild().getNodeValue();
                        PhoneBookSearch phoneSearch = new PhoneBookSearch(s, 0);
                        String result = phoneSearch.getData();
                        Element element2 = messageDOC.createElement("GetName-Response");
                        Element element3 = messageDOC.createElement("Result");
                        Element element4 = messageDOC.createElement("Code");
                        org.w3c.dom.Text text = messageDOC.createTextNode("200");
                        element4.appendChild(text);
                        element3.appendChild(element4);
                        element2.appendChild(element3);
                        Element element5 = messageDOC.createElement("Name");
                        org.w3c.dom.Text text1 = messageDOC.createTextNode(result);
                        element5.appendChild(text1);
                        element2.appendChild(element5);
                        return element2;
           }

  /**
   * This method will be used to get the phone of an user using his name
   *
```

```
 * @param element as <code>Element</code>
 * @return Node
 */
          public Node processGetPhoneRequest(Element element) {

                    Element element1 = (Element)element.getElementsByTagName("Name").item(0);
                    String s = element1.getFirstChild().getNodeValue();
                    PhoneBookSearch nameSearch = new PhoneBookSearch(s, 1);
                    String result = nameSearch.getData();
                    Element element2 = messageDOC.createElement("GetPhone-Response");
                    Element element3 = messageDOC.createElement("Result");
                    Element element4 = messageDOC.createElement("Code");
                    org.w3c.dom.Text text = messageDOC.createTextNode("200");
                    element4.appendChild(text);
                    element3.appendChild(element4);
                    element2.appendChild(element3);
                    Element element5 = messageDOC.createElement("Phone");
                    org.w3c.dom.Text text1 = messageDOC.createTextNode(result);
                    element5.appendChild(text1);
                    element2.appendChild(element5);
                    return element2;
                    }

/**
 * This method will be used to get the presence information of an user
 *
 * @param element as <code>Element</code>
 * @return Node
 */
   public Node processGetPresenceRequest(Element element)
   {
      boolean flag = false;
      boolean flag2 = false;
      NodeList nodelist1 = messageDOC.getElementsByTagName("User");
      PresenceSAPInterfaceClient presencesapinterfaceclient = new
           PresenceSAPInterfaceClient("192.168.200.11");
      String as[] = new String[nodelist1.getLength()];
      for(int j = 0; j < nodelist1.getLength(); j++) {
         Element element3 = (Element)nodelist1.item(j);
         Element element4 = (Element)element3.getElementsByTagName("UserID").item(0);
         as[j] = element4.getFirstChild().getNodeValue();
         System.out.println(as[j]);
      }
      String s1 = "Online";
      String s2 = null;
      Element element6 = messageDOC.createElement("GetPresence-Response");
      Element element7 = messageDOC.createElement("Result");
      Element element8 = messageDOC.createElement("Code");
      org.w3c.dom.Text text = messageDOC.createTextNode("200");
      element8.appendChild(text);
      Element element9 = messageDOC.createElement("Description");
      org.w3c.dom.Text text1 = messageDOC.createTextNode("Successfully completed.");
      element9.appendChild(text1);
      element7.appendChild(element8);
      element7.appendChild(element9);
      element6.appendChild(element7);
      for(int k = 0; k < as.length; k++) {
         try {
            s2 = presencesapinterfaceclient.getLocation(as[k]);
         }
```

```
            catch(RemoteException remoteexception) {}

            Element element10 = messageDOC.createElement("OnlineStatus");
            Element element11 = messageDOC.createElement("Qualifier");
            org.w3c.dom.Text text2 = messageDOC.createTextNode("T");
            element11.appendChild(text2);
            Element element12 = messageDOC.createElement("PresenceValue");
            org.w3c.dom.Text text3 = messageDOC.createTextNode(s1);
            element12.appendChild(text3);
            element10.appendChild(element11);
            element10.appendChild(element12);
            Element element13 = messageDOC.createElement("Address");
            Element element14 = messageDOC.createElement("Qualifier");
            org.w3c.dom.Text text4 = messageDOC.createTextNode("T");
            element14.appendChild(text4);
            Element element15 = messageDOC.createElement("City");
            org.w3c.dom.Text text5 = messageDOC.createTextNode("Brasilia");
            element15.appendChild(text5);
            Element element16 = messageDOC.createElement("NamedArea");
            org.w3c.dom.Text text6 = messageDOC.createTextNode(s2);
            element16.appendChild(text6);
            element13.appendChild(element14);
            element13.appendChild(element15);
            element13.appendChild(element16);
            Element element17 = messageDOC.createElement("PresenceSubList");
            element17.appendChild(element10);
            element17.appendChild(element13);
            Element element5 = messageDOC.createElement("UserID");
            org.w3c.dom.Text text7 = messageDOC.createTextNode(as[k]);
            element5.appendChild(text7);
            Element element18 = messageDOC.createElement("Presence");
            element18.appendChild(element5);
            element18.appendChild(element17);
            element6.appendChild(element18);
        }

        return element6;
    }

}
```

Apêndice D: BD.java

```
/*
 * @ 2003 - Lemom - UnB
 */

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

/**
 *
 * @author Lemom
 *
 * Class responsible for returning the result of all queries
 */
public class BD {

	private String csql = null;
	private Connection conectar = null;
	private Statement resultado = null;
	private ResultSet ler = null;
	private String url = "jdbc:mysql://127.0.0.1:3306/";
	private String driver = "org.gjt.mm.mysql.Driver";

/**
 * General constructor for this class which should not be used
 *
 */
	public BD() {}

/**
 * Constructor used if the user provides a url and a driver
 *
 * @param urlIncoming as <code>String</code>
 * @param driveIncoming as <code>String</code>
 */
	public BD(String urlIncoming, String driverIncoming) {

		url = urlIncoming;
		driver = driverIncoming;
		Conexao();
	}

/**
 * Method for creating a connection with the database
 *
 */
	public void Conexao() {

		try {
			Class.forName(driver);
			conectar = DriverManager.getConnection(url,"root","");
			resultado = conectar.createStatement();
		}
		catch (ClassNotFoundException e) {}
		catch (SQLException e) {}
```

```
        }

/**
 * Method for creating a connection with the database if the user provides the url
 * and driver
 *
 * @param urlIncoming as <code>String</code>
 * @param driverIncoming as <code>String</code>
 */
        public void Conexao(String urlIncoming, String driverIncoming) {

                url = urlIncoming;
                driver = driverIncoming;
                Conexao();
   }

/**
 * Setter method for the query String
 *
 * @param csql as <code>String</code>
 */
        public void setCsql(String csql) {

                this.csql = csql;
        }

/**
 * Setter method for the Connection
 *
 * @param conectar as <code>Connection</code>
 */
        public void setConectar(Connection conectar) {

                this.conectar = conectar;
        }

/**
 * Setter method for the Statement
 *
 * @param resultado as <code>Statement</code>
 */
        public void setResultado(Statement resultado) {

                this.resultado = resultado;
        }

/**
 * Setter method for the ResultSet
 *
 * @param ler as <code>ResultSet</code>
 */
        public void setLer(ResultSet ler) {

                this.ler = ler;
        }

/**
 * Setter method for the Database URL
 *
 * @param url as <code>String</code>
```

```
 */
	public void setUrl(String url) {

		this.url = url;
	}

/**
 * Setter method for the Driver name
 *
 * @param driver as <code>String</code>
 */
	public void setDrive(String driver) {

		this.driver = driver;
	}

/**
 * Getter method for the query String
 *
 * @return <code>String</code>
 */
	public String getCsql() {

		return csql;
	}

/**
 * Getter method for the Connection
 *
 * @return <code>Connection</code>
 */
	public Connection getConectar() {

		return conectar;
	}
/**
 * Getter method for the Statement
 *
 * @return <code>Statement</code>
 */
	public Statement getResultado() {

		return resultado;
	}

/**
 * Getter method for the ResultSet
 *
 * @return <code>ResultSet</code>
 */
	public ResultSet getLer() {

		return ler;
	}

/**
 * Getter method for the Database URL
 *
 * @return <code>String</code>
 */
```

```
	public String getUrl() {

		return url;
	}

/**
 * Getter method for the Driver name
 *
 * @return <code>String</code>
 */
	public String getDrive() {

		return driver;
	}

/**
 * Method responsible for making all searches with the Database
 *
 * @param sql as <code>String</code>
 * @return ResultSet
 * @throws SQLException
 */
	public ResultSet Consulta(String sql) throws SQLException {

		return resultado.executeQuery(sql);
	}

/**
 * Method responsible for inserting information in the Database
 *
 * @param sql as <code>String</code>
 * @return int
 * @throws SQLException
 */
	public int Inserir(String sql) throws SQLException {

		return resultado.executeUpdate(sql);
	}

/**
 * Method responsible for removing information in the Database
 *
 * @param sql as <code>String</code>
 * @return int
 * @throws SQLException
 */
	public int Apagar(String sql) throws SQLException {

		return resultado.executeUpdate(sql);
	}

/**
 * Method responsible for updating information in the Database
 *
 * @param sql as <code>String</code>
 * @return int
 * @throws SQLException
 */
	public int Atualizar(String sql) throws SQLException {
```

```
                return resultado.executeUpdate(sql);
        }

/**
 * Method responsible for closing connections with the Database
 *
 * @throws Exception
 */
        public void FecharConexao()throws Exception {

                if (conectar != null)
                        conectar.close();
        }

}
```

Apêndice E: SAP_Authentication.java

```
/*
 * @ 2003 - Lemom - UnB
 */

import java.sql.*;

/**
 *
 * @author Lemom
 *
 * Class that will authenticate the WV client to the server
 */
public class SAP_Authentication {

   private static BD bd;
   private String sessionID;
   private boolean flag = false;

/**
 * Constructor for this class which will create a new instance of the Database
 * connection class
 *
 */
   public SAP_Authentication() {

         bd = new BD();
         bd.Conexao();
   }

/**
 * Class that will check if the login information for a user is correct
 *
 * @param login as <code>String</code>
 * @param password as <code>String</code>
 * @return boolean
 */
   public boolean checkLogin(String login, String password) {

         try {
               String csql;
               csql = "SELECT * FROM User WHERE UserID = \"" + login + "\" AND Password = \"" +
password +"\"";
               bd.setCsql(csql);
               bd.setLer(bd.Consulta(bd.getCsql()));
                if (bd.getLer().getString("SessionID") != null) {
                  sessionID = bd.getLer().getString("SessionID");
                  flag = true;
               } else {
                  sessionID = null;
               }
         } catch (SQLException e) {}

         return flag;
   }

/**
 * Getter method for the SessionID
 *
```

```
 * @return String
 */
   public String getSessionID() {

           return sessionID;
   }

/**
 * Getter method for the UserID by using the SessionID
 *
 * @param sessionID as <code>String</code>
 * @return String
 */
   public static String getUserID(String sessionID) {

           String userID = null;
           try {
                 String csql;
                 csql = "SELECT UserID FROM User WHERE SessionID = \"" + sessionID + "\"";
                 bd.setCsql(csql);
                 bd.setLer (bd.Consulta(bd.getCsql()));
                 if (bd.getLer().getString("UserID") != null) {
                      userID = bd.getLer().getString("UserID");
                 } else {
                      userID = null;
                 }
           } catch (SQLException e) {}
           return userID;
   }

}
```

Apêndice F: PhoneBookSearch.java

```
/*
 *  @2003 Lemom - UnB
 */

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

/**
 *
 * @author Marcos Dytz
 *
 * This class will connect to the Database and retrieve the phone or name
 *  information of an user and prepare it for being delivered to the
 *  mobile client
 */
public class PhoneBookSearch {

	private String query;
	private String result;
	private final String DRIVER = "com.mysql.jdbc.Driver";
	private final String URL = "jdbc:mysql://127.0.0.1:3306/wv";

	Connection connection;
	Statement statement;
	ResultSet resultSet;

/**
 * General constructor for this class, should not be used
 *
 */
	public PhoneBookSearch () {

		System.out.println("Please fill a Name or Phone.");
	}

/**
 * This constructor will handle all the requests for this class where the<br>
 * <code>type</code> will be used to set the proper method to call
 *
 * @param data as <code>String</code>
 * @param type as <code>int</code>
 */
	public PhoneBookSearch(String data, int type) {

		if (type == 0) {
		// Name Search
			getName(data);
		} else if (type == 1) {
		// Phone Search
			getPhone(data);
		} else {
			System.out.println("Improper data entered for search.");
		}
	}
```

```
/**
 * Method that will handle the requests to retrieve a name from a phone through<br>
 * the database
 *
 * @param Phone as <code>String</code>
 */
	public void getName (String Phone) {

		query = "SELECT Name, AllowSearch FROM PhoneBook WHERE Phone = \"" +
Phone + "\"";
		try {
			Class.forName(DRIVER);
			connection = DriverManager.getConnection(URL,"root","");
			statement = connection.createStatement();
			resultSet = statement.executeQuery(query);
			while (resultSet.next()) {
				if (resultSet.getString("AllowSearch").equals("1")) {
					result = resultSet.getString("Name");
				} else {
					System.out.println("Search for " +
resultSet.getString("Name") + " not permitted by the user.");
					result = "Not Permitted";
				}
				System.out.println(result);
			}
			resultSet.close();
			statement.close();
			connection.close();
		} catch(SQLException e) {
			System.out.println("SQL Exception: " + e.getMessage());
			e.printStackTrace();
			System.exit(1);
		} catch(ClassNotFoundException e) {
			System.out.println("Driver Exception: " + e.getMessage());
			e.printStackTrace();
			System.exit(1);
		} catch(Exception e) {
			System.out.println("Other Exception: " + e.getMessage());
			e.printStackTrace();
			System.exit(1);
		}
	}

/**
 * Method that will handle the requests to retrieve a phone from a name through<br>
 * the database
 *
 * @param Name as <code>String</code>
 */
	public void getPhone (String Name) {

		query = "SELECT Phone, AllowSearch FROM PhoneBook WHERE Name LIKE \"%"
+ Name + "%\"";
		try {
			Class.forName(DRIVER);
			connection = DriverManager.getConnection(URL,"root","");
			statement = connection.createStatement();
			resultSet = statement.executeQuery(query);
			while (resultSet.next()) {
				if (resultSet.getString("AllowSearch").equals("1")) {
```

```
                                        result = resultSet.getString("Phone");
                                } else {
                                        System.out.println("Search for " +
resultSet.getString("Phone") + " not permitted by the user.");
                                        result = "Not Permitted";
                                }
                                System.out.println(result);
                        }
                        resultSet.close();
                        statement.close();
                        connection.close();
                } catch(SQLException e) {
                        System.out.println("SQL Exception: " + e.getMessage());
                        e.printStackTrace();
                        System.exit(1);
                } catch(ClassNotFoundException e) {
                        System.out.println("Driver Exception: " + e.getMessage());
                        e.printStackTrace();
                        System.exit(1);
                } catch(Exception e) {
                        System.out.println("Other Exception: " + e.getMessage());
                        e.printStackTrace();
                        System.exit(1);
                }
        }

/**
 * This method will return all the requests for this class
 *
 * @return <code>String</code>
 */
        public String getData () {
                return result;
        }
}
```

Apêndice G: WVClient.java

```
/*
 * @ 2003 - LEMOM - UnB
 *
 * Executa todo o controle da entrada de dados, do funcionamento de MIDlet e da interface
 *  gráfica do aparelho móvel
 */
import javax.microedition.lcdui.*;          // Command, Textfield, Form
import javax.microedition.midlet.*;         // MIDlet
import lemom.wv.client.services.*;          // Serviços criados para o sistema
import lemom.wv.client.xml.*;               // Manipulação XML
import java.util.*;

/**
 * This class is the MIDlet os the client application and is responsible for all
 * actions taken by the client
 */
public class WVClient extends MIDlet implements CommandListener {

        String urlSAP = "http://127.0.0.1:8080/Sap/servlet/SAP_CSPservlet";

        private String userID = "teste";
        private String password = "";
        private String userIDrecipient;

        private Display display;
        private Form inputForm;
        private Alert outputAlert;
        private TextField TFitem1;
        private TextField TFitem2;
        private ChoiceGroup choiceItem;
        private List list;
        private List PhoneSearchList;
        private List NameSearchList;
        private List ContactList;
        private Vector vector(15, 5);

        private Command quit = new Command("Sair", Command.EXIT, 1);
        private Command cancel = new Command("Cancela", Command.CANCEL, 1);
        private Command select = new Command("Selec.", Command.OK, 1);
        private Command submit = new Command("Submete", Command.OK, 1);
        private Command submitLogin = new Command("OK", Command.OK, 1);
        private Command submitRead = new Command("Ler", Command.OK, 1);
        private Command submitFwd = new Command("Encaminhar", Command.OK, 1);
        private Command submitFwd2 = new Command("Encaminhar", Command.OK, 2);
        private Command submitDel = new Command("Apagar", Command.OK, 1);
        private Command newSearch = new Command("Nova busca", Command.OK, 1);
        private Command back = new Command("Voltar", Command.OK, 1);
        private Command locate = new Command("Localizar", Command.OK, 1);

        private static final String[] Option = {
                        "SendMessageRequest",
                        "GetMessageListRequest",
                        "SetDeliveryMethodRequest",
                        "GetContactList",
                        "SetPhoneSearch",
                        "SetNameSearch"
            };
```

```
        private static final String[] OptionNames = {
                                "Enviar nova IM",
                                 "Ver lista de mensagens",
                                "Modo de entrega",
                                "Carregar a Lista de Contatos",
                                "Procurar Telefone",
                                "Procurar Nome"
            };

        private List IMFeatureList = new List("Instant Messaging",
                                 List.IMPLICIT,
                                 OptionNames,
                                 null);

        private PrimitiveRequest primReq;
        private Session session;
        private WVConnector con;
        private IMFeature primIM;
        private IMpolling primIMpolling;

        private String phoneBookSearch = null;
        private String[] resultSearch;
//      int busca;          // Usada para guardar o valor para uma nova busca de nome ou telefone

/**
 * WVClient constructor starts the connection with the server application
 */
        public WVClient() {

           setLoginForm();
           con = new WVConnector(urlSAP);
           primIM = new IMFeature(con);  primIMpolling = new IMpolling(con);
           primReq = new PrimitiveRequest(con);  session = Session.getInstance();
        }

/**
 * Responsible for the login of the user
 */
        public void setLoginForm() {

                inputForm = new Form("Tela de Login");
                TFitem1 = new TextField("Login", userID, 40, TextField.ANY);
                inputForm.append(TFitem1);
                TFitem2 = new TextField("Senha", password, 15, TextField.PASSWORD);
                inputForm.append(TFitem2);
                inputForm.addCommand(quit);
                inputForm.addCommand(submitLogin);
                inputForm.setCommandListener(this);
        }

/**
 * Responsible for showing the features available by the Instant Messaging service
 */
        public void setIMFeatureList() {

                IMFeatureList.addCommand(quit);
                IMFeatureList.addCommand(select);
                IMFeatureList.setCommandListener(this);
        }
```

```
/**
 * Responsible for cleaning the Input Form
 */
	public void clearInputFormItems() {

		if (inputForm != null) {
			for (int i = (inputForm.size()-1);i>=0;i--)
				inputForm.delete(i);
		}
	}

/**
 * Responsible for opening the view for writing a new message
 *  @param  strTitle as <code>String</code>
 *  @param strMsg as <code>String</code>
 */
	public void setMsgInputForm(String strTitle, String strMsg) {

		inputForm = new Form(strTitle);
		inputForm.append(strMsg);
		inputForm.setCommandListener(this);
	}

/**
 * Responsible for sending the message
 */
	public void setSendMessageRequestInputForm() {

		inputForm = new Form(null);
		Ticker titulo = new Ticker ("SendMessageRequest");
		inputForm.setTicker(titulo);
		TFitem1 = new TextField("Destinatario", userIDrecipient, 50, TextField.ANY);
		inputForm.append(TFitem1);
		TFitem2 = new TextField("Mensagem", "Texto da mensagem.", 500, TextField.ANY);
		inputForm.append(TFitem2);
		inputForm.addCommand(cancel);
		inputForm.addCommand(submit);
		inputForm.setCommandListener(this);
	}

/**
 * Responsible for setting the delivery method (Pull or Notification)
 */
	public void setSetDeliveryMethodRequestInputForm() {

	    inputForm = new Form(null);
	    Ticker titulo = new Ticker ("DeliveryMethodRequest");
	    inputForm.setTicker(titulo);
	    String auxArray[] = {"Push", "Notify/Get"};
	    choiceItem = new ChoiceGroup ("Metodo de entrega", Choice.EXCLUSIVE, auxArray, null);
	    inputForm.append(choiceItem);
	    TFitem1 = new TextField("Tamanho do conteudo", "2048", 7, TextField.NUMERIC);
	    inputForm.append(TFitem1);
	    inputForm.addCommand(cancel);
	    inputForm.addCommand(submit);
	    inputForm.setCommandListener(this);
	}

/**
 * Responsible for getting the Message List of the user
```

```
 */
	public void setGetMessageListRequest() {

		inputForm = new Form(null);
		Ticker titulo = new Ticker ("GetMessageListRequest");
		inputForm.setTicker(titulo);
		TFitem1 = new TextField("Limite de mensagens", "5", 3, TextField.NUMERIC);
		inputForm.append(TFitem1);
		inputForm.addCommand(cancel);
		inputForm.addCommand(submit);
		inputForm.setCommandListener(this);
	}

/**
 * Responsible for getting the Contact List of the user
 */
	public void setGetContactList () {

		inputForm  = new Form(null);
		Ticker titulo = new Ticker ("GetContactList");
		inputForm.setTicker(titulo);
		inputForm.addCommand(cancel);
		inputForm.addCommand(submit);
		inputForm.setCommandListener(this);
	}

/**
 * Responsible for starting the phone search
 */
	public void setSetPhoneSearchInputForm() {

		inputForm = new Form(null);
		TFitem1 = new TextField("Phone", phoneBookSearch, 40, TextField.ANY);
//		Ticker titulo = new Ticker ("PhoneSeach");
//		inputForm.setTicker(titulo);
		inputForm.append(TFitem1);
		inputForm.addCommand(cancel);
		inputForm.addCommand(submit);
		inputForm.setCommandListener(this);
	}

/**
 * Responsible for starting the name search
 */
	public void setSetNameSearchInputForm() {

		inputForm = new Form(null);
		TFitem1 = new TextField("Name", phoneBookSearch, 40, TextField.ANY);
//		Ticker titulo = new Ticker ("NameSearch");
//		inputForm.setTicker(titulo);
		inputForm.append(TFitem1);
		inputForm.addCommand(cancel);
		inputForm.addCommand(submit);
		inputForm.setCommandListener(this);
	}


/**
 * Responsible for showing the list of all messages
 */
```

```
	public void setMessageList () {

		list = new List("Mensagens", List.IMPLICIT);
		list.addCommand(cancel);
		list.addCommand(submitRead);
		list.addCommand(submitFwd);
		list.addCommand(submitDel);
		list.setCommandListener(this);
	}

/**
 * Responsible for forwarding a message to another user
 */
	public void setForwardMessageRequest() {

		String aux = ((MessageInfo)vector.elementAt(list.getSelectedIndex())).Sender +
		" " + ((MessageInfo)vector.elementAt(list.getSelectedIndex())).ContentSize;
		inputForm = new Form(null);
		Ticker titulo = new Ticker ("Encaminhando mensagem");
		inputForm.setTicker(titulo);
		inputForm.append(aux);
		TFitem1 = new TextField("Destinatario", userIDrecipient, 50, TextField.ANY);
		inputForm.append(TFitem1);
		inputForm.addCommand(back);
		inputForm.addCommand(submitFwd2);
		inputForm.setCommandListener(this);
	}

/**
 * Starts the application
 */
	public void startApp() {

		display = Display.getDisplay(this);
		display.setCurrent(inputForm);
	}

/**
 * Responsible for listening for the user entries and making the actions according to this entry
 *  @param c as a <code>Command,</code>
 *  @param s as a <code>Displayable</code>
 */
	public void commandAction(Command c, Displayable s) {

		if (c == quit) {
			destroyApp(false);
			notifyDestroyed();
		}
	 // pos - posicao da opcao selecionada pelo usuario na IMFeatureList
	 //	será usada para criar o FORMULARIO e tratar a submissao deste FORMULARIO

		int pos = IMFeatureList.getSelectedIndex();

	 // Trata a submissao do FORMULARIO
		if (c == submit) {
			clearInputFormItems();
			setMsgInputForm("PROCESSANDO", "Aguarde...");
			display.setCurrent(inputForm);
			Thread t = new Thread() {
				public void run() {
```

```
                switch (IMFeatureList.getSelectedIndex()) {

                    //"SendMessageRequest"
                    case 0:
                     try {
                      if (primIM.SendMessageRequest(userID,      //strSender
                                                    TFitem1.getString(), //strRecipient
                                                    TFitem2.getString(), //strContent
                                                     null, //strDeliveryReportRequest
                                                    -1)) //intValidity
                      {
                        outputAlert = new Alert ("SUCESSO", "Mensagem Enviada com sucesso!", null, AlertType.CONFIRMATION);
                        outputAlert.setTimeout(4000); // Tempo em milisegundos
                      } else {
                        outputAlert = new Alert ("ERRO", "Ocorreu um erro no envio da mensagem!", null, AlertType.ERROR);
                        outputAlert.setTimeout(4000); // Tempo em milisegundos
                      }
                     display.setCurrent(outputAlert, IMFeatureList);
                     }
                  //Trata do caso em que o conteudo da mensagem possui caracteres especiais
                     catch (NoSuchElementException e) {
                        outputAlert = new Alert ("ERRO", "Ocorreu um erro no envio da mensagem! Caracteres invalidos!", null, AlertType.ERROR);
                      outputAlert.setTimeout(4000); // Tempo em milisegundos
                      display.setCurrent(outputAlert, IMFeatureList);
                      };
                      break;

                    //"GetMessageListRequest"
                    case 1:
                      String strMessageCount = null;
                      String strlist = null;
                      if (TFitem1.size() > 0)
                        strMessageCount = TFitem1.getString();
                      vector = primIM.GetMessageListRequest(strMessageCount, null);
                      if (vector.size() != 0)          {
                        setMessageList();
                        for (int i = 0; i<vector.size(); i++) {
                          String aux = ((MessageInfo)vector.elementAt(i)).Sender + " " + ((MessageInfo)vector.elementAt(i)).ContentSize;
                          list.append(aux, null);
                        }
                        display.setCurrent(list);
                      } else {
                        clearInputFormItems();
                        setMsgInputForm("Mensagens", "Nenhuma mensagem");
                        inputForm.addCommand(cancel);
                        display.setCurrent(inputForm);
                      }
                      break;

                    //"SetDeliveryMethodRequest"
                    case 2:
                      String strDeliveryMethod = null,
                      strAcceptedContentLength = null;
                      if (choiceItem.getSelectedIndex() == 0)
                          strDeliveryMethod = "P";
                       else
```

```
                    strDeliveryMethod = "N";
                if (TFitem1.size() > 0)
                    strAcceptedContentLength = TFitem1.getString();
                if (primIM.SetDeliveryMethodRequest(strDeliveryMethod,
                                                    strAcceptedContentLength,
                                                    null))
                {
                        outputAlert = new Alert ("SUCESSO", "Sua alteracao foi
efetuada com sucesso!", null, AlertType.CONFIRMATION);
                    outputAlert.setTimeout(4000); // Tempo em milisegundos
                } else {
                      outputAlert = new Alert ("ERRO", "Ocorreu um erro no
SetDeliveryMethodRequest!", null, AlertType.ERROR);
                    outputAlert.setTimeout(4000); // Tempo em milisegundos
                }
               display.setCurrent(outputAlert, IMFeatureList);
                break;

             // "SetGetContactList"
             case 3:
                PresenceRequest presReq = new PresenceRequest(con);
                String[] clID = {“wv:john/My_friends@smith.com”};
                String[][] testa = presReq.getListRequest(clID);
                        ContactList = new List ("Contact List", List.IMPLICIT,
testa[0],null);
                ContactList.addCommand(cancel);
                ContactList.addCommand(locate);
                ContactList.setCommandListener(WVClient.this);
                display.setCurrent(ContactList);
                break;

             // "setGetPhoneSearchList"
            case 4:
                PhoneSearch phoneSearch = new PhoneSearch(con);
//              busca = 0;
                resultSearch = phoneSearch.getPhoneRequest(TFItem1.getString());
                  PhoneSearchList = new List ("Busca de telefone", List.IMPLICIT,
resultSearch, null);
                if (resultSearch[0].equals("Not Permitted")) {
                    Ticker titulo = new Ticker("Busca não permitida pelo usuário!");
                    NameSearchList.setTicker(titulo);
                }
                PhoneSearchList.addCommand(cancel);
//              PhoneSearchList.addCommand(newSearch);
                PhoneSearchList.setCommandListener(WVClient.this);
                display.setCurrent(PhoneSearchList);
                break;

            // "setGetNameSearchList"
            case 5:
                NameSearch nameSearch = new NameSearch(con);
//              busca = 1;
                resultSearch = nameSearch.getNameRequest(TFItem1.getString());
                    NameSearchList = new List ("Busca de nome", List.IMPLICIT,
resultSearch, null);
                if (resultSearch[0].equals("Not Permitted")) {
                    Ticker titulo = new Ticker("Busca não permitida pelo usuário!");
                    NameSearchList.setTicker(titulo);
                }
                NameSearchList.addCommand(cancel);
```

```
//                                      NameSearchList.addCommand(newSearch);
                                        NameSearchList.setCommandListener(WVClient.this);
                                        display.setCurrent(NameSearchList);
                                        break;

                                    default:
                                        System.out.println(IMFeatureList.getSelectedIndex());
                                        System.out.println("That's not a valid choice.");
                                        break;
                                }
                            }
                        };
                    t.start();
                }

/*                  if (c == newSearch) {

                            switch (busca) {

                                    // "setGetPhoneSearchList"
                                    case 0:
                                        PhoneSearch phoneSearch = new PhoneSearch(con);
                                        busca = 0;
                                        resultSearch = phoneSearch.getPhoneRequest(TFItem1.getString());
                                              PhoneSearchList = new List("Phone Search", List.IMPLICIT,
resultSearch, null);
                                        PhoneSearchList.addCommand(cancel);
                                        PhoneSearchList.addCommand(newSearch);
                                        PhoneSearchList.setCommandListener(WVClient.this);
                                        display.setCurrent(PhoneSearchList);
                                        break;

                                    // "setGetNameSearchList"
                                    case 1:
                                        NameSearch nameSearch = new NameSearch(con);
                                        busca = 1;
                                        resultSearch = nameSearch.getNameRequest(TFItem1.getString());
                                              NameSearchList = new List("Name Search", List.IMPLICIT,
resultSearch, null);
                                        NameSearchList.addCommand(cancel);
                                        NameSearchList.addCommand(newSearch);
                                        NameSearchList.setCommandListener(WVClient.this);
                                        display.setCurrent(NameSearchList);
                                        break;

                                    default:
                                        System.out.println(IMFeatureList.getSelectedIndex());
                                        System.out.println("That's not a valid choice.");
                                        break;
                            }
                }*/

                if (c == locate) {

                            GetPresenceRequest getPresenceRequest = new GetPresenceRequest();
                            String                                  xmlRequest                                  =
getPresenceRequest.makeRequest(ContactList.getString(ContactList.getSelectedIndex()));
                            System.out.println(xmlRequest);
                            String xmlResponse = con.communicate(xmlRequest);
                            System.out.println(xmlResponse);
```

```
                    String status = "Status: "+XMLUtils.getNodeValue(xmlResponse,"PresenceValue");
                    String cidade = "Cidade: "+XMLUtils.getNodeValue(xmlResponse,"City");
                    String area = "Area: "+XMLUtils.getNodeValue(xmlResponse,"NamedArea");
                    List locArea = new List("Location", List.IMPLICIT);
                    locArea.insert(0, status, null);
                    locArea.insert(1, cidade, null);
                    locArea.insert(2, area, null);
                    locArea.addCommand(cancel);
                    display.setCurrent(locArea);
                }

                if (c == submitRead) {

                    primIMpolling.cancelPolling();
                    clearInputFormItems();
                    setMsgInputForm("AGUARDE", "Recebendo mensagem...");
                    display.setCurrent(inputForm);
                    Thread t = new Thread()           {
                       public void run() {
                         String Message,
                         MessageID = ((MessageInfo)vector.elementAt(list.getSelectedIndex())).MessageID,
                         DateTime = ((MessageInfo)vector.elementAt(list.getSelectedIndex())).DateTime,
                         Sender = ((MessageInfo)vector.elementAt(list.getSelectedIndex())).Sender;
                         Message = primIM.GetMessageRequest(MessageID);
                         if (Message != null) {
                           clearInputFormItems();
                          setMsgInputForm("Mensagem", "");
                          inputForm.append(new StringItem("Enviada em:", DateTime));
                          inputForm.append(new StringItem("Remetente:", Sender));
                          inputForm.append(new StringItem("Conteudo:", Message));
                          inputForm.addCommand(back);
                          vector.removeElementAt(list.getSelectedIndex());
                          list.delete(list.getSelectedIndex());
                          display.setCurrent(inputForm);
                        } else {
                          outputAlert = new Alert ("ERRO", "Ocorreu um erro no recebimento da
mensagem!", null, AlertType.ERROR);
                          outputAlert.setTimeout(4000); // Tempo em milisegundos
                          display.setCurrent(outputAlert, list);
                        }
                       }
                    };
                    t.start();
                }

                if (c == submitFwd) {

                   setForwardMessageRequest();
                   display.setCurrent(inputForm);
                }

                if (c == submitFwd2) {

                  clearInputFormItems();
                  setMsgInputForm("AGUARDE", "Encaminhando mensagem...");
                  display.setCurrent(inputForm);

                  Thread t = new Thread() {
                    public void run() {
```

```
                                                                    String MessageID =
((MessageInfo)vector.elementAt(list.getSelectedIndex())).MessageID;
                        if (primIM.ForwardMessageRequest(MessageID, //strMessageID
                                                         TFitem1.getString())) //strRecipient
                        {
                           outputAlert = new Alert ("SUCESSO", "Mensagem Encaminhada com sucesso!", null,
AlertType.CONFIRMATION);
                          outputAlert.setTimeout(4000); // Tempo em milisegundos
                          vector.removeElementAt(list.getSelectedIndex());
                          list.delete(list.getSelectedIndex());
                        } else {
                              outputAlert = new Alert ("ERRO", "Ocorreu um erro ao tentar encaminhaar a
mensagem!", null, AlertType.ERROR);
                          outputAlert.setTimeout(4000); // Tempo em milisegundos
                        }
                        if (list.size() != 0)
                          display.setCurrent(outputAlert, list);
                        else
                          display.setCurrent(outputAlert, IMFeatureList);
                      }
                    };
                    t.start();
                  }

                  if (c == submitDel) {

                    clearInputFormItems();
                    setMsgInputForm("AGUARDE", "Apagando mensagem...");
                    display.setCurrent(inputForm);
                    Thread t = new Thread() {
                      public void run() {
                                                                    String MessageID =
((MessageInfo)vector.elementAt(list.getSelectedIndex())).MessageID;
                        if (primIM.RejectMessageRequest(MessageID)) {
                            outputAlert = new Alert ("SUCESSO", "Mensagem Apagada com sucesso!", null,
AlertType.CONFIRMATION);
                          outputAlert.setTimeout(4000); // Tempo em milisegundos
                          vector.removeElementAt(list.getSelectedIndex());
                          list.delete(list.getSelectedIndex());
                        } else {
                              outputAlert = new Alert ("ERRO", "Ocorreu um erro ao tentar encaminhaar a
mensagem!", null, AlertType.ERROR);
                          outputAlert.setTimeout(4000); // Tempo em milisegundos
                        }
                        if (list.size() != 0)
                         display.setCurrent(outputAlert, list);
                        else
                         display.setCurrent(outputAlert, IMFeatureList);
                      }
                    };
                    t.start();
                  }

                  if (c == back) {

                    if (list.size() != 0)
                      display.setCurrent(list);
                    else {
                              outputAlert = new Alert ("MENSAGENS", "Nenhuma mensagem", null,
AlertType.ERROR);
```

```
                    outputAlert.setTimeout(4000); // Tempo em milisegundos
                    display.setCurrent(outputAlert, IMFeatureList);
                  }
                  primIMpolling.executePolling();
                }

                if (c == cancel) {

                  IMFeatureList.setCommandListener(this);
                  display.setCurrent(IMFeatureList);
                }

                if (c == select) {

                  System.out.print("["+pos+"] ");
                  System.out.println(Option[pos]);
                  clearInputFormItems();
                  switch (pos) {

                    case 0: //"SendMessageRequest"
                            setSendMessageRequestInputForm();
                            break;

                    case 1: //"GetMessageListRequest"
                           setGetMessageListRequest();
                           break;

                    case 2: //"SetDeliveryMethodRequest"
                            setSetDeliveryMethodRequestInputForm();
                            break;

                    case 3: setGetContactList();
                            break;

                    case 4: //"SetPhoneSearch"
                            setSetPhoneSearchInputForm();
                            break;

                    case 2: //"SetNameSearch”
                            setSetNameSearchInputForm();
                            break;

                    default: System.out.println("Hey, that's not valid!");
                             break;
                  }
                 display.setCurrent(inputForm);
                }

                if (c == submitLogin) {

                  clearInputFormItems();
                  setMsgInputForm("AGUARDE", "Autenticando usuario");
                  display.setCurrent(inputForm);
                  Thread t = new Thread(){
                    public void run() {
                      if (primReq.loginRequest(TFitem1.getString(), TFitem2.getString())) {
                        outputAlert = new Alert ("SUCESSO", "Login bem sucedido!", null,
AlertType.CONFIRMATION);
                        outputAlert.setTimeout(4000); // Tempo em milisegundos
                        setIMFeatureList();
```

```
                        System.out.println("setIMFeature");
                        display.setCurrent(outputAlert, IMFeatureList);
                        primIMpolling.setDisplay(display);
                        primIMpolling.setDisplayable(IMFeatureList);
                        primIMpolling.executePolling();
                    } else {
                        outputAlert = new Alert ("ERRO", "Sua senha ou login não conferem", null,
AlertType.ERROR);
                        outputAlert.setTimeout(4000); // Tempo em milisegundos
                        setLoginForm();
                        display.setCurrent(outputAlert, inputForm);
                    }
                }
            };
            t.start();
        }
    }
/**
 * Responsible for pausing the application
 */
    public void pauseApp() {}

/**
 * Responsible for destroying the application
 * @param uncoditional as <code>boolean</code>
 */
    public void destroyApp(boolean unconditional) {

        primIMpolling.cancelPolling();
    }
}
```

Apêndice H: NameSearch.java

```
/*
 * @ 2003 Lemom - UnB
 */
package lemom.wv.client.services;
import java.util.Vector;
import lemom.wv.client.xml.*;

/**
 * This class writes XML codes for searching name attributes. <br>
 * The provided methods handle all XML constructions and use the <code>WVConnection</code>
 * passed to the constructor to communicate with the Wireless Village Server.
 *
 */
public class NameSearch extends AbstractRequest {

/**
 * NameSearch constructor calls the superclass constructor
 * passing the <code>WVConnection</code> con as a parameter.
 * @param con the <code>WVConnection</code>
 * @see AbstractRequest#AbstractRequest(WVConnection)
 */
   public NameSearch(WVConnection con) {
          super(con);
   }

/**
 * Writes a XML code for getting the name request and sends it to the WV Server.
 * @param phoneBookSearch the Phone of a person as a <code>String</code>
 * @return a <code>String</code> containing the Name of the person searched
 */
   public String getNameRequest(String phoneBookSearch) {

          message = new XMLMessage();
          String sessionValues[] = new String[2];
          sessionValues[0] = "Inband";
          sessionValues[1] = session.getSessionID();
          message.setSessionDescriptor(sessionValues); // Descritor da sessão
          String transaction[] = new String[3];
          transaction[0] = "Request";
          transaction[1] = "IMApp01#12345@NOK5110";
          transaction[2] = null;
          message.setTransactionDescriptor(transaction); // Descritor da transação
          String content = message.createSubElement("Phone", phoneBookSearch); // Elemento do XML
          message.setTransactionContent("GetName-Request", content); // Conteúdo da transação
          message.wrapMessage();
          System.out.println(message);
          String response = con.communicate(message.toString());
          System.out.println(response);
          if (XMLUtils.getNodeValue(response, "Code").firstElement().equals("200"))
             System.out.println("OK - " + session.getSessionID());
          Vector searchVector = XMLUtils.getNodeValue(response, "Phone");
          String[] result = new String[searchVector.size()];
          for (int i = 0; i < searchVector.size(); i++)
                 result[i] = (String) searchVector.elementAt(i); // Resultado da consulta
          message = null;
          return result;
   }
}
```

Apêndice I: PhoneSearch.java

```
/*
 * @ 2003 Lemom - UnB
 */
package lemom.wv.client.services;
import lemom.wv.client.xml.*;
import java.util.Vector;

/**
 * This class writes XML codes for searching phone attributes. <br>
 * The provided methods handle all XML constructions and use the <code>WVConnection</code>
 * passed to the constructor to communicate with the Wireless Village Server.
 *
 */
public class PhoneSearch extends AbstractRequest {

/**
 * PresenceRequest constructor calls the superclass constructor
 * passing the <code>WVConnection</code> con as a parameter.
 * @param con the <code>WVConnection</code>
 * @see AbstractRequest#AbstractRequest(WVConnection)
 */
   public PhoneSearch(WVConnection con) {
          super(con);
   }

/**
 * Writes a XML code for getting the phone request and sends it to the WV Server.
 * @param phoneBookSearch the Name of a person as a <code>String</code>
 * @return a <code>String</code> containing the Phone of the Name searched
 */
    public String[] getPhoneRequest(String phoneBookSearch) {

          message = new XMLMessage();
          String sessionValues[] = new String[2];
          sessionValues[0] = "Inband";
          sessionValues[1] = session.getSessionID();
          message.setSessionDescriptor(sessionValues); // Descritor da sessão
          String transaction[] = new String[3];
          transaction[0] = "Request";
          transaction[1] = "IMApp01#12345@NOK5110";
          transaction[2] = null;
          message.setTransactionDescriptor(transaction); // Descritor da transação
          String content = message.createSubElement("Name", phoneBookSearch); // Elemento do XML
          message.setTransactionContent("GetPhone-Request", content); // Conteúdo da transação
          message.wrapMessage();
          System.out.println(message);
          String response = con.communicate(message.toString());
          System.out.println(response);
          if (XMLUtils.getNodeValue(response, "Code").firstElement().equals("200"))
             System.out.println("OK - " + session.getSessionID());
          Vector searchVector = XMLUtils.getNodeValue(response, "Phone");
          String[] result = new String[searchVector.size()];
          for (int i = 0; i < searchVector.size(); i++)
                result[i] = (String) searchVector.elementAt(i); // Resultado da consulta
          message = null;
          return result;
   }
}
```

Apêndice J: PrimitiveRequest.java

```
/*
 * @LEMOM-UnB - 2003
 */

package lemom.wv.client.services;

import lemom.wv.client.xml.*;

/**
 * This class writes XML codes based on WV especifications for requesting primitive attributes. <br>
 * The provided methods handle all XML constructions and use the <code>WVConnection</code>
 * passed to the constructor to communicate with the Wireless Village Server.
 */

public class PrimitiveRequest extends AbstractRequest {

/**
 * PresenceRequest constructor calls the superclass constructor passing the <code>WVConnection</code> con
 * as a parameter.
 * @param con the <code>WVConnection</code>
 * @see AbstractRequest#AbstractRequest(WVConnection)
 */
public PrimitiveRequest(WVConnection con)  {

        super(con);
}

/**
 * Writes a XML code based on WV especifications for requesting login and sends it to the WV Server. <br>
 * <code>Session</code> attributes are set depending on success or failure during the authentication process.
 * @param user a registered user name.
 * @param password the user password for logging in.
 * @see Session
 */
public boolean loginRequest(String user, String password)  {

        boolean LoginResult;
        message = new XMLMessage();
        String sessionValues[] = new String[2];
        sessionValues[0] = "Outband";
        sessionValues[1] = null;
        message.setSessionDescriptor(sessionValues);
        String transaction[] = new String[3];
        transaction[0] = "Request";
        transaction[1] = "IMApp01#12345@NOK5110";
        transaction[2] = null;
        message.setTransactionDescriptor(transaction);
        String content = new String();
        String contentAux = new String();
        content = message.createSubElement("UserID", user);
        contentAux = message.createSubElement("URL", "http://206.226.10.25:80/IMPSAPP");
        contentAux = message.createSubElement("ClientID", contentAux);
        content += contentAux;
        contentAux = message.createSubElement("Password", password);
        content += contentAux;
        contentAux = message.createSubElement("TimeToLive", "120");
        content += contentAux;
        contentAux = message.createSubElement("SessionCookie", "im.user.com#20011224#328746293");
```

```
        content += contentAux;
        message.setTransactionContent("Login-Request", content);
        message.wrapMessage();
        String response = con.communicate(message.toString());
        System.out.println(message);
        System.out.println(response);

        if (XMLUtils.getNodeValue(response, "Code").firstElement().equals("200"))  {
                session.setStatusDescription((String)XMLUtils.getNodeValue(response,
                    "Description").firstElement());
                session.setLoginStatus(200);
                session.setSessionID((String)XMLUtils.getNodeValue(response,
                    “SessionID").firstElement());
                LoginResult = true;
        } else {
                session.setLoginStatus(531);
                LoginResult = false;
        }
        message = null; //garbage collector
        return LoginResult;
        }

}
```

Apêndice K: AbstractRequest.java

```
/*
 * @ 2003 Lemom - UnB
 */

package lemom.wv.client.services;

import lemom.wv.client.xml.XMLMessage;

/**
 * Defines a model for all requests (Presence, Message, Group Features and Shared Content).
 * <br>
 * Default protected instance variables (<code>XMLMessage</code>, <code>Session</code> and
 * <code>WVConnection</code>) are provided for subclassing use.
 */

public abstract class AbstractRequest {

/**
 * Holds session attributes
 */
   protected Session session;

/**
 * XML message associated to this request
 */
   protected XMLMessage message;

/**
 * Connector associated to this request
 */
   protected WVConnection con;

/**
 * AbstractRequest constructor, creates a new request with the given connection and gets the
 * <code>Session</code> instance
 * @param con the connection
 */

   public AbstractRequest(WVConnection con)
   {
          session = Session.getInstance();
          this.con = con;

   }

/**
 * AbstractRequest constructor does nothing
 */
   protected AbstractRequest()
   {

   }

}
```

Apêndice L: WVConnector.java

```
/*
 * @ 2003 Lemom - UnB
 */

package lemom.wv.client.services;

import javax.microedition.io.*;
import java.io.*;

/**
 * This class makes the communication between WV Client and the WV SAP, sending XML by HTTP with all
 * the requests from the other classes.
 */
public class WVConnector implements WVConnection {
   private String url;

/**
 * WVConnector constructor, sets the WV SAP's URL used to establish the communication with the WV
 * Client.
 * @param url WV SAP's URL
 */
   public WVConnector(String url)
   {
          this.url = url;
   }

/**
 * Establishes the HTTP connection between the WV Client e the WV SAP @param message the
 * <code>XMLmessage</code> (converted to <code>String</code>)  to be sent as HTTP data @return a
 * <code>String</code> which is SAP's response to the current transaction
 */
   public String communicate(String message)
   {
          StringBuffer response = new StringBuffer();

          try {
             HttpConnection hc = (HttpConnection)Connector.open(url, Connector.READ_WRITE);
             hc.setRequestMethod( HttpConnection.POST );
             DataOutputStream dos = hc.openDataOutputStream();
             dos.write(message.getBytes(), 0, message.getBytes().length);
             DataInputStream input = new DataInputStream( hc.openInputStream() );
             int ch;
             while ( ( ch = input.read() ) != -1 ) {
                    response.append((char)ch);
         }
             dos.close();
             hc.close();
             input.close();
           } catch (Exception err) {
             System.out.println(err);
             err.printStackTrace();
           }

          return response.toString();
   }

}
```

Apêndice M: WVConnection.java

```
/*
 * @ 2003 – Lemom - UnB
 */

package lemom.wv.client.services;

public interface WVConnection  {

/**
 * Establishes the connection between the WV Client e the WV SAP @param message the message be sent
 * @return a <code>String</code> which is the response to the current transaction
 */
   public String communicate(String message);

}
```

Apêndice N: Session.java

```
/*
 * @ 2003 Lemom - UnB
 */
package lemom.wv.client.services;

/**
 * This class contains all session attributes used from logging in through logging out  from the Wireless Village
 * Server. <br><br>
 * Only one instance of this class is allowed at any time. The active instance (or a new one, in case there isn't
 * any) can be retrieved using the static method <code>getInstance</code>,  and all session values can be set or
 * fetched using the corresponding <code>set</code> or <code>get</code> methods.
 */
public class Session {

   private String sessionID = null;
   private int loginStatus = 0;
   private static Session sessionInstance = null;
   private String statusDescription = null;

/**
 * Session constructor does nothing
 */
   protected Session()
   {
   }

/**
 * Creates the session or retrieves the active instance in case there is one.
 * @return the <code>Session</code> instance.
 */
   static public Session getInstance() {

         if (sessionInstance == null)
            sessionInstance = new Session();
         return sessionInstance;
   }

/**
 * Gets the session ID.
 * @return a <code>String</code> specifying the session ID.
 */
   public String getSessionID()
   {
         return sessionID;
   }

/**
 * Sets the session ID
 * @param SessionID the session identity number as a <code>String</code>.
 */
   public void setSessionID(String sessionID)
   {
         this.sessionID = sessionID;
   }
```

```
/**
 * Sets the login status
 * @param status an <code>int</code> representing the login status.
 */
   public void setLoginStatus(int status)
   {
         loginStatus = status;
   }

/**
 * Gets the login status
 * @return an <code>int</code> specifying the login status.
 */
   public int getLoginStatus()
   {
         return loginStatus;
   }

/**
 * Sets the description of the login status
 * @param description the session status description.
 */
   public void setStatusDescription(String description)
   {
         statusDescription = description;
   }

/**
 * Gets the login status description.
 * @return a <code>String</code> describing the session status.
 */
   public String getStatusDescription()
   {
         return statusDescription;
   }
```

Apêndice O: XMLMessage.java

```
/*
 * @ 2003 Lemom - UnB
 */
package lemom.wv.client.xml;

import org.kxml.*;
import org.kxml.io.*;
import org.kxml.kdom.*;
//import org.kxml.parser.*;
import java.io.*;

/**
 *  Message containing a XML to be sent to the Wireless Village server
 */
public class XMLMessage extends Document {

   Element sessionDescriptor,
           transaction,
           transactionDescriptor,
           transactionContent;

/**
 * XMLMessage constructor, creates the main XML elements
 * (sessionDescriptor, transaction, transactionDescriptor, transactionContent)
 */
   public XMLMessage() {
          sessionDescriptor = this.createElement(Xml.NO_NAMESPACE, "SessionDescriptor");
          transaction = this.createElement(Xml.NO_NAMESPACE, "Transaction");
          transactionDescriptor = this.createElement(Xml.NO_NAMESPACE, "TransactionDescriptor");
          transactionContent = this.createElement("http://www.wireless-village.org/TRC1.1",
"TransactionContent");
   }

/**
 * Inserts the Session attributes into the corresponding elements
 * @param sessionValues Session attributes (SessionType and SessionID)
 */
   public void setSessionDescriptor(String sessionValues[]) {
          Element e = null;

          e = insertElement( "SessionType", sessionValues[0]);
          sessionDescriptor.addChild(Xml.ELEMENT, e);
          if (sessionValues[1] != null) {
             e = insertElement("SessionID", sessionValues[1]);
             sessionDescriptor.addChild(Xml.ELEMENT, e);
          }
   }

/**
 * Inserts the Transaction attributes into the corresponding elements
 * @param transactionValues Transaction attributes (TransactionMode, transactionID and Poll)
 */

   public void setTransactionDescriptor(String transactionValues[]) {
          Element e = null;

          e = insertElement("TransactionMode", transactionValues[0]);
          transactionDescriptor.addChild(Xml.ELEMENT, e);
```

```
            if (transactionValues[1] != null) {
               e = insertElement("TransactionID", transactionValues[1]);
               transactionDescriptor.addChild(Xml.ELEMENT, e);
            }

            // else feito para acomodar o caso do polling, onde o TransactionID é EMPTY
            else {
              e = createElement(Xml.NO_NAMESPACE, "TransactionID");
              transactionContent.addChild(Xml.ELEMENT, e);
            }

            if (transactionValues[2] != null) {
               e = insertElement("Poll", transactionValues[2]);
               transactionDescriptor.addChild(Xml.ELEMENT, e);
            }
      }

   /**
    * Inserts the corresponding value of an element
    * @param tag the element corresponding tag
    * @param value the element corresponding value
    * @return the resulting <code>Element</code>
    */
      public Element insertElement(String tag, String value) {
            Element e = null;
            e = this.createElement(Xml.NO_NAMESPACE, tag);
            e.addChild(Xml.TEXT, value);
            return e;
      }

   /**
    * Creates a sub-element that will take part of a more complete content to be inserted in a  father node. <br>
    * The complete content can be generated by appending all the sub-elements.
    * @param tag the element corresponding tag
    * @param value the element corresponding value
    * @return the resulting <code>Element</code> converted to <code>String</code>
    */
      public String createSubElement(String tag, String value) {

            return insertElement(tag, XMLUtils.decode(value)).toString();
      }

   /**
    * Inserts the Transaction content into the corresponding element
    * @param tag name of the current transaction
    * @param value content of the current transaction
    */
      public void setTransactionContent(String tag, String value) {

            Element e;
            e = insertElement(tag, value);
            transactionContent.addChild(Xml.ELEMENT, e);
      }

   /**
    * Override of method setTransactionContent for a empty content tag
    */
      public void setTransactionContent(String tag) {

            Element e;
```

```
            e = createElement(Xml.NO_NAMESPACE, tag);
                        transactionContent.addChild(Xml.ELEMENT, e);
    }

/**
 * Builds the XML message based on the hierarchy described on the WV specification
 */
    public void wrapMessage()  {

            Element e = null,
            session;
            transaction.addChild(Xml.ELEMENT, transactionDescriptor);
            transaction.addChild(Xml.ELEMENT, transactionContent);
            session = this.createElement(Xml.NO_NAMESPACE, "Session");
            session.addChild(Xml.ELEMENT, sessionDescriptor);
            session.addChild(Xml.ELEMENT, transaction);
            e = this.createElement("http://www.wireless-village.org/CSP1.1", "WV-CSP-Message");
            e.addChild(Xml.ELEMENT, session);
            this.addChild(0, Xml.ELEMENT, e);
    }
```

```
/*
 * © 2003 Lemom - UnB
 */
package lemom.wv.client.xml;

import org.kxml.*;
import org.kxml.io.*;
import org.kxml.kdom.*;
import org.kxml.parser.*;
import java.io.*;
import java.util.Vector;

/**
 * This class contains the Utils to manipulate a XML message
 */
public class XMLUtils {

/**
 * Converts a <code>String</code> into XML tags using scape caracters
 * @param cooked <code>String</code> to be converted
 * @return the resulting XML tags
 */
   public static String decode (String cooked) {

      StringBuffer result = new StringBuffer ();
      int i0 = 0;
      while (true) {

         int i1 = cooked.indexOf ('&', i0);
         if (i1 == -1) break;
         result.append (cooked.substring (i0, i1));
         int i2 = cooked.indexOf (';', i1);
         if (i2 == -1) {
            result.append ("&amp;");
            i0 = i1+1;
            break;
         }

         String decode = cooked.substring (i1 + 1, i2);
          if (decode.equals("amp")) {
                    i1 = i2 + 1;
                    i2 = cooked.indexOf(';', i1);
                    decode = cooked.substring(i1, i2);
          }

         if (decode.startsWith ("#x"))
            result.append ((char) Integer.parseInt (decode.substring (2), 16));
         else if (decode.startsWith ("#"))
            result.append((char) Integer.parseInt (decode.substring (1)));
         else if (decode.equals ("lt"))
            result.append ('<');
         else if (decode.equals ("gt"))
            result.append ('>');
         else if (decode.equals ("quot"))
            result.append ('"');
         else if (decode.equals ("apos"))
            result.append ('\'');
         else throw new RuntimeException ("illegal character entity: &"+decode+";");
```

```
            i0 = i2+1;
        }
        result.append (cooked.substring (i0));
        return result.toString ();
    }

/**
 * Gets the content of a determined element belonging to a XML message
 * @param message the XML message converted to <code>String</code>
 * @param tag the element to searched in the message
 * @return a <code>Vector</code> containing the contents of all found elements
 */
   public static Vector getNodeValue(String message, String tag)
   {
           Vector response = new Vector();
           try {
              InputStreamReader isr = new InputStreamReader(new
                    ByteArrayInputStream(message.toString().getBytes()));
              XmlParser parser = null;
              parser = new XmlParser(isr);
              ParseEvent event;
              while((event = parser.read()).getType() != Xml.END_DOCUMENT) {
                    if (event.getType() == Xml.START_TAG) {
                       StartTag sTag = (StartTag) event;
                       if (!sTag.getName().equals(tag))
                          continue;
                       event = parser.read();
                       if (event.getType() == Xml.TEXT) {
                           response.addElement(event.getText());
                       }
                    }
              }
           } catch (IOException e) {
              e.printStackTrace();
           }
           return response;
   }

/**
 * Return the number of times that the <tag> appears in the XML message
 */
   public static int getNodeTags(String message, String tag) {

           int response = 0;
           try {
                    InputStreamReader isr = new InputStreamReader(new
                    ByteArrayInputStream(message.toString().getBytes()));
              XmlParser parser = null;
              parser = new XmlParser(isr);
              ParseEvent event;
              while((event = parser.read()).getType() != Xml.END_DOCUMENT) {
                    if (event.getType() == Xml.START_TAG) {
                                StartTag sTag = (StartTag) event;
                                if (!sTag.getName().equals(tag))
                                   continue;
                                response++;
                       }
              }
           } catch (IOException e) {
```

```
            e.printStackTrace();
        }
        return response;
    }
}
```