FUNDAÇÃO DE ENSINO “EURÍPIDES SOARES DA ROCHA”
CENTRO UNIVERSITÁRIO “EURÍPIDES DE MARÍLIA” - UNIVEM
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

ANDRÉ LUIS GUIMARÃES CRUZ

UM FRAMEWORK JAVA PARA DISPOSITIVOS WIRELESS

MARÍLIA
2005

ANDRÉ LUIS GUIMARÃES CRUZ

UM FRAMEWORK JAVA PARA DISPOSITIVOS WIRELESS

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Ciência da Computação do Centro Universitário Eurípides de Marília, mantido pela Fundação Eurípides Soares da Rocha para a obtenção do título de Bacharel em Ciência da Computação.

Orientador:
Prof. Dr. Marcos Luiz Mucheroni

Marília
2005

ANDRÉ LUIS GUIMARÃES CRUZ

UM FRAMEWORK JAVA PARA DISPOSITIVOS WIRELESS

Banca examinadora do Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Ciência da Computalção da UNIVEM/F.E.E.S.R., para obtenção do Título de Bacharel em Ciência da Computação.

Resultado: _____(_____,_____)

ORIENTADOR: Prof. Dr. Marcos Luiz Mucheroni _____

1º EXAMINADOR: Prof. MSc. Ricardo Petruzza do Prado _____

2º EXAMINADOR: Profª Drª Ana Paula Piovesan Melchiori Peruzza _____

Marília, 30 de Novembro de 2005.

A todos aqueles que tomaram parte
no desenvolvimento da minha vida acadêmica.

AGRADECIMENTOS

A Deus.
A meus pais e irmãos que me deram o apoio necessário.
A Adriana que suportou bravamente meu mau humor.
A meu filho Paulo, pedindo desculpas pela ausência.

CRUZ, André Luis Guimarães. Um framework Java para dispositivos wireless. 2005. 73 fls. Trabalho (Bacharelado em Ciência da Computação) - Centro Universitário "Eurípides de Marília", "Fundação de Ensino Eurípides Soares da Rocha", Marília, 2005.

RESUMO

O presente trabalho apresenta uma discussão à respeito da tecnologia Java para confecção de aplicações para dispositivos wireless (Celulares e PDAs), visando a utilização de Serviços Web, através do protocolo SOAP e do uso de XML, como meio de transporte de dados padrão para a Internet.

Palavras-chave: Dispositivos Wireless, Framework, Serviços Web, Java.

CRUZ, André Luis Guimarães. Um framework Java para dispositivos wireless. 2005. 73 fls. Trabalho (Bacharelado em Ciência da Computação) - Centro Universitário “Eurípides de Marília”, “Fundação de Ensino Eurípides Soares da Rocha”, Marília, 2005.

ABSTRACT

This research discuss about Java technology for wireless devices software design (mobile phones and PDAs) using web services implemented with the SOAP protocol and XML language, as data transfer way for Internet standards.

Key-words: Wireless Devices, Framework, Web Services, Java.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATUARAS

API – Application Programmer Interface (Interface de programadores de aplicação)
CDC – Connected Device Configuration (Configuração de dispositivo conectado)
CLDC – Connected Limited Device Configuration (Configuração de dispositivo limitado conectado)
CGI – Common Graphics Interface (Interface gráfica comum)
CORBA – Common Object Request Broker (Corretor de requisição de objetos comum)
CVM – Compact Virtual Machine (Máquina virtual compacta)
DOM – Document Object Model (Modelo de objetos documento)
DTD – Data Type Definition (Definição de tipos de dados)
E/S – Entrada e Saída
HTTP – Hyper Text Transfer Protocol (Protocolo de transferência de hipertexto)
J2EE – Java 2 Enterprise Edition (Edição empresarial Java 2)
J2ME – Java 2 Micro Edition (Edição micro Java 2)
JNI – Java Native Interface (Interface nativa Java)
J2SE – Java 2 Standard Edition (Edição padrão Java 2)
JCP – Java Community Process (Processo comunitário Java)
JSP – Java Serve Pages (Servidor de páginas Java)
KVM – Kilo Virtual Machine (Máquina virtual kilo)
MIDP – Mobile Information Device Profile (Perfil de dispositivo de informação móvel)
NFS – Network File System (Sistema de arquivos de rede)
OAK – Object Application Kernel (Objeto núcleo de aplicação)
PDA – Personal Digital Assistant (Assistentes digitais pessoais)
RMI – Remote Method Invocation (Invocação de método remoto)
SAX – Simple API for XML (API Simples para XML)
SGBD – Sistema Gerenciador de Banco de Dados
SOAP – Simple Object Access Protocol (Protocolo de acesso simples à objetos)
TCP-IP – Transfer Control Protocol – Internet Protocol (Protocolo de transferência de controle – protocolo internet)
UDDI – Universal Description Discovery and Integration (Descritor universal, descoberta e integração)
URL – Uniform Resource Locator (Localizador uniforme de recursos)
WAP – Wireless Application Protocol (Protocolo de aplicação sem fio)
WSDL – Web Services Description Language (Linguagem de descrição de Serviços Web)
XML – Extensible Markup Language (Linguagem de Marcação extensível)

SUMÁRIO

INTRODUÇÃO

Nas últimas duas décadas ocorreram grandes mudanças de paradigmas alterando profundamente a forma como se faz software. Até o início dos anos 80, os computadores eram máquinas de grande porte e custo alto. Os chamados *Mainframes* dominavam o mercado e as linguagens de programação eram de certa forma simplistas. Usava-se C para o desenvolvimento de Sistemas Operacionais, Cobol para desenvolvimento de aplicações coorporativas, Fortran para aplicações científicas, Pascal para uso genérico, etc. Não haviam muitas camadas entre o Sistema Operacional e o aplicativo final. Dava-se grande ênfase em estruturas de dados graças ao modelo de programação estruturado. Eram disponibilizadas bibliotecas de funções especializadas, mas que de certa forma não eram tão úteis quanto poderiam ser, já que normalmente disponibilizavam funções para o trato com os dispositivos e com o sistema operacional deixando as regras de negócios dos sistemas para serem tratadas completamente pelo programador. A programação era feita em blocos. Os dados e a forma como eram tratados estavam separados o que dificultava a manutenção de sistemas.

Embora as técnicas que estão atualmente em uso tenham se originado nos anos 70, sendo que algumas até mesmo nos anos 60, passaram a ser utilizadas de maneira definitiva nos anos 80, isso graças a popularização do micro-computador,

que aumentou a demanda por *softwares*, e conseqüentemente, gerou a necessidade de baratear seu custo para se atingir uma gama maior de usuários. Os *softwares* passaram a ser desenvolvidos utilizando linguagens mais especializadas. O paradigma de programação deixou de ser estruturado e à partir da criação da orientação para objetos, que possibilitava entre outras coisas a reutilização de *software* e o feitio de componentes. Por exemplo, durante a década de 80, os sistemas gerenciadores de bancos de dados (SGBDs) entraram no cenário do *software*. Nesse ponto o programador já deveria compreender além do funcionamento da linguagem, as normas e procedimentos para acessar os dados contidos pelos SGDBs.

O surgimento do paradigma orientado a objetos e do conceito de classes de objetos, agrupando num mesmo local os dados e a forma como eles são tratados, melhoraram dois princípios fundamentais no feitio de *softwares*: a manutebilidade e a reusabilidade. Os *softwares* passaram a ser vistos como objetos interagindo dentro de um ambiente. A manutebilidade foi facilitada graças a compartimentalização dos *softwares*.

A reusabilidade, entretanto, foi um conceito mais aprofundado pelo paradigma orientado a objetos. Agora, é possível construir classes de objetos que tratem de um problema em sí. Tanto pode-se facilitar o feitio de *software* pela utilização de objetos gráficos, quanto criar objetos que tratem de um problema

comum à um tipo de aplicação. Nas linguagens como o Smaltalk e C++ é possível criar classes de objetos ou gabaritos que gerem o esqueleto de uma aplicação que trate de relatórios, ou que facilitem o acesso a bancos de dados. Isso permite que o programador utilize essas classes para criar objetos customizados para suas aplicações com o mesmo fim.

Nos anos 90 insere-se no contexto global uma nova forma de comunicação: a Internet. Com suas origens na guerra fria, passou de uma rede de uso militar para forma de comunicação entre institutos de pesquisa e universidades até chegar a uma rede de comunicação de uso geral, em escala global, tomando a forma que possue hoje. Através da Internet é possível comunicar-se com o mundo quase que instantaneamente. Isso modificou a forma como a sociedade e os aplicativos interagem. Os novos conceitos de comunicação de redes de computadores agora passaram a ser obrigatórios e foram adicionados às linguagens de programação junto com o aumento de seu uso. Os gabaritos de programas deveriam ser capazes de comunicar-se numa arquitetura TCP-IP. Os sistemas tornavam-se distribuídos, à medida que as empresas se davam conta da necessidade de comunicação on-line e crescia a demanda por processamento. Novas linguagens para atender essa demanda foram criadas. Os sistemas não estavam mais restritos às janelas de texto. Navegadores deveriam ser utilizados como clientes genéricos. O HTML englobava novas possibilidades tais como a confecção de formulários aumentando a

dinamicidade do conteúdo.

O Java foi criado nesse contexto, com a promessa de ser a linguagem que a integraria a Internet. Algo que não aconteceu da maneira prevista, mas criou tecnologias como os Applets Java, os Servlets e finalmente o JSP, que integra a tecnologia Java para a rede através de clientes genéricos como os navegadores. Outra grande vantagem da linguagem é a independência de plataforma, que facilitaria sua utilização por todos.

A construção dos *softwares* compatíveis com as modernas tecnologias em termos de usabilidade, confiabilidade e conectividade demanda por funcionalidades complexas que não estão disponíveis numa linguagem de programação estrututrada, sendo que o único caminho possível para conseguir suprir essa alta demanda de *software* é a reutilização de código (JACOBSON et al, 1997). A programação de um *software* deixou de ser uma arte de programadores e passou a ser a forma como se integra componentes padronizados, normalmente desenvolvidos por terceiros. O ambiente de programação passou de um editor de textos para um ambiente visual que disponibiliza objetos para serem combinados.

Nesse contexto, a utilização de *frameworks* vêm se firmando como um dos principais paradigmas do desenvolvimento de *software*. A reusabilidade de um *framework* é o ponto chave dessa discussão, além, é claro das definições de padrões e gabaritos que ele propicia para o desenvolvedor.

Noutra ponta do desenvolvimento tecnológico estão os dispositivos móveis *wireless* (sem fio). A utilização de PDAs e Celulares conectados à redes sem fio tornam possível o acesso a informação *on-line* de qualquer lugar com cobertura de uma rede móvel. No século passado, acreditava-se que a Televisão seria a o centro da convergência tecnológica, posteriormente, acreditou-se ser o computador, e hoje acredita-se que será o celular. O nível de sofisticação tecnológica é tal que os telefones celulares estão englobando cada vez mais novas tecnologias e um telefone agora serve para conversar, fotografar, ouvir música, consultar o banco de dados da empresa, ler e-mails, e até mesmo verificar a taxa de glicemia de uma pessoa.

Os serviços para computação móvel prometem ser um grande mercado para os próximos anos. Da mesma forma que o "*boom*" do computador pessoal na década de 80 e 90 propiciou a formação das grandes empresas de *software* de hoje, os celulares, da mesma forma que o de computadores, prometem tornar-se mais uma comódite e o grande diferencial será o *software*.

O objetivo desse trabalho é discutir o desenvolvimento de aplicativos através das modernas técnicas de orientação para objetos, utilizando-se de Serviços Web tendo como clientes dispositivos móveis. Os conceitos pertinentes ao desenvolvimento estão apresentados e separados por tópicos e de acordo com a tecnologia utilizada.

O primeiro capítulo define os conceitos de sistemas distribuídos dando ênfase especial aos sistemas cliente servidor.

O segundo capítulo apresenta a tecnologia Java utilizada para o desenvolvimento da Web, apresentando também sua versão para dispositivos móveis o J2ME.

O terceiro capítulo abordas os serviços Web mostrando seu funcionamento e as principais tecnologias envolvidas na prestação de serviços Web.

O quarto capítulo introduz o conceito de *framework* e verifica algumas abordagens utilizadas nesses trabalho, apresentando o modelo definido e mostrando alguns resultados.

CAPÍTULO 1 - CONCEITOS DE SISTEMA DISTRIBUÍDOS

Os sistemas distribuídos são evoluções dos computadores paralelos e das redes de computadores. A principal característica de um sistema distribuído é a descentralização do controle. Eckhouser Jr. (1978) define um sistema distribuído é formado por um conjunto de módulos, compostos por, pelo menos, processador e memória, fracamente acoplados através de um subsistema de comunicação, de forma que o hardware deva oferecer facilidades de comunicação entre os processos, que por sua vez, devem cooperar entre sí, sem a definição de um controle centralizado, possibilitando a execução de aplicações. Peebles e Manning (1978), dão continuidade a esse paradigma afirmando que nenhum processador poderá coordenar os demais e todos deverão interagir harmoniosamente.

Os primeiros sistemas de redes de computadores eram centralizados, um único computador era o responsável por todos os recursos da rede era quem disponibilizava os serviços. Bons exemplos disso são os *Mainframes* com seus terminais e o MULTIX da General Eletric, que apesar de nunca ter saído da prancheta deu aos sistemas modernos as bases para programação concorrente e as redes de computadores.

A evolução e padronização das redes de computadores, auxiliado pelo barateamento de custos das máquinas possibilitaram uma redução dos custos de

comunicação entre computadores, gerando usuários ávidos por mais serviços. Essa necessidade de atendimento aos usuários forneceu a motivação para interligar os computadores e compartilhar o maior número possível de recursos como impressoras e sistemas de arquivos, além de interfaces para acesso a computadores remotos.

No entanto, devido ao alto grau de conhecimento técnico necessário para operar esses novos recursos, embora possibilitem seu acesso direto, exigem um grande conhecimento à respeito das tecnologias envolvidas. Isso gerou a motivação para novas pesquisas objetivando mascarar para o usuário a existência desses recursos, que deveriam ser transparentes para o usuário. Por exemplo, o NFS (Network File System), compartilham recursos de armazenamento em disco rígido em redes UNIX e diponibiliza o conteúdo de diretórios como se fosse um diretório local, completamente tranaparente para o usuário.

Embora centralizada esta arquitetura permitia o uso de recursos em *software* ou hardware, que podiam ser vistos como uma forma de serviços (programas aplicativos acessíveis a partir de uma máquina que centralizava o ambiente), embora estes serviços não incluissem os conceitos atuais de metadados (por exemplo, ambientes com XML e DTD) e acesso via a Web, que definem o conceito chamado de Serviços Web ou Web-Services.

Esses sistemas já permitem certa independência geográfica, podendo uma

aplicação utilizar recursos de sistemas disponibilizados à grandes distâncias bastando para isso uma conexão de rede.

A figura 1 mostra um esquema das redes centralizadas, com um único computador disponibilizando os serviços e recursos de hardware.

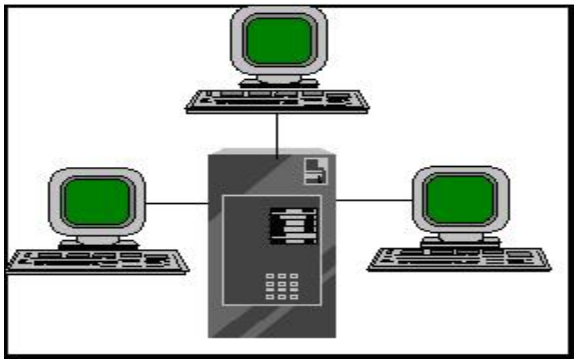

Figura 1: Rede de computadores centralizada

A evolução das redes de computadores permitiu a grande redução de custos de comunicação e hardware, aumentando consideravelmente seu crescimento.

Com a necessidade cada vez maior de atender as necessidades dos usuários por requisições de serviços e recursos, precisava-se de alguma forma ampliar a distribuição e o compartilhamento destes recursos. Assim, começaram a interligar computadores para que juntos pudessem disponibilizar um número maior de benefícios.

1.1 Sistemas Distribuídos Tipo Cliente Servidor

Num primeiro momento, Cliente-Servidor definia uma arquitetura de *software* que descrevia o processamento entre dois programas. O cliente era a aplicação que solicitava o serviço, e o servidor a aplicação que executava o serviço. Um bom exemplo disso é o acesso feito a um Sistema de Gerenciamento de Banco de Dados (SGBD) local. Atualmente, esse conceito foi expandido para processamento cooperativo distribuído, onde clientes e servidores são relacionamentos entre componentes.

A arquitetura Cliente-Servidor, tal como os sistemas distribuídos, compartilha os recursos do computador, sendo que no mínimo devem existir três componentes básicos: o cliente, o servidor e um meio para acessá-lo. Isso pode ser expandido de forma a um cliente utilizar recursos de vários servidores, ou ainda, servidores de um dado recurso ou serviço podem ser clientes de outro servidor, disponibilizando os serviços em camadas.

Dessa forma, numa arquitetura cliente servidor é possível que se construam redes de vários tamanhos, permitindo a customização de hardware e a utilização de recursos compartilhados, além da colaboração entre as partes permitir que o processamento seja dividido em duas ou mais máquinas. As principais desvantagens desse sistema estão na lógica aplicada aos programas que devem

levar em conta a complexidade advinda da divisão de processamento e recursos, de forma a não sobrecarregar o servidor com múltiplas chamadas, o que acarretaria em demora no tempo de execução dos serviços prestados pelo servidor, nem deixar servidores ociosos, o que aumenta o custo de implementação dos sistemas.

1.2 Arquitetura Cliente Servidor

A arquitetura cliente/servidor é um modelo distribuído de computação definida por um paradigma que descreve as relações entre dois programas de computadores baseado em uma seqüência de requisições e respostas enviadas respectivamente do cliente para o servidor e do servidor para o cliente.

Esse tipo de funcionamento é bastante interessante numa rede como a Internet, já que não é necessário manter uma conexão permanente entre o cliente e o servidor. Toda a comunicação é controlada por pedidos de conexão e respostas a esses pedidos. A arquitetura cliente/servidor veio, de certa forma, substituir o antigo modelo de computação centralizada, em que poderosos *mainframes* atendiam a uma série de terminais ditos "burros", pois limitavam-se à exibição e entrada de dados. Os computadores rodando os programas clientes não se limitam à mera exibição de dados na tela, mas participam do processamento destes.

O paradigma de programação distribuída através da separação das aplicações entre servidores (aplicações que disponibilizam algum serviço) e clientes (aplicações que usam esses serviços) foi a arquitetura de distribuição predominante nos anos 1990. Um dos seus atrativos é o aumento da confiabilidade (a falha de uma máquina não necessariamente inviabiliza a operação do sistema como um todo) e a redução de custos (são mais baratos que os *mainframes*).

As aplicações clientes e servidores são programas executados em máquinas distintas, que trocam informação através de uma rede de computadores. Para que os serviços possam ser solicitados, a aplicação cliente deve conhecer quem fornece o serviço (o endereço da aplicação servidora) e qual o protocolo pré-estabelecido para realizar a solicitação.

Um exemplo deste tipo de arquitetura pode ser visto na figura 2:

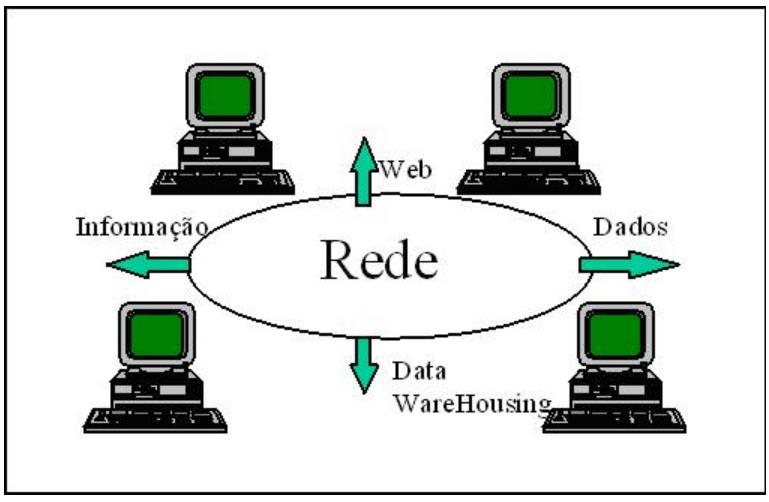

Figura 2: Rede com computadores oferecendo serviços.

É possível verificar que há uma infinidade de serviços que podem ser oferecidos, dando ao ambiente computacional forma flexível, interoperável e escalar, possibilitando a disponibilidade de novos recursos e serviços ao usuário final com certa transparência.

A arquitetura Cliente-Servidor ainda pode ter diversas configurações como servidores de arquivos, servidores de banco de dados, servidores de web entre outros. Todas essas divisões garantem um melhor gerenciamento e facilidade de manutenção dos serviços devido a sua concentração em diferentes locais.

1.2.1 Aplicações Cliente Servidor na WEB.

Uma aplicação desenvolvida para a Web deve fazer uso das tecnologias desenvolvidas para ela. O protocolo HTTP é um protocolo da camada de aplicação do TCP-IP que trabalha através do modelo cliente servidor. Isso permite que todos os serviços disponibilizados nesse protocolo e em protocolos baseados nele possam trabalhar de uma forma bem simples. Ele é um protocolo livre de estados. Não há necessidade de o servidor conhecer seu cliente antes que uma requisição seja realizada, ou que o servidor armazene dados sobre o cliente após a requisição ser despachada. Isso permite o feitio de aplicações que disponibilizam serviços de

resposta imediata, tais como chamadas remotas de procedimento, mas o que foi pensado como uma solução, acabou por gerar a implementação de outros recursos, tais como *cookies* para a identificação dos clientes. O protocolo também não sabe como foi originada a resposta, simplesmente atende às solicitações transmitindo dados.

Numa chamada HTTP um cliente envia uma requisição para um recurso de um servidor e o mesmo retorna uma resposta correspondente ao recurso solicitado. Na chamada estão presentes dados tais como a URL onde o processo servidor está instalada, o método de requisição e a versão do protocolo HTTP utilizada, como pode ser observado na figura 3.

```
GET /index.html?value01=v1&value02=v2 HTTP/1.0
Host: www.myhost.ext:port
User-Agent: Application Name
Accept: XML 1.0
Accept-Language: pt_BR
Accept-Charset: utf-8,isso-8859-1, *

--------

POST /index.html      HTTP/1.0
Host: www.myhost.ext:port
User-Agent: Application Name
Accept: XML 1.0
Accept-Language: pt_BR
Accept-Charset: utf-8,isso-8859-1, *

value01=v1&value02=v2
```

Figura 3: Cabeçalhos de chamadas HTTP utilizando os métodos GET e POST.

As respostas do protocolo seguem o mesmo padrão, entretanto, no corpo da mensagem está disponibilizado o conteúdo a ser retornado, normalmente na forma

de hipertexto.

Os métodos mais comumente utilizados em requisições HTTP são POST e GET, sendo que seu formato apenas varia na forma como os parâmetros são passados. O método GET possui restrição quanto ao tamanho da mensagem e envia os dados requeridos através da URL. O método POST, por sua vez, não possui restrição quanto ao tamanho da mensagem e os dados são enviados junto ao corpo da requisição.

1.3 Ambiente WEB e Computação Wireless

A utilização de dispositivos *wireless* se dá sempre do lado do cliente. Isso ocorre devido à pouca capacidade computacional de dispositivos sem fio e às muitas limitações de memória que são típicas dessa arquitetura. A conectividade dos dispositivos *wireless*, em termos computacionais vem dos anos 90, quando se começou a utilizar telefones celulares junto à *modems* de notebooks e PDAs (Personal Digital Assistant). Nessa época o Pentium 233 era o topo da tecnologia de computadores pessoais padrão x86 e a capacidade dos PDAs estava limitada ao uso de agendas e coletores de dados.

No fim dos anos 90, foram incorporados sistemas computacionais mais

elaborados a esses dispositivos permitindo sua utilização integração com a Internet, além de permitir que desenvolvedores de aplicações pudessem disponibilizar seus produtos para esse mercado, uma vez que sua programação deixou de ser proprietária passando a seguir metodologias de programação comuns a esses dispositivos. Nesse momento, surge o J2ME como um padrão para a confecção dessas aplicações.

Figura 4: Celulares e PDAs com suporte à Java

A integração desses ambientes com a WEB deu-se de forma sistemática. Todos os fabricantes de telefones celulares começaram a disponibilizar capacidades computacionais para a navegação na Internet. Com isso, houve a necessidade de implementação do protocolo HTTP e as redes móveis passaram a disponibilizar esse serviço. Esses telefones celulares pertencem ao que comumente é chamado de 2G ou tecnologia celular de segunda geração. Mesmo assim, devido à limitações de tamanho do visor, e ao custo de transmissão de dados nessa tecnologia, seu uso

encontra-se limitado.

CAPÍTULO 2 - A LINGUAGEM JAVA E O DESENVOLVIMENTO DA WEB

A linguagem Java teve começou a ser desenvolvida no início da década de 90, sua base veio da linguagem OAK (Object Aplication Kernel) desenvolvida por James Gosling enquanto trabalhava com desenvolvimento de *set-top*-boxes para conexão de TV, vídeo sob demanda e programação de PDAs numa subsidiária da Sun Microsystems (NIEMEYER e KNUDSEN, 2000; LINDEN, 1997).

O Java é uma arquitetura para programação e execução de aplicações formada por uma máquina virtual capaz de interpretar arquivo compilados pelo linguagem Java e fazer referências à bibliotecas (API) para execussão de aplicações. Sua linguagem de programação é uma linguagem de alto nível, orientada para objetos, compilada e independente de plataforma. O Java possue ainda gerenciamento de alocação de memória automatizado, segurança, devido a sua forte tipagem de dados, e suporte a programação concorrente (HORSTMAN e CORNEL, 2002).

A máquina virtual Java é um interpretador para o bytecode (instruções binárias para interpretação pela máquina virtual Java) gerado pelo compilador Java. A API Java é um conjunto de bibliotecas de tempo de execução que fornecem ao desenvolvedor de *software* uma forma padrão de acessar os recursos do sistema,

sendo que devem ser implementadas parte dessa API para cada plataforma onde o Java for portado juntamente com a máquina virtual Java, garantindo dessa forma a independência de plataforma das aplicações.

A API Java 2, é um *framework* baseado na linguagem Java que pode ser subdividido em 3 plataformas (J2EE, J2SE, J2ME), essa divisão feita pelo fabricante tem por base a função e localização da API, como mostrado na figura 5.

A plataforma J2SE define o núcleo de aplicações da plataforma Java. É composta do J2RE que é o ambiente de execução Java e as APIs necessárias para rodar aplicativos Java e o JDK um kit de desenvolvimento de aplicações Java distribuído pela fabricante. A figura abaixo mostra os principais componentes da arquitetura Java.

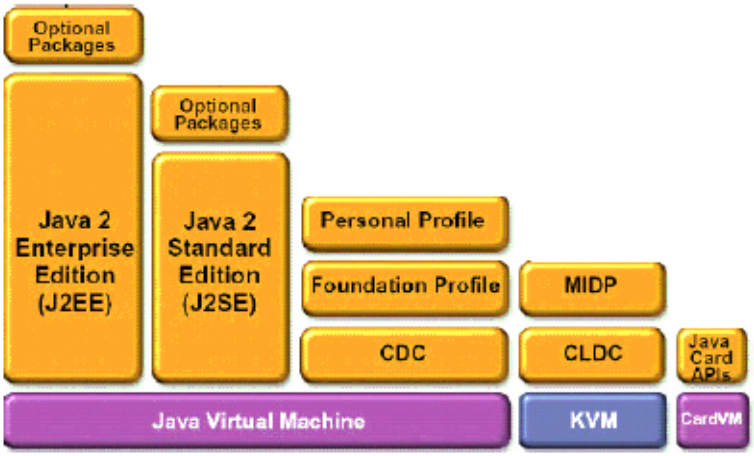

Figura 5: Edições Java e o Java 2 Micro Edition

A plataforma J2EE define os padrões para o desenvolvimento de aplicações

multicamadas para sistemas empresariais distribuídos, com uma ênfase especial em componentes, dando uma nova visão de ambientes, multicamadas, como mostra Armstrong et al (2005):

> "A lógica de aplicação é subdividida em componentes de acordo com sua função ... aplicações J2EE são instaladas em diferentes máquinas de acordo com sua camada no ambiente multicamada que os componentes da aplicação pertencem".

2.1 Java num ambiente cliente servidor na Internet

Desde seu lançamento a linguagem Java teve seu principal enfoque no desenvolvimento de aplicações para a WEB. Os quesitos de portabilidade e independência de arquitetura foram fundamentais para o seu sucesso, e, além disso, possuía uma componente de CGI (Commom Graphics Interface) que tornava qualquer *navegador* um cliente de um aplicativo em potencial: os Applets.

Os Applets são aplicações compiladas e disponibilizadas junto à páginas WEB, que rodam via Máquina Virtual Java atrelados ao navegador como plugin. Na fase de desenvolvimento de um Applet basta ao programador instanciar uma classe filha da classe JApplet.

No entanto, isso não substitui o CGI, pode ser apenas utilizado como cliente

de uma aplicação. A Web passou a ter mais enfoque nos serviços prestados e a utilização de Applets não era suficiente para o desenvolvimento do Servidor. Ciente disso, a Sun Microsystems junto ao JCP (Comunidade de Processo Java) iniciou a produção de uma nova interface de aplicações com base no servidor que deu origem aos Servlets e ao JSP, nesse trabalho representado através do Servidor Apache JakartaTomcat, que é um servidor JSP completo, além de ser um software livre.

Servlets são o formato de apresentação de uma página Web de conteúdo dinâmico usando a tecnologia Java no lado do servidor. Eles funcionam adicionando respondendo a solicitações HTTP, localizando e construindo o conteúdo adequado para a resposta do servidor. Seu funcionamento é semelhante ao de um CGI, mas com a facilidade de não se criar um processo novo para cada solicitação feita por um cliente.

Pelo fato de utilizarem Java, os Servlets aproveitam todos os recursos da tecnologia, como acesso a banco de dados, orientação para objetos e a portabilidade intrínseca à máquina virtual.

O atendimento às chamadas é feito através de linhas de código que criam uma página HTML respondendo à requisição. Quando uma solicitação é feita a um servidor HTTP que implementa Servlets (chamado de *container*), o servidor envia a requisição ao *container* do servelet. O *container*, por sua vez aciona os Servlets

responsáveis pela geração do conteúdo dinâmico e retona para o servidor a página construída.

O grande problema dessa tecnologia é que cabe ao programador realizar toda a confecção da página, impossibilitando o trabalho de um Web Designer. A tecnologia JSP, no entanto resolve esse problema separando a parte funcional (regra de negócios da aplicação e conseqüente programação), da apresentação.

O JSP cria a possibilidade de utilizar tags especiais no HTML que executam chamadas a código Java, separando o trabalho de web designers do de programadores.

2.2 O J2ME

A plataforma J2ME foi lançada pela Sun Microsystems em 1999 na conferência JAVAOne. Trata-se de uma versão reduzida do J2SE para ser utilizada em dispositivos embarcados com limitações de memória, processamento e forma de alimentação, tais como celulares, PDAs e sistemas de navegação.

Seu principal objetivo durante seu desenvolvimento foi a criação de um padrão para dispositivos pequenos. Ele deveria permitir tanto a execução de aplicações desenvolvidas pelo mercado quanto possibilitar a utilização da arquitetura

desenvolvida pelo fabricante do dispositivo, possibilitando que desenvolvedores utilizem o J2ME para desenvolver aplicações personalizadas para usuários, sem que esses fiquem dependentes das aplicações fornecidas apenas pelo fabricante. A figura abaixo mostra a arquitetura genérica do J2ME e algumas arquiteturas desenvolvidas utilizando-se o padrão.

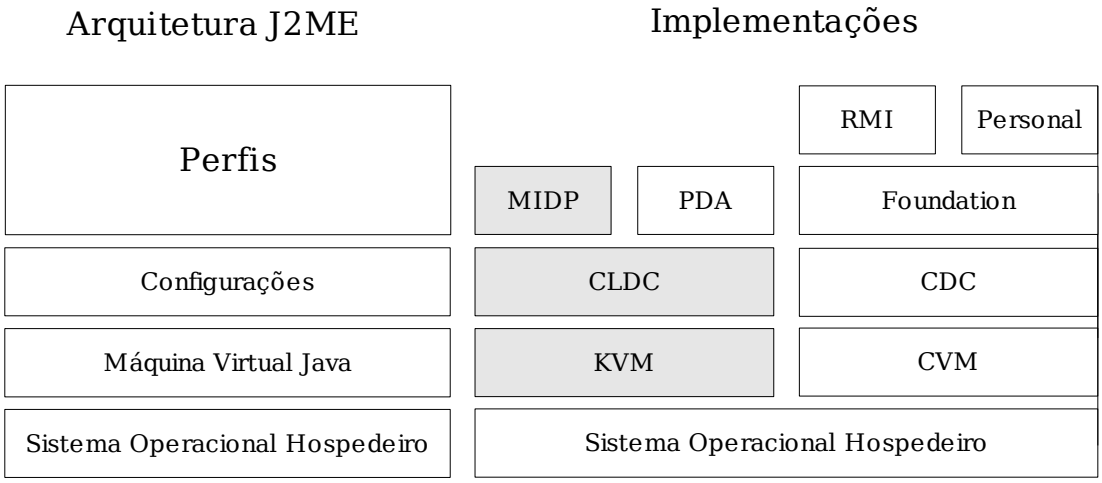

Figura 6: Arquitetura J2ME e suas implementações
Os ítens destacados em tons de cinza mostram as implementações do J2ME utilizadas nesse trabalho.

Um sistema J2ME trabalha sobre um sistema operacional hospedeiro que possibilita o acesso aos recursos do sistema. Sobre ele, uma máquina virtual Java, normalmente implementada pelo fabricante do dispositivo e sob licença da Sun Microsystems, acessa os serviços disponibilizados pelo dispositivo. Na figura 6 podem ser vistas 2 máquinas virtuais, a CVM (Máquina Virtual Compacta) e a KVM (abreviação de Kilo Virtual Machine) que é utilizada em dispositivos móveis.

Devido a enorme variação das características dos dispositivos, como capacidade de processamento, memória disponível e formas de interação com o usuário, o J2ME definiu duas componentes de APIs distintas que se complementam: os perfis e as configurações. Esse conjunto de APIs visa dar suporte à grande diversidade de características em diferentes tipos e modelo de dispositivos.

2.2.1 Configurações

As configurações definem as características da plataforma para um grupo de produtos com base na quantidade de memória disponível e capacidade de processamento, qualificando as características da máquina virtual e do conjunto de bibliotecas necessários. As APIs de configuração mais utilizadas são a CDC e a CLDC. A figura 7 mostra a comparação entre as características de dispositivos capazes de implementar a CDC e a CLDC

	CLDC	***CDC***
Processador	16 bits ou 32bits	32 bits
Memória	160 Kb	2 Mb
Máquina Virtual	KVM	CVM
Banda de Rede	9600 bps ou menos	9600 bps ou menos
Interface Gráfica	Pode existir	Pode existir

Figura 7: Características de dispositivos CLDC e CDC
Retirado de Mahmoud, 2001, p 3 e 4

O CDC foi desenvolvido para sistema que se conectam ocasionalmente a uma rede, normalmente, dipositivos fixos. As características desses dispositivos foram formatadas pelo documento JSR-36 especificado pelo Java Community Process, como descrito: processador de 32 bits; mínimo de 2 MBytes de memória para o Java; funcionalidade completa da máquina virtual CVM, uma máquina virtual completa similar à da J2SE mas com menos requisitos de memória; conectividade com algum tipo de rede.

O CLDC (Connected Limited Device Configuration) possui a característica de ser um denominador comum para a plataforma Java para dispositivos embarcados, mais especificamente, em termos de rede, E/S, segurança, e bibliotecas do núcleo. Foi desenvolvido para utilização com dispositivos que possuam grandes restrições de recursos.

Os dispositivos compatíveis com o CLDC devem possuir processador de 16 ou 32 bits, sendo mais comum à utilização de processadores de 16 bits, pelo menos 160 Kbytes de memória para o Java, deve ter capacidade de alimentação limitada (geralmente por bateria) e deve possuir algum tipo de conexão com uma rede, podendo ser intermitente.

2.2.2 Perfis

Os perfis são bibliotecas para provisão de componentes e bibliotecas que visam atender a um determinado mercado de dispositivos. Trata-se de uma API mínima definida para uma dada configuração. As aplicações escritas em um determinado perfil são portáveis entre dispositivos que suportam esse pefil, permitindo que se faça uma determinada aplicação para ser utilizada tanto em celulares da Motorola e como da Siemens, por exemplo, mas limita a portabilidade da aplicação entre perfis diferentes, ou seja, um determinado aplicativo que pode ser executado num celular dificilmente poderá ser executado em um sistema de navegação automotivo.

As bibliotecas de um perfil estão numa camada acima das configurações. Pode-se citar entre os perfis existentes os seguintes: MIDP, PDA Profile, Foundation Profile, Personal Profile e RMI Profile, sendo que desses, o MIDP é o perfil mais utilizado. Os perfis MIDP e PDA são implementados com base na configuração CLDC e os demais, com base na CDC.

O MIDP (perfil de dispositivo de informação móvel) foi desenvolvido para ser usado com o CLDC e prover APIs para dispositivos móveis (telefones celulares, PDAs e pagers), contendo classes para interface de usuário, armazenamento local e conexão de rede . O MIDP é governado pelo JSR-37.

2.3 CLDC - Connected Limited Device Configuration

"O objetivo do CLDC (configuração de dispositivo limitado conectado) é assegurar a portabilidade e interoperabilidade entre aplicações que são executadas em dispositivos com poucos recursos" (MAHMOUD, 2005, p.15), mantendo dessa forma a portabilidade da máquina virtual Java, ou seja, não possui características adicionais. De fato, o CLDC é a menor implementação para a plataforma Java. Ele requer no mínimo 160 Kbytes de memória para a máquina virtual e suas aplicações, um processador de 16 ou 32 bits (normalmente processadores de 16 bits são utilizados) com no mínimo 25 Mhz de velocidade, capacidade de conexão com algum tipo de rede mesmo que intermitente e normalmente sem fio, e baixo consumo de energia.

Em uma comparação com a máquina virtual Java tradicional, o CLDC possui as seguintes restrições na versão 1.0: não há suporte para números de ponto flutuante, para a finalização de objetos, para interface nativa Java (JNI), para carregadores de classe, para reflexão, e por conseqüência, para a serialização de objetos e invocação de métodos remotos.

O suporte a tratamento de erros é limitado. Quando um erro de tempo de execução ocorre, dependendo da implementação, ou a aplicação é terminada ou o sistema é reiniciado. A versão 1.1 do CLDC trouxe poucos avanços, dando suporte a

dados de ponto flutuante, certa melhoria no tratamento de erros e nas classes Date, TimeZone e Calendar.

O CLDC ainda provê uma biblioteca para acesso a rede. No entanto, os serviços que por ela são disponibilizados dependem da implementação física da rede. O denominador comum e mínimo exigido é o protocolo HTTP, mesmo que parcialmente implementado. Outros serviços como datagramas, sockets, e comunicação serial dependem tanto do dispositivo onde o J2ME está instalado, quanto da arquitetura de rede empregada.

2.4 MIDP – Mobile Information Device Profile

Ao invés de tentar incluir todas os serviços necessários para programação de dispositivos embarcados o MIDP 1.0 e o MIDP 2.0 limitaram-se a um conjunto mínimo de habilidades específicas a áreas absolutamente necessárias de forma a manter a portabilidade, entre outros.

Trata-se de um conjunto de APIs projetado para trabalhar uma camada acima do CLDC. Seu foco está no desenvolvimento de aplicações, ao invés do desenvolvimento de sistemas operacionais. Suas bibliotecas estão relacionadas à programação de aplicações, e, portanto disponibilizam um conjunto de classes

relacionado à interface com o usuário, iniciação de aplicações, armazenamento de dados e tratamento de exceções. As principais funções implementadas pelo MIDP são: controle do ciclo de vida da aplicação (MIDLet), modelo de sinalização de aplicativos, modelo de segurança, rede, armazenamento persistente de dados, som, temporizadores e interface com o usuário.

O MIDP extende o CLDC para prover suporte para o protocolo HTTP, criando facilidades para programação de rede. O modelo de programação HTTP no J2ME é semelhante ao do J2SE, mas com algumas restrições. No MIDP são implementados apenas os métodos POST e GET, o que é suficiente para executar chamadas a servidores Web e utilizar o protocolo SOAP.

2.4.1 MIDLet

As aplicações MIDP tem por base uma super classe chamada MIDlet que define as chamadas para métodos específicos que a máquina virtual executará para carregar, terminar ou suspender uma aplicação. A classe MIDLet é um container que empacota a aplicação disponibilizada através uma arquivo jar, descrito por um arquivo jad. A figura 8 mostra o arquivo jad de um jogo chamado 3D Pool.

Os arquivos jar, são arquivos de bytecodes de aplicações feitos a partir de

vários arquivos .class, agrupados e compactados, normalmente utilizando um compactador do tipo ZIP. Os arquivos jad são descritores do arquivo jar. Tratam-se de arquivos de texto contendo características da aplicação como nome, fornecedor e tamanho, além de algumas especificações a respeito do funcionamento da aplicação, tais como por qual classe iniciar e a versão da aplicação .

```
MIDlet-Name: 3D Pool
MIDlet-Version: 1.2.0
MIDlet-Vendor: www.vstuff.co.uk
MIDlet-Icon: /icon.png
MIDlet-1: 3D Pool, /icon.png, Pool
MicroEdition-Profile: MIDP-1.0
MicroEdition-Configuration: CLDC-1.0
Created-By: Delivery 1.0
Mot-Data-Space-Requirement: 121
Mot-Program-Space-Requirement: 223
MIDlet-Jar-Size: 97515
MIDlet-Jar-URL: 3D_Pool.jar
Nokia-MIDlet-Category: Game
```

Figura 8: Exemplo de um arquivo JAD
Esse arquivo descreve um jogo para celulares: o POOL 3D

No MIDLet está definido o ciclo de vida de uma aplicação MIDP que são representados por 3 estados: Pausado, Ativo e Destruído. Quando um MIDLet é iniciado ele é prontamente colocado no estado Pausado. A figura 9 mostra o funcionamento do ciclo de vida de um MIDLet, identificando os estados e mostrando as chamadas de métodos que são realizadas pela máquina virtual.

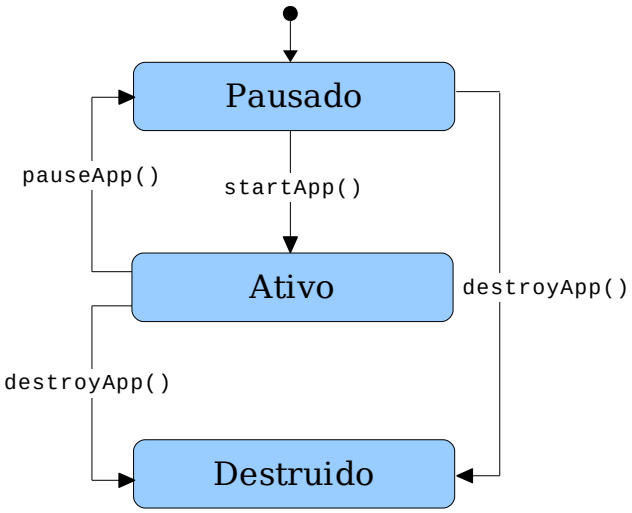

Figura 9: Ciclo de vida de um MIDLet

Uma vez que ele esteja devidamente carregado, é chamado o método startApp(), e a aplicação entra em execução, passando para o estado Ativo. Uma vez ativo, pode-se interromper a aplicação, voltando para o Pausado, chamando-se pauseApp(), por exemplo, no caso de uma chamada de telefone, pausa-se a aplicação para que o telefonema possa ser atendido e então, quando terminado, pode-se retomar a aplicação, através de uma nova chamada de startApp(), ou então, a aplicação ser terminada, chamando-se destroyApp(), que passa a aplicação para o estado Destruído e libera os recursos utilizados para carrega-la. destroyApp() pode ser chamado a partir do estado pausado ou ativo.

```
 /**
  * HelloMidlet
  *
  * Classe de exemplo para identificar os pontos de
  * entrada em uma aplicação MIDP
  */

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class HelloMidlet extends MIDlet implements CommandListener {

  // define o mostrador do dispositivo
  Display dsp = null;
  // Formulario
  Form fmr = null;
  // Comando de saida
  static final Command cmd_exit = new Command( “Sair”, Command.STOP, 2 );

  // Construtor da classe
  public HelloMidlet() {
    if (frm != null){
      fmr = new javax.microedition.lcdui.Form(“Alô”, new Item[]{“Alô”, “mundo!”});
      fmr.addCommand(cmd_exit);
      fmr.setCommandListener(this);
    }
  }

  // toda aplicação midp inicia por aqui
  public void startApp() {
      dsp = Display.getDisplay(this);
      dsp.setCurrent(fmr);
  }
  //caso haja uma requisição, por exemplo, o telefone toque, é chamado esse \
método que suspende a aplição e executa outro serviço. Para voltar usa-se startApp
  public void pauseApp() {
      // nada a fazer
  }
  //Finaliza a aplicação. Boolean unconditional deve ser tratado pelo programador
  public void destroyApp(boolean unconditional) {
      // nada a fazer
  }
}
```

Figura 10: Exemplo de um MIDLet

Na figura 10 pode-se notar ainda que a classe HelloMidlet implementa a interface CommandListener. Essa interface é a responsável pelo gerenciamento de eventos em cada MIDLet. Através do método `setCommandListener`, um escutador de eventos é criado para e são atribuídos a ele os métodos que manipularão o evento gerado. O processo é exemplificado na figura acima que mostra a adição de um comando de saída para um botão exit.

CAPÍTULO 3 - SERVIÇOS WEB

O W3C (2004) define Serviços Web como aplicações autocontidas, com suas interfaces e forma de comunicação baseadas em XML, que descrevem um conjunto de operações acessíveis pela rede e independente de plataforma ou tecnologia utilizada para desenvolve-lo. Já CHAPPEL e JEWLL (2002) definem Serviços Web como uma peça da lógica de negócios localizada em algum local na Internet que é acessível através dos protocolos padrões da Internet.

Serviços podem ser considerados objetos funcionais independentes que podem interagir com outros serviços, desenvolvendo aplicações num conjunto de serviços passíveis de serem publicados, encontrados por ferramentas de busca, combinados e consumidos para gerar os resultados esperados de coleções de serviços autônomos. Eles são componentes que disponibilizam suas funcionalidades (operações e descrição do serviço) através de troca de mensagens descritas no formato XML.

A utilização do padrão XML para a troca de mensagens diminue o acoplamento entre os componentes tornando-os mais flexíveis e simples de se implementar. A reusabilidade do componente também é garantida pelo XML, entretanto a aplicabilidade desse conceito depende mais do modo pelo qual o componente é desenvolvido do que da tecnologia que ele utiliza.

O descritor do serviço é o responsável pela portabilidade e auto contenção dos Serviços Web. Ele possui as informações para interação dos componentes, definido o formato das mensagens para chamada aos métodos, os protocolos de comunicação utilizados pelo serviço e sua localização.

3.1 Arquitetura de Serviços Web

Serviços Web, diferentemente de outras tecnologias, não são acessados por meio de protocolos específicos para modelagem de objetos tais como CORBA, RMI ou IIOP. Utiliza-se para o acesso desses serviços protocolos independentes de plataforma, mais especificamente, SOAP (Protocolo de Acesso a Objetos Simples) e HTTP (Protocolo de Transferência de Hiper Texto).

Um Serviço Web acessa a interface de um componente através de uma mensagem XML padronizada chamada de descrição de serviço (WSDL – Web Service Description Language ou Linguagem de descrição de Serviços Web), onde são definidos o formato das mensagens e seus respectivos tipos de dados e a localização do serviço.

A interface do serviço, por sua vez, empacota a implementação do serviço executando-o no servidor e retornando os dados atualizados para a aplicação

cliente, esse formato de implementação de serviços torna-se independente de plataforma ou hardware.

A arquitetura de Serviços Web se baseia no provedor do serviço, cliente do serviço e servidor de registro. De uma forma geral, as interações são para publicação, busca e execução de operações, como mostrado na figura 11. O provedor do serviço representa a plataforma que hospeda o Serviço Web permitindo que os clientes acessem o serviço.

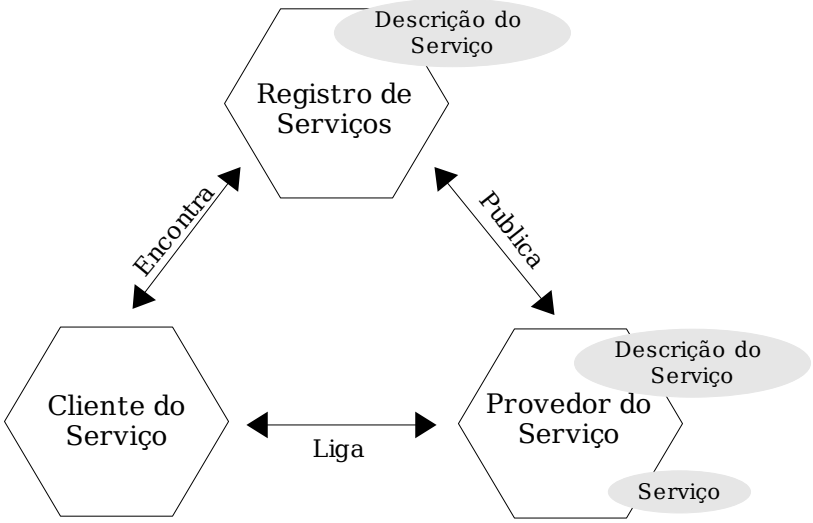

Figura 11: Arquitetura de Serviços Web

SOAP é um protocolo para troca de informações em ambiente distribuído, baseado em definições XML e utilizado para acessar Serviços Web, empacotar as chamadas e retorno aos métodos dos Serviços Web, sendo utilizado, principalmente, sobre HTTP.

WSDL é a linguagem de descrição de Serviço Web baseada em XML, permitindo através dessa definição a possibilidade de descrever serviços e a troca de mensagens, provendo informações necessárias para a invocação do Serviço Web, bem como sua localização, operações disponíveis e suas assinaturas.

A implementação UDDI (Descrição Universal, Descoberta e Integração) corresponde a um regitro de Serviço Web, que provê um mecanismo para busca e publicação Serviços Web contendo informações sobre os serviços e as funcionalidades que eles oferecem, permitindo a associação desses serviços com suas informações técnicas (geralmente definidas usando-se WSDL). A especificação UDDI define uma API baseada em mensagens SOAP, com uma descrição em WSDL do próprio Serviço Web do servidor de registro. O próprio UDDI pode ser entendido também como um Serviço Web.

3.2 Anatomia de Serviços WEB

Antes de qualquer interação ocorrer, um Serviço Web deve ser publicado em um servidor UDDI, para que possa ser encontrado.

Uma aplicação que solicita um Serviço Web (cliente) descreve as características do serviço solicitado e com a ajuda de um provedor de serviços

(Servidor UDDI) localiza o serviço apropriado. Quando o serviço é encontrado as informações na descrição do serviço garantem a interação entre cliente e servidor.

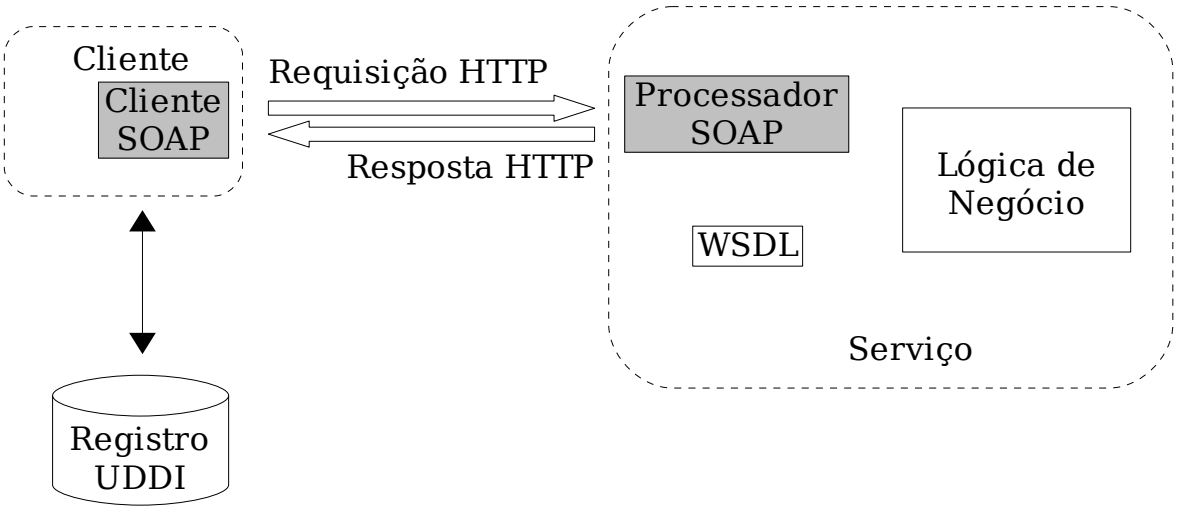

Figura 12: Processo de execução de um Serviço Web

O padrão utilizado para descrição dos serviços é o SOAP (Simple Object Access Protocol ou protocolo de acesso a objetos simples) que garante a comunicação entre cliente e servidor, o UDDI utilizado para localização do serviço e a WSDL. É interessante lembrar que a matriz de todas essas tecnologias é o XML, pois elas são aplicações da linguagem XML.

3.3 XML

A Linguagem XML (Extended Markup Language - Linguagem de Marcação Estendida) foi criado pelo W3 Consortium, mais especificamente pelo grupo de desenvolvedores chamado XML Working Group. Quando foi desenvolvido visava ser uma linguagem altamente usada na Internet, com suporte a uma variedade de aplicações, de fácil escrita e com poucas características. Os documentos XML deveriam possibilitar a leitura e compreensão por pessoas treinadas, não apenas por computadores.

"A linguagem XML é um subconjunto do SGML. Seu objetivo é permitir que um documento SGML genérico possa ser servido, recebido e processado na Web" (BRAY, 2004). A (XML) é um formato utilizado como padrão na WEB, para representar dados dando significado (semântica) a eles. Isso é possível graças à utilização de marcadores[1] (*tags*) e a forma como um documento XML é estruturado. A XML é utilizada para implementar uma classe de documentos chamados documentos XML que descrevem parcialmente o comportamento de programas de computadores que os processam, os marcadores XML codificam uma descrição do que o documento está armazenado através de sua estrutura e forma.

1 Marcadores são elementos expressos entre < e > que definem o significado, ou semântica, de um determinado dado.

3.3.1 Documentos XML

Pela sua definição, um documento XML é um objeto que armazena dados. Cada documento XML possue uma estrutura lógica e uma física. Fisicamente é feito de unidades de armazenamento chamada de entidades. Logicamente, é composto de declarações, elementos, comentários, referências à caracteres e instruções de processamento sendo que todas essas são indicadas explicitamente por marcadores. Os documentos XML devem ser compostos exclusivamente por texto, sendo permitido definir qual o tipo de codificação ele utiliza, aumentando sua portabilidade.

Os documentos XML devem ser produzidos de acordo com uma gramática especificada pelo W3C.

```
<?xml version="1.0" encoding="UTF-8"?>
<Bibloteca Nome=”Dr. Cristiano Altenfender”>
      <Endereço> Av. Higino Muzzi Filho, 529 </Endereco>
      <Regulamento link=”http://www.fundanet.br/biblioteca/regulamento.doc”></<Regulamento>
      <Acervo>
             <Livro Titulo=”Arquitetura de sistemas operacionais” Ano=”1997”>
                    <Autor> Machado, Francis Berenger </Autor>
                    <Autor> Maia, Luiz Paulo </Autor>
                    <Editora> Livros Técnicos e Científicos </Editora>
                    <Emprestado/>
             </Livro>
             <Livro Titulo=”Princípios de sistemas operacionais” Ano=”1980”>
                    <Autor> Guimarães, Célio Cardoso </Autor>
                    <Editora> Campus </Editora>
             </Livro>
      </Acervo>
</Bliblioteca>
```

Figura 13: Exemplo de arquivo XML – Catálogo de biblioteca

Todo documento XML deve iniciar em uma raíz, ou entidade documento. O

documento é dividido em prólogo e elementos. No prólogo dos documentos estão definidas a versão do XML utilizada para compô-lo, a codificação do documento (padrão de caracteres utilizado) e a localização da definição da gramática utilizada, comumente armazenada em um arquivo DTD.

Os elementos de um documento são chamados de *tags* que qualificam o conteúdo dos dados que encapsulam dando significado a ele. As *tags* são classificadas de 3 maneiras: *tags* iniciais, definidas pelo nome da *tag* e, se necessário, pode conter atributos; *tags* finais e *tags* vazias, elementos da forma '<'nome'/>' que são equivalentes a pares '<'nome'></'nome'>' sem conteúdo. As *tags* de um documento XML podem ser aninhadas, mas devem ser iniciadas e terminadas sem se misturarem, sendo que o não cumprimento dessa regra implicará em erro.

Nos documentos DTD estão definidas as regras de validação de *tags* que auxiliam os interpretadores XML a avaliar os dados recebidos, embora sejam opcionais. Um documento XML que utiliza um DTD é definido como válido. Em contraposição, documentos que não fazem uso de DTDs são apenas reconhecidos como bem formatados. A DTD define blocos construtivos de elementos que o documento XML pode usar. Ele define como, quais são e como devem ser produzidos os elementos de um documento

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT Biblioteca (#PCDATA)>
  <!ATTLIST Nome CDATA>
<!ELEMENT Endereço (#PCDATA)>
<!ELEMENT Regulamento (#PCDATA)>
  <!ATTLIST link CDATA>
...
```

Figura 14: Exemplo de arquivo DTD

Outra forma de validar documentos XML é o uso de Esquemas. Os Esquemas foram desenvolvidos visando fornecer novas maneiras de validar documentos XML, em contraposição ao DTD.

3.3.2 XML Schema

Devido às desvantagens da utilização de DTDs, tais como sintaxe diferente de um documento XML, falta de definição detalhada de tipos e dificil compreensão e leitura do arquivo, o W3C propos a utilização de uma aplicação do XML chamada XML Schema Language.

> Os conceitos básicos subjacentes a XML Schema são similares àqueles dos DTDs. A função principal é a mesma: validação ... Um XML Schema define uma hierarquia e tipos de dados rigorosos para os dados e reforça a estrutura do documento e a integridade dos dados. (MILLER JR., 2003).

Numa definição utilizando XML Schema deve-se seguir as restrições definidas pelas recomendações de Estruturas e Tipos de Dados. A recomendação das Estruturas definem os métodos para se escrever as estruturas e as restrições de conteúdos dos documentos. A recomendação de Tipos de Dados define um conjunto de tipos de dados básicos para serem associados com tipos de dados e atributos de elementos. Além disso, o XML Schema dá suporte a espaços de nomes, tipos de dados complexos (registros).

3.3.3 SAX

O SAX é uma interface Java para análise de XML que tem sua principal vantagem na diminuição da quantidade de memória necessários para analisar um documento XML em relação às demais metodologias de análise.

O analisador SAX verifica pequenos conjuntos de documentos XML disparando eventos para cada conjunto analisado, chamando o código para realizar o tratamento do trecho definido através de métodos de retorno. Os retornos habituais do SAX são startDocument, endDocument, startElement, endElement, getEntity e characters.

Seu nome vem de Simle API for XML (API Simples para XML).

1.4 SOAP

SOAP é um protocolo simples baseado no XML que possibilita a troca de informações numa rede. Huang (2003) define o SOAP como um protocolo de empacotamento padrão para mensagens compartilhadas pelas aplicações. Para Chappel e Jewell (2002) SOAP é uma estrutura de encapsulamento padrão para transporte de documentos XML através de várias tecnologias padronizadas da Internet. Em sua especificação pelo W3 Consortium (2002), o SOAP é um protocolo que provê a definição de informações baseadas em XML que pode ser usado para trocar informações estruturadas em uma ambiente distribuído.

O protocolo é uma implementação do XML provê algumas tags para o seu controle. Uma mensagem SOAP contém normalmente 3 partes: o envelope, o cabeçalho (opcional) e o corpo da mensagem

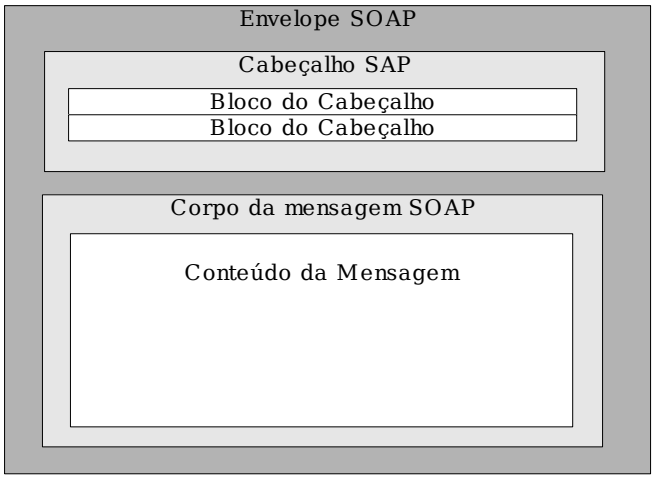

Figura 15: Estrutura de uma Mensagem SOAP

O envelope da mensagem é recipiente superior que engloba a mensagem SOAP por inteiro. As informações contidas no cabeçalho definem como a mensagem deve ser enviada, autenticada e processada. Em seu corpo está a definição da mensagem propriamente dita.

Uma mensagem SOAP é fundamentalmente uma transmissão de um enviador para um receptor, podendo também ser combinadas para implementar modelos como os de requisição e resposta ou chamadas remotas de procedimento.

O Axis é uma implementação do protocolo SOAP e dos processos de requisição de chamada necessários para a chamada remota de procedimentos. Ele é baseado num servidor JSP, que é por onde ele recebe as solicitações, no entanto, é capaz de executar chamadas de aplicações java no servidor. O Axis é o processo servidor utilizado por esse trabalho para atender às requisições de clientes J2ME.

```
<soapenv:Envelope>
    <soapenv:Body>
        <somaResponse soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
            <somaReturn xsi:type="xsd:int">30</somaReturn>
        </somaResponse>
    </soapenv:Body>
</soapenv:Envelope>
```

Figura 16: Exemplo de um documento SOAP

CAPÍTULO 4 - AMBIENTE DE DESENVOLVIMENTO

A conceituação de *frameworks* costuma gerar controvérsias. Isso é devido a sua evolução, que partiu das bibliotecas de acesso a recursos do sistema, como mostrado anteriormente. Esse capítulo define conceitualmente *frameworks* e apresenta a proposta realizada por esse trabalho.

4.1 Algumas observações sobre frameworks

A utilização de *frameworks* é uma técnica de confecção de *software* através da reutilização de objetos. Os *frameworks* são utilizados com sucesso há algum tempo por empresas preocupadas em formas de reutilização de *software*. Johnson (1997) descreve *frameworks* de duas maneiras: “Um *framework* é um projeto reusável de todo ou parte de um sistema que é representado por um conjunto de classes e a forma que essas instâncias interagem*”* e *“*um *framework* é o esqueleto de uma aplicação que pode ser customizado por um desenvolvedor*”.*

Embora, aparentemente, essas definições mostrem-se conflitantes, elas apenas são baseadas e pontos de vista diferentes. A primeira procura descrever como um *framework* é feito e a segunda como é utilizado. Um *framework* está

intimamente ligado ao conceito de reusabilidade. Idealmente, os componentes deveriam ser facilmente conectados, além de prover serviços de forma tal que os desenvolvedores não necessitem saber como ele foi confeccionado, e que o sistema resultante fosse eficiente, de fácil manutebilidade e de alta confiabilidade, mas isso não ocorre dessa maneira. O interesse na reutilização de *software* foi baseado na componentização, mas não foram criados mercados de componentes reusáveis para a grande maioria das ferramentas de programação.

Frameworks, entretando, somente são componentes se o pensarmos como um produto que pode ser vendido. Eles são mais customizáveis que os componentes e possuem interfaces mais complexas, são em sua grande maioria genéricos e altamente customizáveis, e da mesma forma que os componentes em geral propiciam uma diminuição nos custos finais de implementação de aplicações. Os *frameworks* possuem uma relação patriarcal sobre os componentes, uma vez que num *framework* estão definidas as formas de interação entre objetos além de tratamento de erros, e exceções. De fato, componentes e *frameworks* são tecnologias que trabalham juntas para o desenvolvimento de aplicações.

Pode-se considerar *frameworks* como formas de reuso de projeto, similares à templates ou esquemas com a vantagem de estarem atrelados e uma linguagem de programação, não necessitando de ferramentas de alto nível como as outras tecnologias.

De uma maneira geral, *frameworks* são uma forma pratica de expressar projetos reusáveis.

4.2 Bibliotecas utilizadas

Durante o desenvolvimento do projeto foram utilizadas algumas bibliotecas de terceiros para a prover a utilização de xml e soap. A análise de documentos xml é feita pelo por uma implementação da API SAX chamada kxml, e para construção e análise de envelopes SOAP utilizou-se a ksoap. Ambas as bibliotecas foram desenvolvidas pelo projeto Enhydra e são compatíveis com o J2ME. A kxml trabalha com documentos XML manipulando as tags através de um pull parser. A ksoap implementa um gerador e um analisador SOAP para criação e resgate de chamadas a servidores que processem chamadas SOAP. Ela é compatível com as versões 1.0, 1.1 e 1.2 do protocolo.

Para a diminuição do tamanho da biblioteca final foram descartadas partes que não seriam utilizadas pelo *framework* de ambas as bibliotecas. Da kxml foram removidos os itens que davam suporte a WAP (Wireless Aplication Protocol – um protocolo para criação de aplicações em dispositivos móveis) e DOM (Document Object Model – trata-se de modelo de documento em forma árvore baseado nas tags

de um documento escrito em meta-linguagens tais como o HTML ou o XML). Do kxml foi mantido somente o tratamento de analise e criação de envelopes SOAP, sendo que a interface com servidores e outras funcionalidades como tratamento de dados em processadores de 64 bits foram removidas.

Do lado do servidor foi utilizada uma aplicação do servidor Tomcat chamada Axis. Ela é um *framework* para a construção de processadores SOAP. Suas aplicações são desenvolvidas utilizando-se linguagem Java nativa. Para produzir um serviço utilizando-se o Axis basta criar o código da aplicação em Java e disponibilizá-lo (em código fonte) no diretório de aplicações, apenas mudando a extensão do arquivo de .java para .jws (abreviatura de java web service). O Axis ainda cria dinamicamente documentos WSDL para os serviços disponibilizados por ele.

4.3 Modelo do Framework para dispositivos Wireless

O objetivo desse trabalho é possibilitar utilização de dispositivos *wireless* como clientes de Serviços Web. Um serviço é uma requisição de um dispositivo cliente a um servido que a executa e retorna a informação pertinente para o cliente, como mostrado na figura 17.

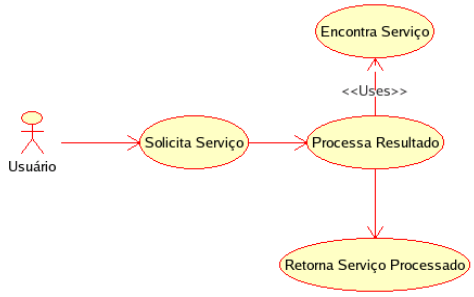

Figura 17: Interação entre clientes e servidores

Um usuário envia uma solicitação para um servidor através de um cliente. O servidor então encontra o serviço que o usuário está solicitando, o processa e envia a resposta do serviço para o cliente.

Procurando tornar isso o mais transparente possível, foram criadas algumas classes que interagem para realizar a chamada para servidores. O diagrama de classes (figura 18) mostra a forma como elas interagem. Todas as classes foram estruturadas utilizando métodos set e get para seus parâmetros visando tornar possível a reutilização de objetos já que o custo de criação e carregamento de um objeto é alto em comparação com o objeto reutilizado quando analisado o tempo de processamento necessário, já que no objeto reutilizado não é necessário executar os procedimentos de carregamento de objetos, além de diminuir os esforços do coletor de lixo da máquina virtual Java.

Na figura 18 está o diagrama de classes do framework. Como todas as requisições para servidores são feitas através da rede, inicialmente foi construída uma classe Link. Nela encontram-se métodos para a análise de URLs, já que o

custo de transmissão de dados é alto e é interessante ter um mínimo de certeza de que o recurso será encontrado pelo dispositivo. Toda vez que uma URL é passada para a classe, ela a separa em partes, definido o esquema, obrigatoriamente http pois é o mínimo denominador comum entre todos os dispositivos que utilizam CLDC/MIDP. O endereço do hospedeiro, o porto ao qual o servidor está atrelado, o caminho para o recurso desejado e parâmetros passados para o servidor.

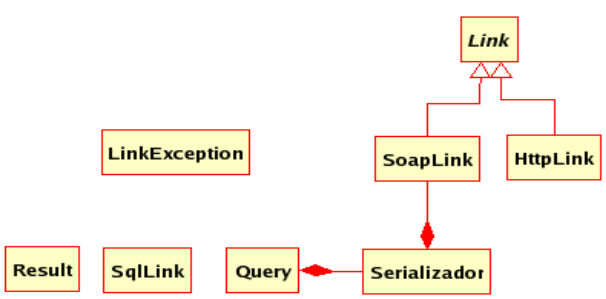

Figura 18: Diagrama de classes do framework

A classe LinkException é responsável pelo tratamento de exceções do *framework*. Nela não há métodos especiais para o tratamento de exceção, apenas torna possível o envio de mensagens ao usuário indicando o que ocorreu com o sistema.

A classe HttpLink é responsável pela comunicação com servidores HTTP. Ela disponibiliza recursos que facilitam a construção de chamadas HTTP, além de implementar chamadas utilizando os métodos GET e POST.

A classe SoapLink define método para fazer chamada a serviços remotos através do protocolo SOAP. Sua implementação é baseada na biblioteca ksoap utilizada como base para a transferência de dados utilizando-se a classe serializador.

Os métodos da classe SoapLink que executam chamadas à servidores SOAP são o Call() e suas sobrecargas. As formas de passagem de parâmetros das sobrecargas variam quanto a utilização de objetos disponibilizados pela biblioteca ksoap. O primeiro é através da utilização de um SoapObject e segundo através da utilização de escritores XML.

A classe Serializador chama serviços fornecidos pelo servidor SOAP escondendo detalhes de implementação da biblioteca ksoap do usuário. O método que executa chamadas para o servidor é call(). A vantagem da sua utilização está na passagem dos parâmetros e do método chamado pelo serviço, um array da classe Object que deve conter as chamadas para serviço disponibilizado.

Não foi implementado chamadas para servidores UDDI, portanto é necessário que o usuário conheça a localização do serviço. Isso foi feito dessa maneira para diminuir o tempo de conexão devido ao alto custo. A chamada a servidores UDDI podem ser facilmente implementadas, bastando para tanto que se defina um servidor UDDI e se crie um interpretador baseado em XML, que pode ser feito em futuras versões.

A classe Serializador propicia facilidades de utilização da classe SOAP Link. Nela é possível acessar Serviços Web utilizando-se apenas uma única chamada, escondendo-se a complexidade dos processos envolvido na chamada.

As demais classes tratam de chamadas sql utilizando XML como base para a transferência de dados, mas no entanto não são dependentes exclusivamente da implementação do cliente. No servidor, o serviço pgsqlConn.jws provê um método para a execução de queries sql apenas Select, Insert, Update e Delete (DML apenas). Ficou entendido durante a implementação do serviço que a utilização de DDL seria desnecessária, e até mesmo incompatível com os sistemas de dispositivos móveis, já que a forma de entrada de dados é muito restrita.

As chamadas devem ser realizadas no banco de dados PostgreSQL que deve estar localizado no mesmo servidor que o Axis. Em futuras versões pretende-se estender esse serviço à outros bancos de dados e possibilitar a conexão em outras máquinas, além da local. O código do serviço está anexo.

Sua utilização é simples. Basta chamar o método `executeQuery()`, passando a query desejada como parâmetro. O método retorna um objeto Result, que trata da recuperação dos dados da query. Através de métodos como `getType()` o tipo de dado da coluna, devidamente convertido para o padrão do Java (String, no lugar de Varchar, por exemplo) e de métodos como `get<Tipo de dado>()`, para a recuperação da informação e `getName()` para o nome da coluna.

A manipulação de linhas de resultado também é feita pela classe Result. São definidas para isso as funções `nextRow()`, `lastRow()`, `firstRow()` e `endRow()`.

Com essas classes é possível acessar qualquer serviço WEB, além de possibilitar a chamada a servidores HTTP padrão e utilizar Servles e servidores JSP.

4.4 Exemplos de Utilização

Esse ítem tem a proposta de mostrar a utilização das classes através de aplicativos simples e didáticos para realizar chamadas a servidores. Iniciar-se-á pela utilização da classe Link. Veja exemplo da figura 19.

```
/** urlTeste.java */
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import org.fw.link.*;

/** @author cruz*/
public class urlTeste extends MIDlet implements javax.microedition.lcdui.CommandListener {
    /** Cria uma nova instância de urlTeste */
    public urlTeste() {
    }

    /**  Esse método inicia a Interface de Usuário da aplicação     */
    private void initialize() {
        javax.microedition.lcdui.Display.getDisplay(this).setCurrent(get_textBoxURLTeste());
    }

    /** Chamdo pelo sistema quando um evento de comando é acionado
     * @param command Comando evocado
     * @param displayable um objeto Displayable que contem o comando
     */
    public void commandAction(Command command,Displayable displayable) {
        if (displayable == textBoxURLTeste) {
            if (command == backCommand) {
                javax.microedition.lcdui.Display.getDisplay(this).setCurrent(null);
                destroyApp(true);
                notifyDestroyed();
            }
        }
    }
```

Figura 19: Exemplo de utilização da classe Link

```
    /** Retorna a instância para o componente textBoxURLTeste
     * @return Instância para o componente textBoxURLTeste */
    private TextBox get_textBoxURLTeste() {
        if (textBoxURLTeste == null) {
            textBoxURLTeste = new TextBox("Url Teste", "", 4096, 0x0);
            textBoxURLTeste.addCommand(get_backCommand());
            textBoxURLTeste.setCommandListener(this);
        }
        return textBoxURLTeste;
    }

    /** Retorna instância do componente backCommand
     * @return Instance instância do componente backCommand */
    private Command get_backCommand() {
        if (backCommand == null) {
            backCommand = new Command("Voltar", "Voltar", Command.BACK, 1);
        }
        return backCommand;
    }

    /** implementação do método startApp() da classe abstrat MIDLet */
    public void startApp() {
        initialize();
        teste();
    }

    /** implementação do método pauseApp() da classe abstrat MIDLet */
    public void pauseApp() {
    }

    /** implementação do método destroyApp(boolean unconditional) da classe abstrat MIDLet */
    public void destroyApp(boolean unconditional) {
    }

    private void teste(){
        String esquema, hospedeiro, porto, caminho, servico, ur;
        Link lnk = new Link();
        String url[] = new String[7];
        Display.getDisplay(this);
        url[0] = "http://www.fundanet.br";
        url[1] = "http:/www.google.com";
        url[2] = "www.citeseer.com";
        url[3] = "www.wacho.cruzpesquisa.com.br/axis/";
        url[4] = "127.0.0.1:8080";
        url[5] = "127.0.0.1:8080/axis/Servico.jws?wsdl";
        url[6] = "127.0.0.1:8080/axis/Servico.jws?method=soma&v1=10&v2=20";
        for (int i = 0; i < url.length; i++){
            try{
               textBoxURLTeste.setString(textBoxURLTeste.getString() + "\nurl=" + url[i] +
                       "\n");
                lnk.setURL(url[i]);
               textBoxURLTeste.setString(textBoxURLTeste.getString() + "esquema=" +
                       lnk.getProtocolo() + "\n");
                textBoxURLTeste.setString(textBoxURLTeste.getString() + "hospedeiro = " +
                       lnk.getHospedeiro() + "\n");
                textBoxURLTeste.setString(textBoxURLTeste.getString() + "porto = " +
                       lnk.getPorto() + "\n");
                textBoxURLTeste.setString(textBoxURLTeste.getString() + "caminho = " +
                       lnk.getCaminho() + "\n");
                textBoxURLTeste.setString(textBoxURLTeste.getString() + "servico = " +
                       lnk.getParameter() + "\n");
                textBoxURLTeste.setString(textBoxURLTeste.getString() + "url final = " +
                       lnk.getURL() + "\n");
            } catch (LinkException le) {
                textBoxURLTeste.setString(textBoxURLTeste.getString() + "URL Incorreta = " +
                       le.getMessage() + "\n");
            }
        }
    }
    TextBox textBoxURLTeste;
    Command backCommand;
}
```

Figura 19: Exemplo de utilização da classe Link (Continuação)

Nesse exemplo é verificada a sintaxe das URLs passadas para a aplicação se a URL estiver sintaticamente correta, o exemplo analisa a URL e mostra suas partes constituintes. Caso contrário lança uma exceção tratada por `LinkException`, como mostrado quando analisada a url "http**:**/www.google.com" que deveria ser construída como “http**://**www.google.com” como pode ser visto na figura 20. As demais URLs são separadas em suas partes constituintes e mostradas ao usuário .

Figura 20: Resultado do aplicativo da figura 19

Essa classe ainda permite a reutilização de objetos através das chamadas set e get para cada um dos atributos e para a URL como um todo. Dessa forma,

quando se quer carregar uma nova página de servidor acessado previamente, basta mudar o caminho para a página, como será visto em HttpLink e SoapLink.

HttpLink implementa a classe Link expandindo sua utilidade para a chamada a servidores HTTP e JSP. Nela estão definidas a forma como o acesso ao servidor é realizada os métodos para a recuperação do conteúdo retornado pelo servidor.

Na figura 21 é mostrado um exemplo de como recuperar uma página da WEB através da utilização da classe HttpLink. Note que a chamada ao servidor é realizada em uma única linha `lnk.getContentPage("http://127.0.0.1")`. A página é carregada de forma transparente para o usuário através de uma String retornada pelo método.

```
/** Carrega página de servidor HTML */
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import org.fw.link.*;


public class MidletFetchPage extends MIDlet {
    TextBox t = new TextBox("Buscando http://127.0.0.1", "", 4096, 0x0);
    HttpLink lnk = new HttpLink();

    public void startApp() {
        this.Teste();
    }

    public void pauseApp() {
    }

    public void destroyApp(boolean unconditional) {
    }

    public void Teste(){
        try {
            t.setString(lnk.getContentPage("http://127.0.0.1"));
            Display.getDisplay(this).setCurrent(t);
        } catch (LinkException le){
            System.out.println(le.getMessage());
            le.printStackTrace();
        }
    }
}
```

Figura 21: Exemplo da utilização da classe HttpLink - Carregando uma página da WEB

Os exemplos mostrados nas figuras 22 e 23 executam chamadas a servidores utilizando os métodos GET e POST através da sobrecarga do método `getContentPage` permitindo a chamada a servidores HTTP ou JSP utilizando os respectivos métodos visando manter a transparência para o usuário. Para tanto, basta indicar qual o método a ser utilizado e passar os parâmetros necessários através de uma String. Note-se também que devem ser utilizadas as constantes definidas pela classe para execução das chamadas GET e POST. Além disso, o construtor da classe foi sobrecarregado possibilitado o carregamento da página Web durante a instanciação do objeto

```
/* MidletGet.java */
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import org.fw.link.*;

/** @author  cruz */
public class MidletGet extends MIDlet {
    public void startApp() {
        this.Teste();
    }
    public void pauseApp() { }
    public void destroyApp(boolean unconditional) { }
    public void Teste(){
        try {
            t.setString(
                    "URL:\nhttp://127.0.0.1:8084/tcc/ShowParamServlet\n\n" +
                    "Chamada:\nHttpLink.getContentPage(\n\"http://127.0.0.1:8084/" +
                    "tcc/ShowParamServlet\"," +
                    "\nHttpLink.GET," +
                    "\n \"name=Andre\")\n\n" +
                    "Resultado:\n" +
                    lnk.getContentPage("http://127.0.0.1:8084/tcc/ShowParamServlet",
                        HttpLink.GET, "name=Andre")
                    );
            Display.getDisplay(this).setCurrent(t);
        } catch (LinkException le){
            System.out.println(le.getMessage());
            le.printStackTrace();
        }
    }
    TextBox t = new TextBox("Get Teste", "", 4096, 0x0);
    HttpLink lnk = new HttpLink();
}
```

Figura 22: Exemplo da utilização da classe HttpLink – Utilização do método GET

Enquanto não forem realizadas novas chamada para outra URL a página carregada fica armazenada no objeto, podendo ser requisitada a qualquer momento. Essa característica foi definida pensando-se em facilitar a utilização de caches para páginas.

A classe HTTPLink ainda oferece a possibilidade da reutilização de objetos através do conjunto de métodos set e get disponibilizados para o usuário.

```
/* MidletPost.java */

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import org.fw.link.*;

/** @author  cruz */
public class MidletPost extends MIDlet {
    public MidletPost(){
    }
    public void startApp() {
        Teste();
    }
    public void pauseApp() {
    }
    public void destroyApp(boolean unconditional) {
    }
    public void Teste(){
        try {
            t.setString(
                    "URL:\nhttp://127.0.0.1:8084/tcc/ShowParamServlet\n\n" +
                    "Chamada:\nHttpLink.getContentPage(\n\"http://127.0.0.1:8084" +
                    "/tcc/ShowParamServlet\"," +
                    "\nHttpLink.POST," +
                    "\n \"name=Andre\")\n\n" +
                    "Resultado:\n" +
                    lnk.getContentPage("http://127.0.0.1:8084/tcc/ShowParamServlet",
                        HttpLink.POST, "name=Andre")
                    );

            Display.getDisplay(this).setCurrent(t);
        } catch (LinkException le){
            System.out.println(le.getMessage());
            le.printStackTrace();
        }
    }
    TextBox t = new TextBox("POST teste", "", 4096, 0x0);
    HttpLink lnk = new HttpLink();
}
```

Figura 23: Exemplo da utilização da classe HttpLink – Utilização do método POST

A chamada a servlets é feita através utilização dos métodos GET e POST em requisições para um servidor JSP. Ela pode ser vista em detalhes na figura 24. Sua utilização é idêntica a chamadas convencionais a servidores HTTP, como pode ser visto na figura 24.

Nesse exemplo é utilizado o serviço `HelloServlet` que está disponível nos anexos.

```
/* MidletInvockingServelet.java */

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import org.fw.link.*;

/** @author  cruz */
public class MidletInvockingServelet extends MIDlet {
    public void startApp() {
        this.teste();
    }

    public void pauseApp() { }
    public void destroyApp(boolean unconditional) { }
    private void teste(){
        try {
            t.setString(lnk.getContentPage("http://127.0.0.1:8084/tcc/HelloServlet", +
                HttpLink.GET, null));
            Display.getDisplay(this).setCurrent(t);
        } catch (LinkException le){
            System.out.println(le.getMessage());
            le.printStackTrace();
        }
    }


    HttpLink lnk = new HttpLink();
    TextBox t = new TextBox("InvockingServelet", "", 4096, 0x0);
}
```

Figura 24: Exemplo da utilização da classe HttpLink – Acessando um ServLet

A figura 25 mostra os resultados obtidos pelos aplicativos mostrados que utilizam a classe HttpLink. Vale também ressaltar que a partir de um servidor HTTP pode-se obter qualquer arquivo de qualquer tipo e depende do usuário trata-lo de acordo com as especificações desse tipo de arquivo. Esse Framework apenas

facilita a forma de acesso ao recurso, não o tratamento dele. Isso pode vir a ser incorporado em futuras versões.

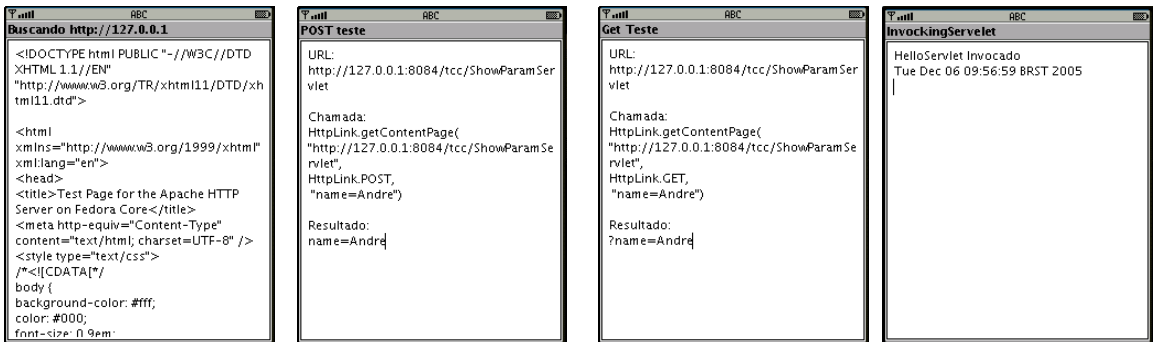

Figura 25: Resultado dos aplicativos das figura 21, 22, 23 e 24 respectivamente

A chamada remota a procedimento pode ser realizada à partir de duas classes, respectivamente SoapLink e Serializador. A classe SoapLink é uma facilitador da utilização da biblioteca ksoap. Com ela é possível realizar chamadas obtendo-se objetos SoapObject ou XmlIO como resultado.

A classes Serializador por sua vez é mais transparente e permite a chamada a métodos remotos obtendo-se o resultado das chamadas a métodos remotos imediatamente. Para tanto, basta que seja passado a URL do serviço junto com um *array* de objetos que deve ser construído da seguinte forma: nome do serviço como String na posição 0 do *array*, nome do primeiro, nas posições ímpares devem ser colocados os nomes dos objetos a serem passados como parâmetros e nas posições pares o valor desses objetos, com a ressalva de que o nome dos objetos

devem ser Strings. No exemplo da figura 26 pode ser visto o objeto `method` do tipo `Object []`. Nele estão o nome do método a ser chamado, o nome dos parâmetros e o valor passado para cada parâmetro.

```
/** MidletSoma.java */

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import org.fw.link.*;
import org.ksoap.*;

/** @author  cruz */
public class MidletSoma extends MIDlet {
    public MidletSoma(){
    }
    public void startApp() {
        Teste();
    }
    public void pauseApp() {
    }
    public void destroyApp(boolean unconditional) {
    }
    public void Teste(){
        try {
            StringBuffer resposta = new StringBuffer();
            String url = "http://127.0.0.1:8080/axis/Servico.jws";
            Object [] method = {"soma", "v1", new Integer(10), "v2", new Integer(20) };

            Serializador s = new Serializador();
            Object o = s.call(url, method);
            resposta.append("Url do Serviço:\n" + url + "\nServiço Chamdado:\n" +
                ((String)method[0]) + "\nParâmetros:\n");
            for (int i = 1; i < method.length; i++)
                resposta.append(((i%2 == 1)? "\t": " = ") + method[i] +
                    ((i%2 == 0)? "\n": ""));
            resposta.append("Resultado: " + ((Integer)o).toString());
            t = new TextBox("http://127.0.0.1:8080/axis/Servico.jws", "", 4096, 0);
            t.setString(resposta.toString());
            Display.getDisplay(this).setCurrent(t);
        } catch (Exception e){
            t.setString(e.getMessage());
            Display.getDisplay(this).setCurrent(t);
        }
    }
    TextBox t;
}
```

Figura 26: Exemplo de utilização da classe serializador

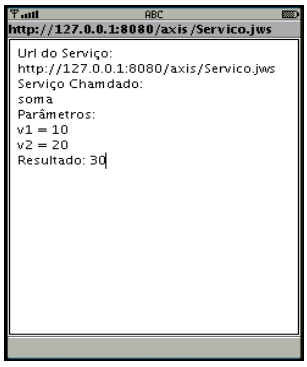

Figura 27: Resultado do exemplo da figura 26

As classes SqlLink, Query, e Result tratam chamadas SQL para o serviço pgsqlConn, mostrado no anexo. O serviço foi desenvolvido para o banco de dados SQL, mas pode-se estender facilmente sua portabilidade para outros bancos, bastando para isso acrescentar os *drivers* necessários e modificar a forma como é construída a chamada para a URL do SGBD dentro do serviço. Outra característica que pode ser adicionada em futuras versões do Framework é o acesso a bancos de dados em máquinas que não a onde está instalado o serviço.

A classe SqlLink realiza chamadas ao serviço pgsqlConn, passando a query a ser executada como parâmetro e retornando um objeto do tipo Result. Para instanciar-se o objeto deve ser passado para o construtor o nome do banco a ser acessado, o nome do usuário e a senha. Não foi implementada segurança, de forma que a senha trafega através da rede sem criptografia. Isso deve ser alterado em futuras versões do Framework.

O objeto Result retornado de SqlLink possue os metadados da tabela consultado e as linhas de resultado. Eles podem ser recuperados através dos métodos `getColumnName()` e `getColumnType()` e get<tipo de dado>() respectivamente.

A classe Query é privada. Ela é responsável pela analise e tradução dos dados enviados pelo serviço no formato XML para o objeto do Result.. Na figura 28 está um exemplo da utilização das classes de tratamento de SQL.

```
/* MidletSQLLink.java */

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import org.fw.link.*;
/** @author  cruz */
public class MidletSQLLink extends MIDlet {
    public MidletSQLLink(){
    }
    public void startApp() {
        Teste();
    }
    public void pauseApp() {
    }
    public void destroyApp(boolean unconditional) {
    }
    public void Teste(){
        TextBox t = new TextBox("Sql Teste", "", 4096, 0x0);
        StringBuffer resultado = new StringBuffer();
        String query = "select * from teste";
        try {
            SqlLink slnk = new SqlLink("teste", "cruz", "awszsedx");
            Result rs = slnk.executeQuery(query);
            int ncols = rs.getNCols();
            resultado.append("Metadados\n");
            for (int i = 0; i < ncols; i++){
                resultado.append(rs.getColumnName(i) + " ");
                resultado.append(rs.getColumnType(i) + "\n");
            }
            resultado.append("\n\nDados\n");
            rs.first();
            while (rs.next()){
                for (int i = 0; i < ncols; i++)
                    resultado.append(rs.getString(i) + ((i%2 == 0)? " | " : "\n"));
            }
            t.setString(resultado.toString());
            Display.getDisplay(this).setCurrent(t);
        } catch (Exception e){
            t.setString("Exceção Lançada: " + e.getMessage());
            Display.getDisplay(this).setCurrent(t);
            System.out.println(e.getMessage());
            e.printStackTrace();
        }
    }
}
```

Figura 28: Exemplo de chamadas ao serviço pgsqlConn.jws

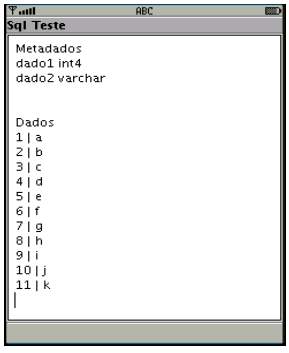

Figura 29: Resultado do exemplo da figura 28

CONCLUSÕES

Os dispositivos *Wireless* são equipamentos eletrônicos com capacidade computacional, portáteis que podem comunicar-se com uma rede para trocar informações, entretanto, possuem grandes limitações quanto a espaço para armazenamento e capacidade de processamento. A melhor forma de adicionar capacidades computacionais a dispositivos *wireless* é através da utilização de sistemas distribuídos, que permitem a utilização de recursos disponíveis em outras máquinas, mais especificamente, o que foi citado nesse projeto, a arquitetura cliente servidor, já que devido às suas pode omitir recursos de interesse para o compartilhamento, ficando a cargo do servidor ou da rede.

Serviços Web são aplicações distribuídas que se comunicam com base no protocolo HTTP/SOAP. A utilização de Serviços Web, devido a sua infraestrutura é ideal para a o tratamento de aplicações distribuídas tendo como cliente o dispositivo *Wireless*. O protocolo SOAP com sua capacidade de realizar chamadas à métodos pode substituir o RMI em dispositivos que utilizam J2ME, e o XML SAX API é útil pois não ocupa espaço excessivo de memória durante o tratamento de arquivos XML, como o DOM, por exemplo, em outras implementações.

O objetivo do framework é facilitar a utilização de Serviços Web por usuários de dispositivos móveis. Durante seu projeto, foram adicionados meios para o

reaproveitamento de código e objetos visando facilitar a implementação e desempenho dos aplicativos baseados nele. A classes fornecidas disponibilizadas por ele tratam do acesso a recursos na WEB através do protocolo HTTP utilizando-se SOAP. Com ele é possível também carregar o conteúdo de servidores HTTP, JSP.

Em futuras versões do framework devem ser acrescentados suporte para criptografia e segurança, carregamento de arquivos binários disponibilizados por servidores e conexão com outros SGBDs além do PostgreSQL.

REFERÊNCIAS

ARMSTRONG, Eric, et al. **The J2EE™ 1.4 Tutorial**. Disponível em <http://JAVA.sun.com/j2ee/1.4/docs/tutorial/doc/J2EETutorial.pdf>. Acesso em 23 de junho de 2005.

CHAPPELL, David, JEWELL, Tyler. **JAVA Web Services**. Sebastopol: O'Reilly, 2002.

ECKHOUSE JR., R. H. et al – **Issues in Distribuited Processing – An Overview of Two Workshops**. Computer Magazine, n 11 vol. 1, p. 22-26, jan de 1978.

HORSTMAN, Cay S., CORNEL, Gary. **Core JAVA™ 2: Volume I – Fundamentals**. 5ed. Palo Alto: Prentice Hall, 2000.

HUANG, Nan-Chao. **A Cross Platform Web Service Implementation Using SOAP**. 2003. 109 f. Grau: Dissertação (Mestrado em Ciência da Computação) - Knowledge Systems Institute, Skokie, 2003.

JACOBSON, I., Griss, M., JHONSON, P. **Software Reuse: Architecture Process and Organization for Business Success**. New Jersey: ACM Press, 1997.

LINDEN, Peter. **Just Java**. São Paulo: Makron Books, 1997.

MILLER JR., Ralph. **Teoria e Problemas de XML**. Porto Alegre: Bookman, 2003.

NIEMEYER, Pat, KNUDSEN, Jonathan. **Learning JAVA**. Sebastopol: O'Reilly, 2000.

PEEBLES, R & MANNING, E. **System Architecture for Distributed Data Management**. Computer Magazine, n° 11, vol. 1, p. 40-47, jan de 1978.

PETERSON, J. **Notes on a Workshop on Distributing Computing**. ACM Operating System Review, n. 13, vol. 3, p. 18-30, jul de 1979.

W3 Consortium. **Extensible Markup Language (XML) 1.0 (Third Edition)**. Disponível em: <http://www.w3.org/TR/2004/REC-xml-20040204/>. Acesso em 12

de agosto de 2005.

W3 Consortium. **Web Services Architecture**. Disponível em <http://www.w3.org/TR/ws-arch/>. Acesso em 16 outubro de 2005.

ANEXO - Código dos serviços utilizados

HelloServlet.java

```
/*
 * HelloServlet.java
 */

import java.io.*;
import java.net.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

/**
 * @author cruz
 */
public class HelloServlet extends HttpServlet {

    /** Processes requests for both HTTP <code>GET</code> and
<code>POST</code> methods.
     * @param request servlet request
     * @param response servlet response
     */
    protected void processRequest(HttpServletRequest request,
HttpServletResponse response)
    throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        out.println("HelloServlet Invocado");
        out.println(new Date());
        out.close();
    }

    // <editor-fold defaultstate="collapsed" desc="HttpServlet methods.
Click on the + sign on the left to edit the code.">
    /** Handles the HTTP <code>GET</code> method.
     * @param request servlet request
     * @param response servlet response
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse
response)
    throws ServletException, IOException {
        processRequest(request, response);
    }

    /** Handles the HTTP <code>POST</code> method.
     * @param request servlet request
     * @param response servlet response
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse
response)
    throws ServletException, IOException {
        processRequest(request, response);
    }

    /** Returns a short description of the servlet.
     */
    public String getServletInfo() {
```

```
        return "Short description";
    }
    // </editor-fold>
}
```

showParamServlet.java

```
/*
 * ShowParamServlet.java
 */

import java.io.*;
import java.net.*;

import javax.servlet.*;
import javax.servlet.http.*;

/**
 * @author cruz
 */
public class ShowParamServlet extends HttpServlet {

    /** Processes requests for both HTTP <code>GET</code> and
<code>POST</code> methods.
     * @param request servlet request
     * @param response servlet response
     */
    protected void processRequest(HttpServletRequest request,
HttpServletResponse response)
    throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        BufferedReader br = request.getReader();
        String buff = br.readLine();
        out.print(buff);
        out.close();
    }

    // <editor-fold defaultstate="collapsed" desc="HttpServlet methods.
Click on the + sign on the left to edit the code.">
    /** Handles the HTTP <code>GET</code> method.
     * @param request servlet request
     * @param response servlet response
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse
response)
    throws ServletException, IOException {
        processRequest(request, response);
    }

    /** Handles the HTTP <code>POST</code> method.
     * @param request servlet request
     * @param response servlet response
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse
response)
    throws ServletException, IOException {
        processRequest(request, response);
    }
```

```
    /** Returns a short description of the servlet.
     */
    public String getServletInfo() {
        return "Short description";
    }
    // </editor-fold>
}
```

Soma.jws

```
/** Serviço soma */
public class Soma {
   public int soma(int v1, int v2) {
      return v1 + v2;
   }
}
```

pgsqlConn.jws

```
import java.io.*;
import java.net.*;
import java.util.*;
import java.lang.*;
import java.sql.*;
import org.postgresql.*;


/** Serviço  pgsqlConn */
public class pgsqlConn{

    public String execQuery
           (String db, String query, String user, String password){
        this.conn = null;
        this.rs = null;
        this.st = null;
        query = query.trim();
        if (query.indexOf('   ') != -1)
            return "<ERROR>\\t não é aceito pelo sistema.</ERRROR>";
        String str = query.
                     substring(0, query.indexOf(' ')).toLowerCase().trim();
        if (str.equals("insert") || str.equals("update") ||
                        str.equals("delete")){
            return "<?xml version=\"1.0\" ?>" +
                   this.execquery(db, query, user, password);
        } else if (str.equals("select")){
            return "<?xml version=\"1.0\" ?>" +
                   this.execSelect(db, query, user, password);
        } else
            return "<?xml version=\"1.0\" ?>\n<ERROR>” +
                   “Query invalida: é apenas aceito Select, “ +
                   “Insert, Update e Delete.</ERRRO>";
    }

    private String execquery(String db, String query, String user,
```

```
                        String password){
    java.lang.String str = "";

    try {
        Class.forName("org.postgresql.Driver");
    } catch (Exception e){
        str = "<ERROR>Não foi possivel carregar o driver.</ERROR>";
        return str;
    }

    try {
        this.connect(db, user, password);
        this.st = conn.createStatement();
        this.rs = this.st.executeQuery(query);

    } catch (SQLException sqle){
        str = (sqle.getErrorCode() == 0)? str =
               "</OK>": "<ERROR>" + sqle.getErrorCode() + ":" +
               sqle.getMessage() + "\n" + "</ERROR>";
    } finally {
        try {
            if (this.rs != null)this.rs.close();
            if (this.st != null) this.st.close();
            if (this.conn != null) this.conn.close();
        } catch (Exception e){
            System.out.println(e.getMessage());
            e.printStackTrace();
        }
    }
    return str;
}

private String execSelect(String db, String query, String user,
                          String password){
    java.lang.String str;

    try {
        Class.forName("org.postgresql.Driver");
    } catch (Exception e){
        str = "<ERROR>Não foi possivel carregar o driver.</ERROR>";
        return str;
    }

    try {
        this.connect(db, user, password);
        this.st = conn.createStatement();
        this.rs = this.st.executeQuery(query);
        str = toXML(rs);
    } catch (SQLException sqle){
        str = "<ERROR>" + sqle.getErrorCode() + ":" +
              sqle.getMessage() + "\n" + "</ERROR>";
    } finally {
        try {
            if (this.rs != null)this.rs.close();
            if (this.st != null) this.st.close();
            if (this.conn != null) this.conn.close();
        } catch (Exception e){
        }
    }
    return str;
}
```

```
private void connect(java.lang.String db, java.lang.String user,
                     java.lang.String senha)
                     throws java.sql.SQLException {
    this.db = db;
    this.user = user;
    this.passwd = senha;
    conn = DriverManager.getConnection(this.url + db, this.user,
                                       this.passwd);
}

private java.lang.String toXML(ResultSet rs)
                                      throws java.sql.SQLException{
    StringBuffer sb = new StringBuffer();
    ResultSetMetaData rsmd;

    rsmd = rs.getMetaData();
    int ncols = rsmd.getColumnCount();

    sb.append("<ResultSet>\n");
    sb.append("\t<Metadata>\n");
    for (int i = 1; i <= ncols; i++){
        sb.append("\t\t<Column>\n");
        sb.append("\t\t\t<Name>");
        sb.append(rsmd.getColumnLabel(i));
        sb.append("</Name>\n");
        sb.append("\t\t\t<Type>");
        sb.append(rsmd.getColumnTypeName(i));
        sb.append("</Type>\n");
        sb.append("\t\t</Column>\n");
    }
    sb.append("\t</Metadata>\n");
    sb.append("\t<rows>\n");
    while (rs.next()){
        sb.append("\t\t<row>");
        for (int i = 1; i <= ncols; i++){
            sb.append("<col>");
            sb.append(this.rs.getString(i));
            sb.append("</col>");
        }
        sb.append("</row>\n");
    }
    sb.append("\n\t</rows>\n");
    sb.append("</ResultSet>\n");

    return sb.toString();
}

// url do banco de dados
private String url = "jdbc:postgresql://localhost:5432/";

// nome do banco de dados
private String db;

// nome do usuário
private String user;

// senha do usuário
private String passwd;

// Connection
```

```
    private java.sql.Connection conn;
    // statement
    private java.sql.Statement st;
    // resultset
    private java.sql.ResultSet rs;
}
```

CRUZ, André Luis Guimarães
Um framework Java para dispositivos wireless.
Marília, SP: [s.n.], 2005.
73 f.

Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Centro Universitário Eurípides de Marília - Fundação de Ensino Eurípides Soares da Rocha.
1. Framework 2. Serviços web 3. Dispositivos wireless 4. Java.

CDD: 005.1151