UNIVERSIDADE FEDERAL DE SANTA CATARINA - UFSC

DESENVOLVIMENTO DE APLICAÇÕES DE DISPOSITIVOS MÓVEIS COM J2ME E INTEGRAÇÃO COM WEB SERVICES

Trabalho de Conclusão de Curso
apresentado ao Curso de Bacharelado
em Sistemas de Informação da
Universidade Federal de Santa Catarina
como requisito parcial para obtenção do
grau de bacharel em Sistemas de Informação

Orientador: Prof. Frank Augusto Siqueira, Dr.

Florianópolis
2005

LUCAS DE SOUZA REIS GOMES

DESENVOLVIMENTO DE APLICAÇÕES DE DISPOSITIVOS MÓVEIS COM J2ME E INTEGRAÇÃO COM WEB SERVICES

Florianópolis, 05 de Julho de 2005.

BANCA EXAMINADORA

_____

Prof. Frank Augusto Siqueira
Universidade Federal de Santa Catarina
Orientador

_____

Prof. João Bosco Sobral
Universidade Federal de Santa Catarina

_____

Prof. Mário Dantas
Universidade Federal de Santa Catarina

AGRADECIMENTOS

Agradeço primeiramente aos meus pais, avós e irmãos, que são as pessoas mais importantes na minha vida e que sempre me apoiaram durante toda esta etapa.

Agradeço aos meus familiares, amigos, todos que me ajudaram durante este trabalho.

SUMÁRIO

Lista de Figuras

Glossário

XML: Extensible Markup Language

SOAP: Simple Object Access Protocol

UDDI: Universal Description Discovery and Integration

WSDL: Web Services Description Language

J2ME: Java 2 Platform Micro Edition

J2SE: Java 2 Platform Standard Edition

ERP: Enterprise Resource Planning

CRM: Customer Relationship Management

RPC: Remote Procedure Call

ORPC: Object Remote Procedure Call

DCOM: Distributed Component Object Model

IDL: Interface Definition Language

HTTP: Hypertext Transport Protocol

SMTP: Simple Mail Transfer Protocol

FTP: File Transfer Protocol

W3C: World Wide Web Consortium

Resumo

O crescimento do mercado de telefonia móvel e a demanda por aplicações que possam ser executadas em dispositivos portáteis, com a possibilidade de comunicação com outras aplicações, utilizando uma rede wireless, motivou o estudo da arquitetura J2ME. Esta pode ser executada em qualquer plataforma que possua uma máquina virtual instalada, sem a necessidade de se prender a um fabricante ou a uma tecnologia. Esta plataforma possui conceitos não encontrados em outras plataformas Java como a configuração, o perfil e os pacotes opcionais.

Os Web Services permitem a interação entre objetos pela internet ou por uma rede corporativa, utilizando padrões neutros de plataforma, permitindo ocultar os detalhes relativos à implementação provenientes do cliente. O cliente não precisa saber em que linguagem o serviço foi construído e nem mesmo em que sistema operacional o serviço está sendo executado.

Portanto, quando pensamos em dispositivos móveis, cuja capacidade de processamento, armazenamento e transmissão de dados é limitada, a necessidade de integração é indispensável para desenvolver sistemas de grande porte, e os web services propiciam esta integração. E neste trabalho serão explicados os conceitos relacionados a Web Services e a plataforma de J2ME. E também será desenvolvida uma aplicação que comprove a integração de J2ME com Web Services.

Palavras-chave: Web services, Java, J2ME, MIDP, CLDC, JSR-172, segurança, WSDL, SOAP, UDDI.

1. INTRODUÇÃO

1.1. Contextualização

O desenvolvimento de aplicações distribuídas sempre foi um campo muito importante desde que os sistemas migraram de mainframes para computadores pessoais, estações de trabalho ou dispositivos móveis que podem ser acessados através de uma rede. A integração entre aplicações sempre foi um grande problema para desenvolvedores de sistemas. Em sistemas legados, isto ocorria devido às limitações tecnológicas das plataformas em que foram desenvolvidos. Já em sistemas mais modernos, como os ERP (Enterprise Resource Management), CRM (Customer Relationship Management), ou Supply Chain, esta dificuldade também está presente, principalmente quando são desenvolvidos por fabricantes diferentes.

Devido à necessidade de integração dos sistemas, surgiu o conceito de operação conjunta, onde sistemas heterogêneos devem ser capazes de estabelecer comunicação e compartilhar dados, sem estarem ligados fisicamente entre si, através de uma comunicação transparente. Através deste conceito, surgiu a idéia de web services, onde serviços são disponibilizados e podem ser acessados remotamente sem intervenção humana, e utilizam-se de protocolos padrões para definir sua arquitetura.

O mercado de Web Services está explodindo em novas oportunidades, e a cada dia os desenvolvedores criam dispositivos mais poderosos e novas aplicações. Os telefones celulares e os handhelds (PDAs) estão se tornando populares entre os

consumidores de todo o mundo, onde é possível cada vez mais utilizá-los em atividades cotidianas.

Antes restritos a executar aplicações simples como calculadoras e calendários, hoje estes dispositivos estão ganhando aplicações mais complexas como streaming de vídeo, tornando possível para viajantes ou pessoas de negócios que estão sempre ocupadas, assistir novos clipes em tempo real ou apresentações de negócios a partir de um táxi ou em um hotel.

Por outro lado, o crescimento do mercado de telefonia móvel e a demanda por aplicações que possam ser executadas em dispositivos portáteis, com a possibilidade de comunicação com outras aplicações, utilizando uma rede wireless, motivou o estudo da arquitetura Java J2ME. Esta possui as mesmas características da plataforma Java convencional (J2SE – Java 2 Standard Edition), e pode ser executada em qualquer plataforma que possua uma máquina virtual instalada, sem a necessidade de se prender a um fabricante ou a uma tecnologia, além de ser uma linguagem de padrão aberto e que oferece um conjunto de tecnologias e APIs.

A arquitetura Java J2ME possui características diferentes da arquitetura tradicional do JAVA, a J2SE, pois ela é otimizada para o dispositivo para a qual ela foi desenvolvida.

1.2. Objetivos do Trabalho

1.2.1. Objetivo Geral

O objetivo geral deste trabalho é apresentar conclusões a respeito do estudo da arquitetura de Web Services na plataforma de dispositivos móveis ou embutidos J2ME. Pretende-se também estudar tecnologias capazes de integrar estas duas arquiteturas e criar uma aplicação como estudo de caso.

1.2.2. Objetivos Específicos

- Estudar a arquitetura da plataforma J2ME;
- Estudar a arquitetura de Web Services;
- Estudar a API J2ME Web Services (WSA);
- Estudar mecanismos de segurança aplicados à plataforma J2ME e Web Services;
- Desenvolver uma aplicação que utilize a tecnologia J2ME e Web Services;

1.3. Justificativa do Trabalho

O crescimento do mercado de telefonia móvel e a demanda por aplicações que possam ser executadas em dispositivos portáteis, com a possibilidade de comunicação com outras aplicações, utilizando uma rede wireless, motivou o estudo da arquitetura J2ME. Esta possui as mesmas características da plataforma Java convencional (J2SE – Java 2 Standard Edition), podendo ser executada em qualquer plataforma que possua uma máquina virtual instalada, sem a necessidade de se prender a um fabricante ou a uma tecnologia, além de ser uma linguagem de padrão aberto e que oferece um conjunto de tecnologias e APIs.

Os Web Services são o que há de mais recente em desenvolvimento de aplicações, e vem atraindo o interesse de desenvolvedores que trabalham em todas as plataformas. Eles permitem a interação entre objetos pela internet ou por uma rede corporativa. Porém, não são a primeira tecnologia a permitir isto, mas diferem de outras tecnologias existentes pois utilizam padrões neutros de plataforma, como o HTTP e XML, permitindo ocultar os detalhes relativos à implementação provenientes do cliente. O cliente precisa conhecer o URL do serviço e os tipos de dados usados para as chamadas do método, mas não precisa saber em que linguagem o serviço foi construído e nem mesmo em que sistema operacional o serviço está sendo executado.

Portanto, quando pensamos em dispositivos móveis, cuja capacidade de processamento, armazenamento e transmissão de dados é limitada, a necessidade de integração é indispensável para desenvolver sistemas de grande porte, e os web services propiciam esta integração. Por exemplo, se um dispositivo móvel tem acesso a

diversas aplicações distribuídas pela rede (ou Internet), ele poderia solicitar uma série de serviços, que seriam executados no servidor, e receberia somente o resultado deste processamento como resposta, economizando recursos.

E como será explicado durante este trabalho, a tecnologia J2ME Web Services permite realizar estas operações, através de padrões e diversas tecnologias gratuitas.

1.4. Escopo

O escopo deste trabalho compreende o estudo da arquitetura J2ME, que pode ser definida como Genérica, pois é composta por um conjunto de bibliotecas, que se divide nos seguintes níveis: perfil, configurações, máquina virtual e sistema operacional. Neste projeto será priorizado o estudo do perfil MIDP 2.0 e da configuração CLDC, que é utilizado em celulares com poucos recursos de memória e processamento, como será explicado durante o trabalho. No âmbito de estudo de Web Services serão abordadas as tecnologias envolvidas nesta arquitetura, como XML, SOAP, WSDL, UDDI, para que seja possível utilizar a API opcional de J2ME para Web Services (WSA).

Também serão abordados alguns requisitos essenciais para um sistema de Serviços Web, como transações, Qualidade de Serviço, gerenciamento de serviços web e segurança.

Será desenvolvida uma aplicação em J2ME que utilizará os serviços de um web service. O escopo desta aplicação será focado no desenvolvimento do cliente. Portanto, o servidor simulará algumas operações representando um acesso a um banco de dados ou uma chamada a outras aplicações ou serviços.

2. A ARQUITETURA DE JAVA 2 MICRO EDITION (J2ME)

2.1. Visão Geral da Plataforma Java J2ME

Reconhecendo que uma tecnologia não é a melhor opção para todas as áreas da indústria, a Sun Microsystems[COURTNEY, 2002] agrupou a plataforma Java 2 em três edições, cada uma com um comportamento específico para seu mercado:

- Java 2 Enterprise Edition (J2EE) – utilizada em servidores corporativos.

- Java 2 Standard Edition (J2SE) – para uso residencial e mercado desktop.

- Java 2 Micro Edition (J2ME) – para dispositivos pequenos e com poucos recursos de memória e processamento.

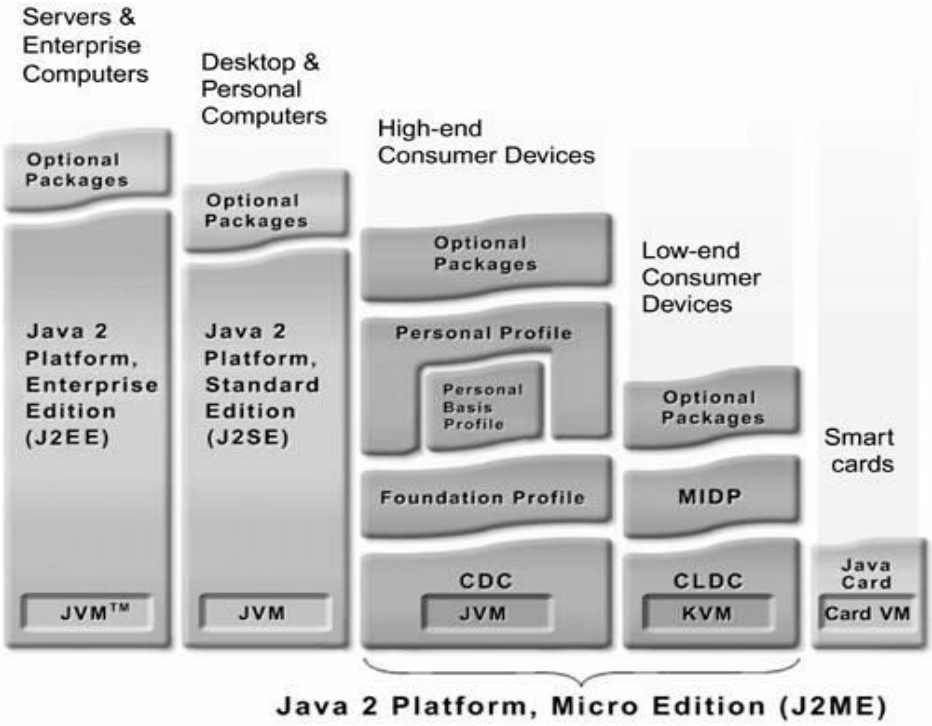

Figura 1 - Edições da plataforma Java 2 e seu mercado. Fonte: [RIGS, 2003]

A plataforma Java 2 Micro Edition está focada para um mercado muito amplo que está crescendo rapidamente, e abrange um grande número de dispositivos, desde pagers, aparelhos celulares, codificadores de televisão, até computadores pessoais. E assim como as outras plataformas Java, J2ME manteve as características de portabilidade de código, segurança, comunicação com rede, e grande escalabilidade.

J2ME pode ser dividida em duas categorias que agrupam dispositivos com características semelhantes. Dispositivos de alto-nível são representados pela configuração CDC (Connected Device Configuration). Exemplos destes dispositivos são os computadores de bordo de automóveis, conversores de TV por assinatura (set-top-boxes) e demais dispositivos que possuem uma rica capacidade de interfaces com usuário, tamanho de memória de pelo menos 2 a 4 megabytes, persistência e conexões de rede de alta velocidade. Já os dispositivos de baixo-nível são representados pela configuração CLDC (Connected Limited Device Configuration), que é utilizada em telefones celulares, pagers e outros dispositivos que possuem uma interface simples para o usuário (se comparada a um computador pessoal), pouca capacidade de memória (de 128 a 256 kilobytes), redes com baixa taxa de transferência e conexão intermitente. Muitos desses dispositivos são alimentados por baterias.

O mercado consumidor necessita medir a flexibilidade com a qual uma tecnologia é empregada ao desenvolver uma aplicação. Isto se mostra necessário devido ao grande número de tipos de dispositivos, configurações de hardware, ao modo como as aplicações são operadas pelo dispositivo (através do teclado, comando de voz, etc), ao grande número de aplicações e funcionalidades, e à necessidade de adaptação a mudanças e crescimento em escala.

A arquitetura J2ME foi projetada para ser modular e escalável para suportar estas necessidades. Contudo, para permitir isso, o ambiente J2ME oferece diversas tecnologias de máquina virtual, cada uma otimizada para diferentes tipos de processador, e memórias encontradas no mercado consumidor. E para suportar essa personalização e extensibilidade foram definidos pela arquitetura J2ME três conceitos: Configuração, Perfil e API Opcional.

A Configuração define uma plataforma mínima ou um grupo de dispositivos, que são semelhantes no tamanho da memória e capacidade de processamento. Uma configuração define a linguagem Java e a funcionalidade mínima da máquina virtual, e uma biblioteca de classes mínima que um fabricante ou provedor de conteúdo pode esperar de dispositivos da mesma categoria.

O Perfil é uma camada acima da configuração que especifica a demanda de um segmento de mercado ou família de dispositivo, ou seja, ele garante a interoperabilidade entre certas famílias de dispositivo ou domínio definido pela plataforma Java para aquele mercado. O Perfil inclui tipicamente bibliotecas de classes que são mais específicas para um domínio. Um dispositivo pode conter diversos Perfis.

Os pacotes opcionais são algumas APIs aplicáveis a um grande número de aparelhos e famílias de aparelhos. Os pacotes opcionais estão uma camada acima do Perfil. Esses são independentes do segmento de mercado e famílias de dispositivos, e tem como característica fundamental permitir adicionar APIs e garantir mais flexibilidade aos Perfis. Um dispositivo pode suportar múltiplos pacotes opcionais.

Nos próximos capítulos, será detalha cada um destes conceitos definidos pela arquitetura J2ME.

2.2. Configuração

A configuração define o mínimo que um desenvolvedor pode esperar de um dispositivo, agrupando-os de acordo com a semelhança entre seus requisitos de memória e de hardware[MUCHOW, 2004]. Ou seja, uma configuração especifica:

- quais as funcionalidades suportadas pela linguagem Java;

- quais as funcionalidades suportadas pela máquina virtual;

- quais as bibliotecas Java e APIs são suportadas.

O ambiente J2ME pode ser instalado sobre mais de uma configuração. Cada configuração especifica a linguagem, a máquina virtual e um conjunto de APIs que será implementado sobre uma família de dispositivos. Esta camada é menos visível ao usuário, porém, é muito importante para os desenvolvedores de perfil.

Foram definidos dois padrões de configuração J2ME, o CLDC (Connected, Limited Device Configuration) e o CDC (Connected Device Configuration)[COURTNEY, 2002].

2.2.1. – CLDC (Connected, Limited Device Configuration)

A configuração CLDC, segundo [RIGS, 2003], é focada para os dispositivos considerados de uso pessoal, móvel, operado por bateria como telefones celulares, pagers, organizadores pessoais e outros aparelhos de tamanho semelhante. Esta configuração inclui algumas novas bibliotecas que não fazem parte da arquitetura J2SE e que foram desenvolvidas especificamente para suprir algumas necessidades dos dispositivos com poucos recursos.

A especificação CLDC[COURTNEY, 2002] assume que os dispositivos devem possuir de 160 KB a 512 KB de memória não-volátil para a máquina virtual e bibliotecas CLDC, e pelo menos 32 KB de memória volátil para a execução da máquina virtual. O processador pode ser de 16 ou 32 bits.

A conexão de rede é limitada devido ao fato de ser lenta e intermitente. Muitos telefones possuem uma taxa de transferência de apenas 9.6 Kbps, além de possuírem uma tela pequena, memória limitada, obrigando as aplicações desenvolvidas para esta configuração a serem bastante otimizadas em relação ao uso de memória e rede.

Essa configuração é baseada numa máquina virtual de tamanho reduzido chamada KVM. Este nome provém do fato da JVM (Java Virtual Machine) deste dispositivo possuir um tamanho que é medido em kilobytes ao invés de megabytes. Enquanto CLDC é um documento de especificação, a KVM é uma parte do software, e devido ao seu tamanho, a KVM não pode realizar as mesmas funcionalidades que a tradicional JVM especificada através de J2SE. Por exemplo, não podem ser adicionados métodos nativos à KVM em tempo de execução (*runtime*). Todas as funcionalidades devem ser construídas dentro da KVM. Esta somente inclui uma parte do verificador padrão do *bytecode* Java. Isto significa que a tarefa de verificar classes é dividida entre o dispositivo CLDC e algum mecanismo externo.

Abaixo, segue tabela contendo as áreas que são abrangidas e não são abrangidas por esta configuração:

Áreas abrangidas por CLDC	**Áreas não abrangidas por CLDC**
Linguagem Java e funcionalidades da máquina virtual	Gerenciamento de ciclo de vida da aplicação (instalação, execução, remoção)
Bibliotecas Padrão Java (Java.lang.*, Java.io.*, Java.util.*)	Funcionalidade da interface do usuário
Entrada e saída	Manipulação de eventos

Segurança	Modelo de aplicação de alto nível (interação entre o usuário e a aplicação)
Internacionalização	

As funcionalidades que não são abrangidas pelo CLDC podem ser implementadas através do perfil para manter a interoperabilidade entre os dispositivos que utilizam essa configuração, e não exceder a limitação de memória definida na especificação ou excluir alguma categoria de dispositivo.

2.2.2. – CDC (Connected Device Configuration)

A configuração CDC é focada para dispositivos como aparelhos conversores de TV a cabo, sistemas navegadores de carro e PDAs, que devem possuir capacidade para suportar todas as funcionalidades da máquina virtual Java (JVM)[COURTNEY, 2002]. Quando se falou em CLDC e pequenos dispositivos, era indispensável pensar em termos de memória volátil e não volátil. Porém, ao tratar-se de CDC, este requisito não é tão importante, pois esta configuração especifica a necessidade que um dispositivo tenha no mínimo 512KB de memória ROM, 256KB de memória RAM, e algum tipo de conexão de rede.

Esta configuração especifica o uso de todas as funcionalidades e bibliotecas da máquina virtual do J2SE; contudo, neste contexto ela é chamada de CVM (Compact Virtual Machine). A máquina virtual CVM foi projetada para oferecer as seguintes funcionalidades:

- portabilidade;
- execução de classes Java fora da memória ROM;

- suporte a interfaces e rotinas de processamento do sistema operacional em tempo real;

- mapear threads Java diretamente para threads nativa do sistema operacional;

- suporte a todas funcionalidades da versão Java 2 Standard Edition v 1.3, como segurança, Java Native Interfaces (JNI), Remote Method Invocation (RMI), Java Virtual Machine Debugging Interface (JVMDI).

A configuração junto com o perfil, cria um ambiente J2ME. As configurações e os serviços suportados por uma configuração são escondidos do desenvolvedor da aplicação. Na verdade, o desenvolvedor é proibido de acessar estes serviços e configurações.

Na visão do desenvolvedor, o perfil é requerido para se trabalhar. Através dele é definida a camada que contém as APIs que o desenvolvedor manipulará.

Observa-se que os pacotes que definem a interface gráfica em Java, como Abstract Window Toolkit (AWT) e Swing são excluídos da configuração CDC. Se uma aplicação necessita de uma interface gráfica em J2ME, será necessário adicionar um perfil, que é construído sobre a configuração. Um perfil pode ser adicionado sobre outro; entretanto, em uma implementação J2ME, somente é permitida uma configuração.

Abaixo, gráfico com a comparação entre as bibliotecas Java J2SE, e as configurações CLDC e CDC.

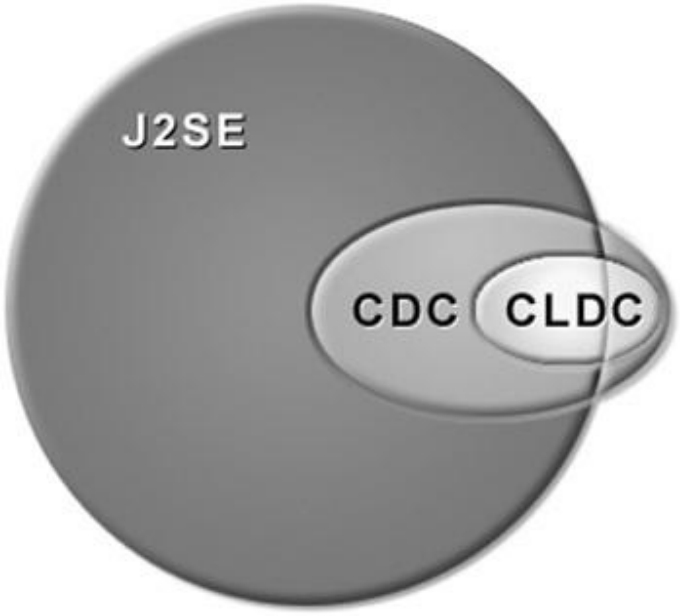

Figura 2 - Comparativo entre J2SE, CDC e CLDC. Fonte: [RIGS, 2003]

No próximo tópico será detalhado um importante conceito na pilha da arquitetura de J2ME, o perfil.

2.3. Perfis

O perfil é a camada no topo da configuração, responsável por adicionar APIs e especificações necessárias para desenvolver aplicações para famílias específicas de dispositivos[RIGS, 2003].

Diversos perfis estão sendo desenvolvidos sobre a JCP (Java Community Process), que são agrupados conforme a configuração sobre a qual ele é especificado. Os perfis MIDP (Mobile Information Device Profile) e IMP (Information Module Profile) são especificados sobre CLDC e os perfis FP (Foundation Profile), o PP (Personal Profile) e o PBP (Personal Basis Profile) sobre CDC.

O perfil IMP foi desenvolvido para aplicações de dispositivos que utilizam rede mas não possuem um rico display gráfico, ou com recursos muito limitados como os aparelhos de emergência, medidores de estacionamento, módulos wireless de sistemas de alarme residenciais, dispositivos medidores de indústrias e assim por diante. O IMP é um subconjunto do perfil MIDP versão 1.0.

O perfil MIDP é o mais utilizado atualmente pois ele atende aplicações dos dispositivos que são mais populares hoje em dia, como celulares e PDAs. Este perfil será mais detalhado adiante.

O Foundation Profile tem dois propósitos. O primeiro seria definir um perfil para dispositivos que possuem suporte a operações de rede, mas não utilizam interface gráfica. E o segundo propósito seria para servir como base de outros perfis com mais funcionalidades.

O Personal Profile é utilizado em dispositivos com total suporte a bibliotecas de interface gráfica ou applets, como os PDAs de alto nível, comunicadores e console de jogos. Este perfil utiliza a biblioteca completa de AWT (Java Abstract Window Toolkit) e possui um grande suporte ao desenvolvimento de aplicações baseadas em applets que são os mesmos que rodam em aplicações Desktop convencionais.

O Personal Basis Profile é utilizado em dispositivos com recursos reduzidos que necessitam de um framework de interface gráfica mais leve, sem todas as funcionalidades do AWT. Ele possui suporte ao modelo de programação baseado em xlet, e pode ser utilizado em aplicações de televisão interativa, na indústria automotiva, ou em um mercado consumidor definido.

2.3.1. – Mobile Information Device Profile (MIDP)

O Mobile Information Device Profile, segundo [RIGS, 2003], é uma arquitetura e um conjunto de bibliotecas Java que criam um ambiente de desenvolvimento para dispositivos pequenos e com recursos reduzidos, os chamados MIDs. Podemos citar os aparelhos celulares, pagers e organizadores pessoais como os dispositivos mais populares que se enquadram neste perfil. Estes dispositivos devem possuir, no mínimo, a seguinte especificação:

Memória	- 256 KB de memória não-volátil para componentes MIDP; - 8 KB de memória não-volátil para persistência de dados; - 128 KB de memória volátil para a máquina virtual;
Display	- Tamanho: 96 x 54; - Profundidade: 1 bit; - Divisão de pixel: 1:1;
Entrada de dados	No mínimo, um ou mais dos seguintes mecanismos: - teclado de 1 mão; - teclado de 2 mãos; - touch screen;
Rede	- bidirecional, wireless, possivelmente intermitente e com limitada largura de banda.
Som	Habilidade de tocar sons, utilizando hardware ou algoritmos de software.

A especificação MIDP estende as funcionalidades definidas na configuração CLDC, possuindo cobertura as seguintes áreas:

- modelo de aplicação: define a semântica da aplicação e como ela é controlada;

- interface gráfica: Limited Capability Device User Interface (LCDUI);

- rede: baseada no protocolo HTTP e no Generic Connection Framework introduzido pelo CLDC;

- mecanismo de persistência: Record Management System (RMS);

- sons;

- jogos 2D;

- segurança fim a fim através de HTTPS e sockets seguros;

- modelo de assinatura de MIDlet para adicionar segurança;

- entrega e instalação de aplicação;

- diversas classes como timers e exceções;

Além de adicionar estas áreas, o MIDP define uma extensão do modelo para execução e comunicação de aplicações, o chamado MIDlet. Um MIDlet é a unidade básica de execução de um MIDP. Um ou mais MIDlets empacotados juntos são chamados de suíte MIDlet.

Um MIDlet é uma classe que estende a interface javax.microedition.midlet.MIDlet e implementa alguns métodos – startApp, pauseApp e destroyApp – que definem o ciclo de vida da aplicação.

Os MIDlets possuem um ciclo de vida bem definido, podendo estar entre um de três estados, que seguem regras claras de transição:

- Pausado: um MIDlet está no estado pausado quando ele já foi construído, porém ainda não entrou no método startApp(), ou caso tenha sido invocado o método

pauseApp() ou notifyPaused(). Quando pausado, o MIDlet deve armazenar o mínimo de recursos possível. Neste estado ele pode receber notificações assíncronas, como por exemplo, um despertador.

- Ativo: um MIDlet está no estado ativo quando entra no método startApp(). A sua transição do estado pausado para ativo resulta no método resumeRequest(). Quando no estado ativo, um MIDlet pode alocar todos os recursos necessários para a sua execução.

- Destruído: um MIDlet é destruído quando ele retorna do método destroyApp() ou notifyDestroyed(). Uma vez que o MIDlet é destruído, ele não pode mais entrar em outro estado. O estado destruído substitui a chamada padrão para System.exit() nas aplicações convencionais. Caso seja chamado o método System.exit() em um MIDlet será gerada uma exceção do tipo java.lang.SecurityException.

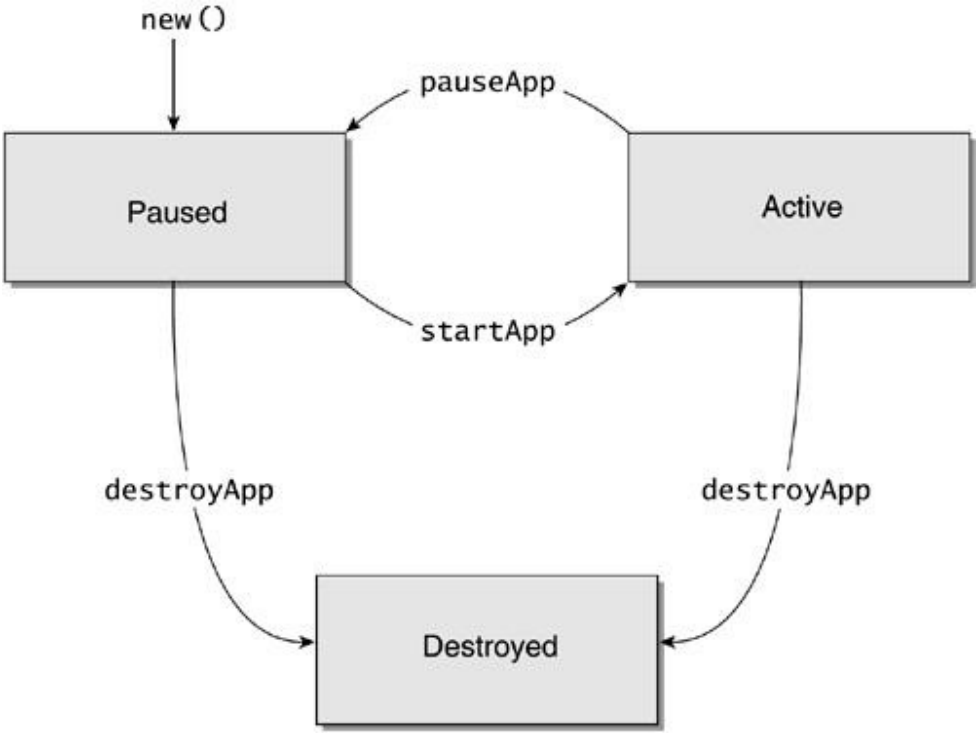

Figura 3 - Ciclo de vida de um MIDlet. Fonte: [RIGS, 2003]

A transição dos estados de um MIDlet pode ser gerada através do próprio sistema MIDP ou de uma requisição da aplicação. Conforme a figura 3, o construtor é invocado para criar uma instância do MIDlet. O método startApp() é chamado pelo MIDlet pra movê-lo para o estado ativo pela primeira vez, ou quando ele está sendo reiniciado de um estado pausado. O método pauseApp() é chamado para mover o MIDlet para o estado pausado. Durante a operação normal de um dispositivo móvel, o sistema deve funcionar em situações onde é necessário suspender ou pausar o MIDlet. O exemplo mais comum é quando o dispositivo possui pouca memória, e para requisitar mais memória, o sistema chama o método pauseApp() de todos os MIDlets ativos. O método destroyApp() é a forma convencional de encerrar um MIDlet.

2.4. Pacotes opcionais

Logo após a definição da configuração CLDC e do padrão MIDP, observou-se que além das configurações e dos perfis, existiam bibliotecas que não pertenciam a simples categorias ou famílias de dispositivos. Assim, para permitir uma melhor localização de bibliotecas que seriam aproveitadas por um grande número de dispositivos e famílias de dispositivos e que não estavam restritos a uma configuração, foi criado o conceito de pacotes opcionais, que nada mais é que um conjunto de bibliotecas que pode ser utilizado para estender um perfil, com a vantagem de ter suas funcionalidades independentes de qualquer perfil. Espera-se que os pacotes opcionais sirvam como um guia da evolução dos perfis, visto que bibliotecas são desenvolvidas conforme novas

tecnologias vão surgindo. A tendência é que tão logo estas bibliotecas fiquem maduras, com o surgimento de novas versões, elas sejam incorporadas aos perfis.

Abaixo, segue tabela identificando os pacotes opcionais especificados até o momento:

Pacote Opcional	Descrição
JSR 120: Wireless Messaging API	Define um conjunto de bibliotecas que padronizam recursos de comunicação. Utilizam: SMS (Short Message Service) e CBS (Cell Broadcast Service)
JSR 135: Mobile Media API (MMAPI)	Define um conjunto de biblioteca de multimídia, providenciando acesso e controle a recursos básicos de áudio, multimídia e arquivos.
JSR 172: J2ME Web Service Specification	Define padrões de acesso para acessar Web Services. Definindo uma infra-estrutura para processamento de XML, reuso de conceitos de Web Service no acesso a servidores, permitir interoperabilidade entre clientes J2ME e Web services.
JSR 177: Security and Trust Services for J2ME	Define um conjunto de bibliotecas que definem serviços de segurança aos

	dispositivos (serviços confiáveis)
JSR 179: Location API for J2ME	Define um conjunto de bibliotecas que permite aos desenvolvedores escrever aplicações que produzam informações sobre a localização física da aplicação, utilizando mecanismos de GPS e E-OTD.
JSR 180: Session Initiation Protocol (SIP) for J2ME	O SIP é utilizado para estabelecer e gerenciar sessões de IP multimídia – o mesmo mecanismo usado em serviços de mensagem instantânea.
JSR 184: Mobile 3D Graphics for J2ME	Define uma biblioteca leve e interativa de gráficos 3D que consume pouco processamento e memória.
JSR 190: Event Tracking API for J2ME	Define um padrão para envio de eventos para um servidor de eventos via um protocolo padrão. Estes eventos podem ser utilizados para pagamento de contas, notificação de atualizações, revisões, etc.

Agora que sabemos os conceitos utilizados na arquitetura J2ME, precisamos estudar os conceitos da arquitetura de Web Services, que será descrita nos próximos capítulos.

3. A ARQUITETURA DE WEB SERVICES

3.1. Visão Geral da Arquitetura de Web Services

A plataforma de web services pode ser definida, basicamente, como um componente de software que está disponibilizado na internet e utiliza o XML (eXtensible Markup Language) como formato padrão de troca de mensagens. Esta definição é considerada muito restrita, pois não explica quais as características de um web service, não permitindo ao desenvolvedor entendê-lo e utilizá-lo.

Figura 4 - Definição básica de Web Services. Fonte: [HENDRICKS, 2002]

Existem diversas definições de web services, segundo [HENDRICKS, 2002], um Web service é um pedaço de lógica de negócio localizado em algum lugar da Internet, que é acessível através de protocolos padrões da Internet como o HTTP ou SMTP.

Segundo [RODRIGUES, 2003], Web services são aplicações que fazem uso de um registro e padrões de comunicação para trabalhar de uma forma dinâmica (onde uma aplicação fornece transações, mensagens ou serviços para outras aplicações). Estas aplicações utilizam um formato definido (XML ou algum variante de XML) para apresentação de informação e dados; e eles utilizam alguns padrões da arquitetura de

Web service para encontrar serviços (UDDI), negociar como enviar e receber informações (WSDL), iniciar sessões de comunicação (SOAP), e transferir informação sobre a Internet (HTTP).

Segundo [RODRIGUES, 2003], um web service pode representar a evolução de tecnologias de componentes distribuídos como chamadas remotas de procedimento (RPC), ORPC - Object Remote Procedure Call (DCOM, Corba, Java RMI), mensagens (MSMQ – Microsoft Message Queuing), e modernas aplicações web (como Google.com).

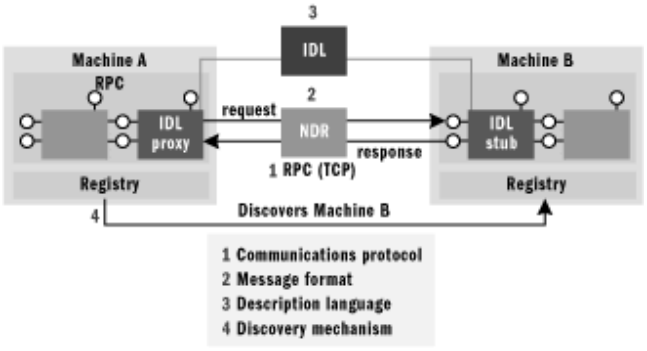

Figura 5 - Arquitetura DCOM. Fonte: [HENDRICKS, 2002]

Conforme a definição de [RODRIGUES, 2003], a tecnologia de RPC era considerada muito complexa, obrigando os desenvolvedores a criar camadas de objetos sobre o mecanismo RPC para esconder a sua complexidade. Isto fez muitos desenvolvedores partirem para o desenvolvimento utilizando a tecnologia ORPC (orientada a objetos). Porém, deve-se observar que nem todas as aplicações comunicam-se através de RPC, outros paradigmas para troca de mensagens também são utilizados, surgindo à necessidade de interoperabilidade entre estes sistemas.

Uma das maiores promessas de web services é o seu potencial para alcançar a interoperabilidade através de sistemas heterogêneos, plataformas, aplicações e linguagens de programação. As organizações passaram a dar mais atenção à necessidade de interoperabilidade, principalmente nas áreas de Enterprise Application Integration (EAI) e Business-to-Business Integration (B2Bi).

Um dos maiores desafios representados na área de EAI é integração entre aplicações que rodam em diferentes plataformas e servidores web. Estes sistemas precisam se comunicar e trocar informações, inclusive com outros dispositivos como impressoras e fax.

B2Bi representa integrações de negócios entre diferentes sistemas, ou seja, se um negócio for à realização de compra de estoque, os sistemas precisariam interagir e trocar informações. Geralmente esta troca de informação ocorre utilizando diferentes tecnologias. Muitas organizações gostariam de estender isto ao alcance de seus usuários, mas para que isto ocorra, os web services e os navegadores web precisam tornar-se disponíveis para a maioria das plataformas, tornando a interoperabilidade uma realidade.

O surgimento de aplicações web, ou seja, aplicações distribuídas construídas sobre navegadores web, foi inevitável. Para o usuário interagir com este tipo de aplicação ele precisa preencher formulários e executar ações que são enviadas através do protocolo HTTP para processamento e retorno de requisições. Devido à necessidade de intervenção humana como parte deste tipo de sistema, os navegadores web não são utilizados como um componente de aplicações distribuídas, eles somente fazem parte destas.

Ao mesmo tempo em que o "*ponto-com*" se tornou uma palavra conhecida em todo o mundo, o XML começou a chamar mais atenção das indústrias como um padrão amplamente aceito para a representação de dados. Em pouco tempo a indústria adicionou novas camadas de serviços ao XML como DOM (Document Object Model)[COUNSORTIUM, 1998], XPath (XML Path Language)[COUNSORTIUM, 1999], XSLT[COUNSORTIUM, 1999], XMLSchema[CONSORTIUM, 2004], SOAP[BOX, 2000], WSDL[CHRISTENSEN, 2001] e UDDI[BELLWOOD, 2002]. Assim, é possível através destas tecnologias, oferecer soluções robustas para preencher a deficiência das aplicações web.

Com a perspectiva de manter a interoperabilidade, ocorreu uma revolução no desenvolvimento de sistemas distribuídos, onde os aspectos de design das aplicações web foram combinados com os benefícios das tecnologias XML, surgindo à arquitetura de Web Services - construída sobre padrões, protocolos e linguagem amplamente aceitos, e que possuem as seguintes características:

- protocolos de comunicação padronizados;
- padrões para representação de dados;
- linguagens padronizadas de descrição;
- mecanismos de descoberta padronizados.

Uma característica que distingue a plataforma de Web Services de um sistema ORPC é a sua interface de sistema (wire-contract). Muitos sistemas ORPC escondem detalhes da interface do desenvolvedor. Os desenvolvedores simplesmente descrevem suas interfaces através de IDL (ou linguagem semelhante) e um compilador gera o código para efetuar a transmissão da requisição através da rede. Esta infra-estrutura trabalha muito bem quando ambos os sistemas utilizam ORPC, porém, tratando de

sistemas interoperáveis, esta estrutura não trabalha muito bem. Quando se utiliza Web Services, é necessário definir primeiramente esta interface, feita em termos de diversas tecnologias XML, cada uma com uma regra definida dentro da plataforma. Algumas destas tecnologias ajudam a representar, descrever e descobrir a interface do sistema de um web service:

- XML + Namespaces;
- SOAP;
- XML Schema;
- WSDL;
- UDDI;

A utilização de XML + Namespaces serve para definir a sintaxe para a representação de um dado em XML. Utilizando XML os desenvolvedores podem representar seus dados em um formato portável.

O XML Schema possibilita descrever o tipo e a estrutura dos documentos XML. A definição de um XML Schema serve como uma documentação oficial para um conjunto de mensagens XML.

O SOAP permite estender o mecanismo de enquadramento de mensagens XML, sendo semelhante ao protocolo HTTP ao tratar da distinção entre cabeçalhos e corpo das mensagens. Ele também é responsável pela definição de padrões para representação de erros e localização para HTTP e RPC.

WSDL é uma camada adicional, definida por um XML Schema, empregada para descrever as interfaces de web services. Através de WSDL é possível gerar classes

Proxy e Stub que sabem como invocar operações do web service sem a intervenção do desenvolvedor.

UDDI é utilizado para padronizar como os Web Services são descobertos. Ela define um repositório centralizado de web services baseando-se em uma API SOAP.

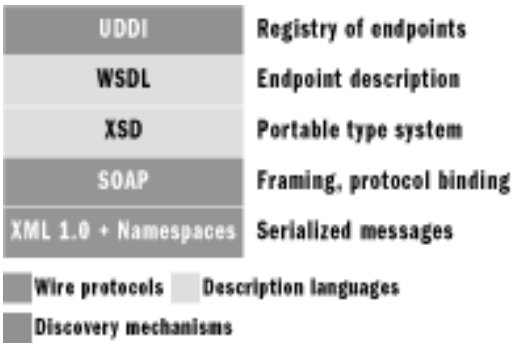

Figura 6 - Plataforma de Web Services. Fonte: [HENDRICKS, 2002]

Nos capítulos posteriores serão detalhadas todas as tecnologias que são utilizadas na arquitetura de web services, que agora, segundo Hendricks[HENDRICKS, 2002], pode ser definida como:

- uma tecnologia que se comunica via protocolos abertos (HTTP, SMTP, etc);
- processa mensagens XML utilizando SOAP;
- descreve suas mensagens utilizando XML Schema;
- permite a descrição de um serviço utilizando WSDL;
- pode ser descoberto utilizando UDDI;

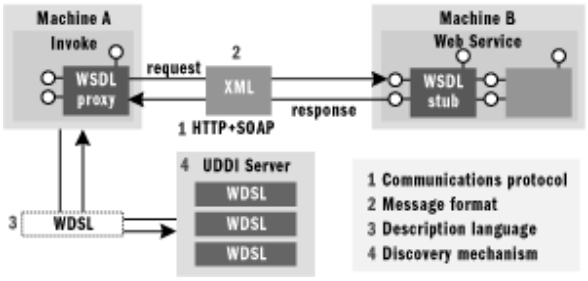

Figura 7 - Arquitetura de Web Services. Fonte: [HENDRICKS, 2002]

3.2. XML

XML, Extensible Markup Language, é uma linguagem de marcação de dados, que provê um formato para descrever dados estruturados, facilitando a declaração mais precisa de conteúdos e resultados mais significativos de busca através de múltiplas plataformas.

O XML foi inicialmente projetado como um formato de dados para facilitar o recebimento, o processamento e a geração de informações na Web de uma maneira simplificada, assim como o HTML é utilizado para descrever a forma como dados devem ser exibidos para o usuário. Enquanto no HTML utiliza-se tags (markup tags) pré-definidas para visualizar uma página, no XML, estas as tags são definidas conforme a sua necessidade, da maneira mais coesa para representar um dado.

A tag é responsável pela separação de elementos. Através das tags é possível também a inserção de comentários e de instruções especiais, além de declarar configurações em um ambiente de parsing.

Existem diferentes tipos de elementos XML, conforme a tabela abaixo:

Objeto	Proposta	Exemplo
Elemento vazio	Representa uma	<xref link=”abc” />

	informação específica em um ponto do documento.	
Elemento preenchido	Agrupa elementos e caracteres.	Este é um exemplo
Declaração	Adiciona um parâmetro, entidade, ou definição de gramática em um ambiente de configuração.	<!ENTITY author “Lucas Gomes”>
Instruções de processamento	Define instruções particulares de processamento de um software.	<?print-formatter force-linebreak?>
Comentário	Insere uma anotação que será ignorada pelo processador de XML.	<!— este texto não será processado -->
Referência à entidade	Comando enviado ao parser para inserir texto	&minha-empresa

Os elementos são os tipos de objetos mais comuns em XML. Eles podem dividir um documento em pequenas células, como se fossem caixas dentro de caixas. Cada uma dessas peças possui suas próprias propriedades e regras em um documento, conforme sua definição.

Abaixo, segue exemplo de um documento XML que define um telegrama.

```
<?xml version="1.0"?>
<telegrama prioridade="importante">
    <destinatario>Sarah Bellum</destinatario>
```

```
    <remetente>Colonel Timeslip</remetente>
    <assunto>Ausência da Empresa</assunto>
<figura fileref="figs/me.jpg"/>
<mensagem>Caro
<nome>Zonky</nome>
A partir de amanhã estarei ausente por
<enfase>dois dias</enfase>
e gostaria que você cuidasse dos negócios da empresa nesse período.
</mensagem>
</telegrama>
```

No capítulo seguinte, será detalhada a estrutura de um documento XML.

3.2.1. Documento XML

Um documento XML é um conceito especial definido para armazenar um dado para ser analisado através de um parser. Seu conceito não está diretamente relacionado a um documento convencional, como um livro ou uma revista, onde os textos destes documentos podem ser armazenados como parte de um documento XML. Deve-se ter em mente que um documento é definido em termos lógicos, não físicos.

Um documento XML é composto por duas partes, a primeira é o prólogo, uma seção especial, opcional que contém um metadado. E a segunda parte é chamada de elemento do documento, ou root, no caso de árvores XML. Neste texto, irei abordar somente a segunda parte que são os Elementos.

Os elementos são parte do XML que dividem o documento em regiões hierárquicas, cada uma com um propósito específico. Alguns elementos possuem textos dentro deles, não possuem nenhuma entrada, ou possuem outros elementos.

Para adicionar mais informação sobre um elemento, é possível adicionar atributos ao elemento. Um atributo é o par nome-valor. Uma razão para utilizar atributos é para distinguir elementos com o mesmo nome.

Um outro mecanismo para associar elementos e atributos a um grupo é o namespace. Ele é utilizado para combinar diferentes vocabulários no mesmo documento, por exemplo:

```
<part-catalog
xmlns:nw="http://www.nutware.com/"
xmlns="http://www.bobco.com/"
>
<nw:entry nw:number="1327">
<nw:description>torque-balancing hexnut</nw:description>
</nw:entry>
<part id="555">
<name>type 4 wingnut</name>
</part>
</part-catalog>
```

Um namespace só afeta a área do documento. O elemento contendo a declaração e todos os seus descendentes são escopos do namespace. Também é possível sobrescrever um namespace criando outro dentro dele com o mesmo nome.

```
<flavor:chocolate-shell
xmlns:flavor="http://www.deliciouscandy.com/chocolate/">
<flavor:chewy-center
xmlns:flavor="http://www.deliciouscandy.com/caramel/">
```

```
<flavor:walnut/>
</flavor:chewy>
</flavor:chocolate-shell>
```

No exemplo acima, o namespace xmlns:flavor é declarado duas vezes. O namespace foi uma grande adição ao XML, porém, ele foi adicionado após a especificação de XML, causando um certo problema, pois não existe maneira para escrever um DTD (Document Type Definition - responsável por definir a estrutura lógica de um documento) definindo um namespace para ele.

Um outro elemento de um documento XML são as entidades. As entidades podem ser declaradas em prólogo ou em um DTD. Existem diversos tipos de entidades com diversas formas de se usar. Por exemplo, a entidade pode ser utilizada para declarar o nome de um cliente que é difícil de se digitar.

```
<?xml version="1.0"?>
<!DOCTYPE message SYSTEM "/xmlstuff/dtds/message.dtd"
[
<!ENTITY client "Mr. Rufus Xavier Sasperilla">
<!ENTITY agent "Ms. Sally Tashuns">
<!ENTITY phone "<number>617-555-1299</number>">
]>
<message>
<opening>Dear &client;</opening>
<body>We have an exciting opportunity for you! A set of
ocean-front cliff dwellings in Pi&#241;ata, Mexico, have been
renovated as time-share vacation homes. They're going fast! To
reserve a place for your holiday, call &agent; at &phone;.
Hurry, &client;.
```

Os outros elementos de XML, o comentário, as instruções de processamento e a seção CDATA, possuem coisas em comum: eles definem a utilização do parser de alguma forma. Os comentários são ignorados pelo parser, o conteúdo da seção CDATA também é ignorado pelo parser, e as instruções de processamento define regras para o parser.

3.3. HTTP

O HTTP (HyperText Transfer Protocol) é o protocolo de nível de aplicação responsável pelo funcionamento da World Wide Web. Torna possível a transmissão dos documentos hipermídias, como textos, gráficos, som, imagem e outros recursos disponíveis na Web para posterior visualização na máquina cliente.

Apesar da aparente complexidade da Web, o protocolo HTTP é relativamente simples. As especificações do protocolo HTTP são úteis para todos que trabalham ativamente na criação de sites Web, manutenção de servidores Web, ou com desenvolvimento de programas-clientes e servidores que venham a interagir com a Web.

O protocolo HTTP é baseado no modelo pedido/resposta (request/response), onde um cliente estabelece uma comunicação com o servidor e envia um pedido. O servidor analisa o pedido, e devolve uma resposta. A conexão deve ser estabelecida antes de cada pedido de cliente e encerrada após a resposta.

Um pedido é uma mensagem enviada de um cliente ao servidor com o método que será aplicado ao recurso, o identificador do recurso, e a versão do protocolo em uso.

Os seguintes métodos podem ser enviados ao servidor: OPTIONS, GET, HEAD, POST, PUT, DELETE, TRACE e CONNECT [ORTIZ, 2004].

O método OPTIONS é responsável por requisitar uma informação sobre o canal de informação durante uma requisição request/response identificada através da "Request-URI". Este método permite ao cliente a determinar as opções e requerimentos associados a um recurso, ou a capacidade do servidor.

O método GET recupera informação, na forma de entidade, através da Request-URI. Se a Request-URI refere-se a um processo de dado, então este dado será retornado como uma entidade resposta.

O método HEAD é idêntico ao GET exceto pelo fato do servidor não retornar um corpo na mensagem de resposta. A meta-informação contida no cabeçalho HTTP devem ser igual nos métodos GET e HEAD.

O método POST é utilizado por uma requisição que aceita a entidade enviada pelo request como parte da Request-URI.

O PUT solicita a entidade a ser armazenada sobre o Request-URI.

O DELETE solicita ao servidor que remova o recurso identificado pela Request-URI.

O TRACE é usado para invocar uma chamada remota de loopback de uma requisição.

E o método CONNECT é usado como um Proxy que pode dinamicamente ser alterado para um túnel (SSL Tunneling).

3.4. SOAP

3.4.1. – Definição de SOAP

Segundo [HENDRICKS, 2002], SOAP é um protocolo superficial que exporta informação de forma centralizada em um ambiente distribuído. Ele é um protocolo baseado em XML que consiste de três partes: um envelope que define um framework para descrever o que é uma mensagem e como processá-la; um conjunto de regras de codificação para expressar instâncias de tipos definidos pela aplicação; e uma convenção para representar chamadas remotas de procedimento e respostas.

O SOAP não está vinculado a uma plataforma de hardware, a um sistema operacional, linguagem de programação ou hardware de rede. Diferentemente de outros sistemas de computação distribuídos, o SOAP é construído no topo de padrões abertos, como HTTP e XML. A utilização de padrões amplamente aceitos e conhecidos facilita o aprendizado, por parte dos desenvolvedores, da nova tecnologia, e também da infra-estrutura existente para suportá-la.

O protocolo SOAP é considerado superficial, pois contém menos recursos do que outros protocolos de computação distribuídos, o que torna o protocolo menos complexo. Além disso, o SOAP não define um mecanismo para a descrição e localização de objetos em sistemas distribuídos. Esses mecanismos são definidos pelas especificações WSDL e UDDI, que serão discutidas posteriormente.

A especificação SOAP consiste em duas partes: encapsulamento de mensagens e de RPC. A parte relativa às mensagens da especificação define um framework de

mensagens para a transmissão de mensagens entre sistemas distribuídos. À parte da especificação relativa às mensagens define como embutir chamadas e respostas de procedimento remoto dentro das mensagens, para efeito de chamada de procedimentos em sistemas remotos.

A parte da especificação relativa ao encapsulamento da RPC permite que o SOAP forneça uma abstração na camada do protocolo para protocolos de comunicação distribuídos dentro de uma intranet ou na Internet. Essa abstração fornece sistemas remotos com acesso a objetos através de plataformas, sistemas operacionais e ambientes que seriam, de outra maneira, incompatíveis. Por exemplo, uma camada de SOAP pode ser adicionada a um objeto Java e CORBA. Essa camada cria um ambiente de computação distribuído comum para os dois objetos – tornando os objetos do CORBA acessíveis pelos objetos Java e vice-versa, sem que você precise se preocupar com as peculiaridades de cada protocolo de computação distribuído.

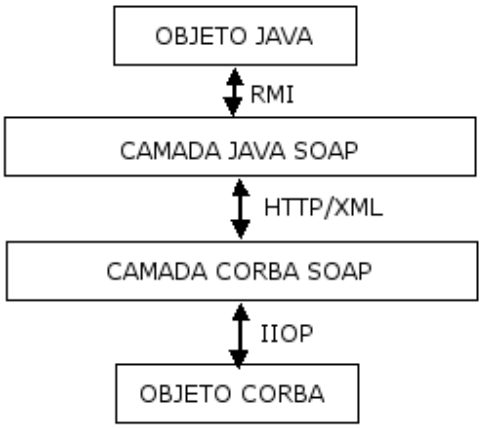

Figura 8 - Abstração das camadas do protocolo SOAP

No exemplo acima, o objeto Java se comunica com uma camada SOAP (Camada Java SOAP) usando o RMI, enquanto o objeto CORBA se comunica com a camada SOAP (Camada CORBA SOAP). As camadas SOAP se comunicam umas com as outras, trocando mensagens SOAP, codificadas em XML, utilizando o protocolo HTTP.

3.4.2. – Vantagens do SOAP

O SOAP proporciona várias vantagens em relação a outros sistemas distribuídos, entre as quais podemos citar:

- O SOAP pode atravessar firewalls com facilidade;

- Os dados do SOAP são estruturados usando XML;

- O SOAP pode ser usado em combinação com vários protocolos de transporte, como HTTP, SMTP e JMS.

- O SOAP mapeia satisfatoriamente para o padrão de solicitação/resposta (request/response) HTTP e do HTTP Extension Framework.

- O SOAP é razoavelmente superficial como um protocolo, pois contém menos recursos do que outros protocolos de computação distribuídos.

- Existe suporte para SOAP por parte de vários fornecedores.

Devido à possibilidade do SOAP poder ser usado com o protocolo HTTP, ele pode transpor firewalls com facilidade, permitindo que aplicações estejam disponíveis internamente (através da intranet) e externamente (Internet) desde a sua distribuição inicial. Isto tem tudo para se tornar um problema sério para a segurança, onde as aplicações do SOAP seriam acessíveis por partes não autorizadas.

Uma outra vantagem proporcionada pelo SOAP é que as mensagens são transmitidas usando o protocolo XML, baseado em texto. Ou seja, as mensagens podem ser compreendidas por quase todas as plataformas de hardware, sistemas operacionais, linguagens de programação ou hardware de rede.

Abaixo, segue o modelo de mensagens SOAP para solicitações e respostas HTTP.

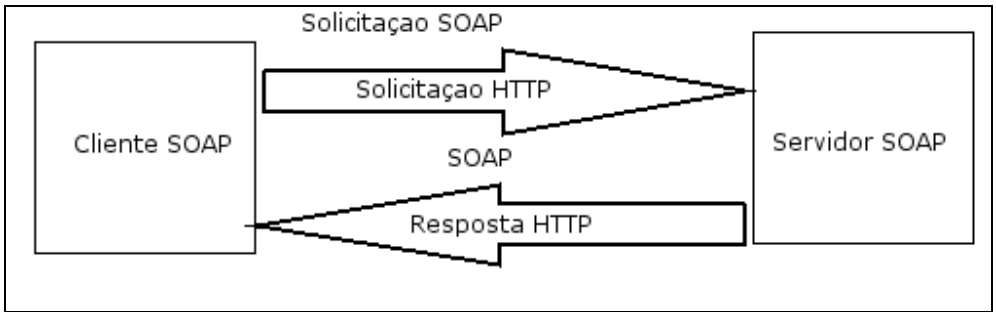

Figura 9 - Modelo de mensagens SOAP/HTTP

No modelo acima, um cliente SOAP faz uma solicitação SOAP incorporada em uma mensagem de solicitação HTTP. O servidor SOAP retorna uma mensagem de resposta SOAP, incorporada em uma mensagem de resposta HTTP.

3.4.3. – Desvantagens do SOAP

Devido à falta de recursos do protocolo SOAP, ele pode apresentar algumas desvantagens:

- Falta de interoperabilidade entre kits de ferramentas do SOAP;

- Mecanismos de segurança são imaturos;

- Não existe garantia quanto à entrega da mensagem;

Embora o SOAP possua um amplo suporte de vários fornecedores, ainda existem problemas de incompatibilidade entre diferentes implementações do SOAP.

O SOAP não define um mecanismo para a autenticação de uma mensagem antes que ela seja processada. Também não define um mecanismo para a criptografia do conteúdo de uma mensagem SOAP, o que evitaria que outros tivessem acesso ao conteúdo da mensagem.

Caso ocorra alguma falha enquanto uma mensagem estiver sendo transferida, um sistema que permita SOAP não saberá como reenviar a mensagem.

Um cliente SOAP não pode enviar uma solicitação a vários servidores, sem enviar a solicitação a todos os servidores.

3.4.4. – O papel de XML no SOAP

O SOAP utiliza XML para codificar todas as mensagens. Devido a sua simplicidade, as mensagens SOAP não podem conter um DTD. Ela deve incluir os *namespaces* característicos de XML em todos os elementos e atributos definidos pelo SOAP. A vantagem de utilizar *namespaces* no SOAP é que os elementos definidos não entrarão em conflito com os elementos padrão do SOAP. Sendo assim, todos os elementos e atributos padrão do SOAP recebem um prefixo com o identificador do *namespace* *SOAP-ENV*¸ que está associado ao *namespace* http://schemas.xmlsoap.org/soap/envelope.

3.4.5. – Mensagem SOAP

Uma mensagem SOAP é um documento XML que pode conter três partes: o envelope SOAP, o cabeçalho SOAP e o corpo SOAP.

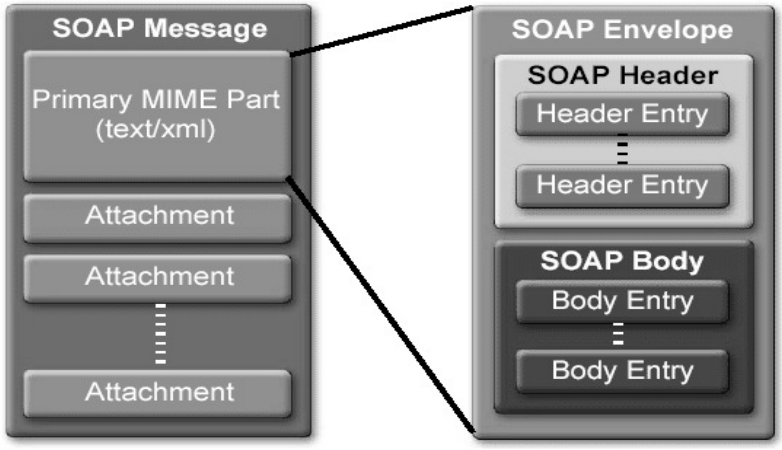

Figura 10 - Mensagem SOAP. Fonte: [BOX, 2004]

O envelope SOAP é o elemento de nível mais elevado de uma mensagem SOAP. Este contém um cabeçalho SOAP opcional e um corpo SOAP obrigatório. O envelope SOAP é designado em uma mensagem SOAP pelo elemento <Envelope>. Abaixo, segue ilustração de um envelope SOAP.

```
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV=”http://schemas.xmlsoap.org/soap/envelope/”
   SOAP-ENV:encodingStyle=”http://schemas.xmlsoap.org/soap/encoding/”>
   <SOAP-ENV:Header>
     ... //opcional
   </SOAP-ENV:Header>
   <SOAP-ENV:Body>
     ... //obrigatório
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Exemplo de um envelope SOAP 1

No exemplo acima, foi utilizado o atributo global *encodingStyle* para especificar como as informações devem ser codificadas.

O cabeçalho SOAP é uma maneira flexível e geral de adicionar recursos como autenticação, gerenciamento de transação e serviços de pagamento a uma mensagem SOAP. Um cabeçalho SOAP é especificado em uma mensagem SOAP, através do elemento *SOAP-ENV:Header*. Este cabeçalho não é obrigatório, contudo, se for utilizado, deverá vir imediatamente depois do elemento envelope SOAP. Um cabeçalho SOAP pode conter elementos filhos, que são representados como entradas do cabeçalho dentro da especificação SOAP.

Abaixo, segue exemplo da utilização de cabeçalho SOAP:

```
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV=”http://schemas.xmlsoap.org/soap/envelope/”
   SOAP-ENV:encodingStyle=”http://schemas.xmlsoap.org/soap/encoding/”>
   <SOAP-ENV:Header>
      <a:authentication
           xmlns:a=”http://www.meuexemplo.com/soap/authentication”>
         <a:username>meu_usuario</a:username>
         <a:password>minha_senha</a:password>
      </a:authentication>
   </SOAP-ENV:Header>
   <SOAP-ENV:Body>
     ...
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Exemplo de cabeçalho SOAP 1

O corpo SOAP contém a carga útil (informações) que deve ser recebida pelo receptor da mensagem, que é especificada usando o elemento *Body*. A carga útil consiste, tipicamente, em uma chamada RPC, resposta RPC, ou relatório de erro, porém, teoricamente pode conter qualquer tipo de informação.

Caso um elemento de cabeçalho esteja presente, o elemento *Body* deve seguir imediatamente o elemento *Header*. O elemento *Body* pode conter elementos filhos que são denominados entradas do corpo. Uma entrada do corpo é identificada por um *namespace.* Os elementos filhos imediatos de uma entrada de corpo não precisam ser qualificados por um *namespace.*

Para o tratamento de erros em uma aplicação SOAP, é utilizado o elemento *Fault*. Este elemento só pode ocorrer uma única vez dentro de uma mensagem SOAP, definindo quatro sub-elementos: *faultcode, faultstring, faultactor* e *detail.* Sendo *faultcode* o único elemento obrigatório, criado para proporcionar um mecanismo fácil de identificação de uma falha.

Abaixo, segue tabela contendo os códigos de falhas e seus significados, segundo a especificação SOAP 1.1:

Nome do faultcode	**Significado**
VersionMismatch	A parte relativa ao processamento encontrou um namespace inválido para o elemento do envelope SOAP.
MustUnderstand	Um elemento filho imediato do elemento cabeçalho SOAP, que não foi compreendido ou não foi obedecido pela parte relativa ao processamento e que continha um atributo *mustUnderstand* do sOAP com um valor *1.*
Client	A classe *Client* de erros indica que a

	mensagem foi formada incorretamente, ou que não continha as informações apropriadas, para ser bem sucedida.
Server	A classe *Server* de erros indica que a mensagem pode não ser processada, por motivos não diretamente atribuíveis ao conteúdo da mensagem propriamente dita, mas devido ao processamento da mensagem.

O sub-elemento *faultstring* pode ser utilizado para fornecer em linguagem legível a humanos, uma explicação da falha que ocorreu.

O sub-elemento *faultactor* destina-se especificar informações sobre o motivo da falha em uma mensagem, sendo útil, se uma mensagem SOAP passa através de várias aplicações intermediárias.

3.4.6. – Codificação do SOAP

A especificação SOAP 1.1 define como os dados na carga útil da mensagem SOAP podem ser codificados, no entanto, não define um novo sistema de tipos, esta se baseia na especificação "XML Schema Part 2: Datatypes" (http://www.w3.org/TR/xmlschema-2/).

Entretanto, o mecanismo de codificação do SOAP pode não ser usado e pode ser definido pelo desenvolvedor.

A especificação define dois grupos de tipos: tipos simples e tipos complexos. A diferença entre estes tipos é que os tipos simples não podem ter filhos de elementos ou atributos, enquanto os tipos complexos podem ter filhos de elementos e atributos.

A especificação “XML Schema Part 2: Datatypes” define 44 tipos simples incorporados que podem ser especificados usando o atributo *xsi-type* em um elemento. Segue abaixo, exemplo, com a definição de um tipo simples:

```
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV=”http://schemas.xmlsoap.org/soap/envelope/”
   SOAP-ENV:encodingStyle=”http://schemas.xmlsoap.org/soap/encoding/”>
   <SOAP-ENV:Header>
        ...
   </SOAP-ENV:Header>
   <SOAP-ENV:Body>
       <cmd:processReboot xmlns:cmd=”http://www.exemplo.com/soap/cmd”>
            <ip xsi:type=”xsd:string”>192.168.10.1</ip>
            <delay xsi:type=”xsd:int”>2000</delay>
       </cmd:processReboot>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Exemplo de definição de tipo simples 1

Neste exemplo foram criados os tipos *ip* e *delay*, que são tipos de dados *string* e *int*, respectivamente.

Alternativamente ao modelo definido no exemplo acima, um tipo pode ser definido através do *namespace SOAP-ENC.* Este *namespace* declara um elemento para cada tipo de dado simples.

```
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV=”http://schemas.xmlsoap.org/soap/envelope/”
   SOAP-ENV:encodingStyle=”http://schemas.xmlsoap.org/soap/encoding/”>
   <SOAP-ENV:Header>
        ...
   </SOAP-ENV:Header>
   <SOAP-ENV:Body>
       <cmd:processReboot xmlns:cmd=”http://www.exemplo.com/soap/cmd”>
            <SOAP-ENC:string id=”ip”>192.168.10.1</SOAP-ENC:string>
            <SOAP-ENC:int id=”delay”>2000</SOAP-ENC:int>
```

```
        </cmd:processReboot>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Exemplo 2 de definição de tipo simples 1

O SOAP utiliza o conceito de um *accessor*, para obter um valor codificado. O *accessor* é o nome do elemento que circunda um valor, ou o valor do atributo id.

Existem dois tipos de valores que podem ser usados em uma carga útil do SOAP. O primeiro tipo é denominado referência-única, o que significa que um valor é acessado apenas por um *accessor*. O segundo tipo do valor é denominado multi-referência, o que significa que um ou mais *accessors* pode acessar um valor.

Além de definir os tipos simples, o SOAP é constituído por dois tipos compostos, o *struct* e o *array*. O tipo composto *struct* permite que valores de tipos diferentes sejam agrupados, sendo cada valor armazenado e recebido usando um único nome de *accessor*. O tipo composto *array* consiste em valores que são armazenados e recuperado usando um número ordinal.

Os arrays podem ser definidos como tendo um tipo *SOAP-ENC:Array* ou um tipo derivado dele. Os elementos contidos dentro de um array podem ser de qualquer tipo, incluindo arrays aninhados.

3.5. WSDL

3.5.1. – Definição de WSDL

Segundo [CHRISTENSEN, 2001], WSDL é um formato XML para descrever serviços de rede como um conjunto de operações nas mensagens contidas em documentos ou procedimentos orientados a informação.

Um documento WSDL representa a interface externa de um serviço Web. Entretanto, além de descrever a interface apresentada, um documento WSDL também contém a localização do serviço. O registro do serviço está disponível em um local previsível e bem conhecido na rede.

Para permitir maior flexibilidade, a WSDL apresenta a definição de um serviço Web em duas partes: a primeira é responsável por representar uma definição abstrata independente do protocolo de transporte de alto-nível de um serviço Web, enquanto a segunda representa uma descrição específica do protocolo utilizado na camada de transporte de rede.

Abaixo, seguem os componentes que compõem um serviço:

- Tipos: são os tipos de dados (String, int, Object, entre outros). Representado em um documento WSDL pelo elemento <types>;

- Mensagem: são os parâmetros de entrada e saída de um serviço. Representado em um documento WSDL pelo elemento <message>.;

- Operações: são os relacionamentos entre parâmetros de entrada e saída, ou seja, a assinatura do método; Representado em um documento WSDL pelo elemento <operation>;

- Tipo de porta: É o agrupamento lógico de operações, podendo ser comparado com uma classe, que é um agrupamento lógico de métodos; Representado em um documento WSDL pelo elemento <portType>;

- Vínculo: É o protocolo utilizado para acessar os métodos de um objeto; Representado em um documento WSDL pelo elemento <binding>;

- Serviço: É o endereço do serviço; Representado em um documento WSDL pelos elementos <service> e <port>.

Na tabela abaixo, segue o exemplo de um documento WSDL que descreve um serviço de consulta à cotação de ações negociadas em bolsas de valores [CHRISTENSEN, 2001]:

```
<?xml version="1.0"?>
<definitions name="StockQuote"
targetNamespace="http://example.com/stockquote.wsdl"
        xmlns:tns="http://example.com/stockquote.wsdl"
        xmlns:xsd1="http://example.com/stockquote.xsd"
        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
        xmlns="http://schemas.xmlsoap.org/wsdl/">

   <types>
     <schema targetNamespace="http://example.com/stockquote.xsd"
          xmlns="http://www.w3.org/2000/10/XMLSchema">
       <element name="TradePriceRequest">
         <complexType>
           <all>
             <element name="tickerSymbol" type="string"/>
           </all>
         </complexType>
       </element>
     </schema>
   </types>
   <message name="GetLastTradePriceInput">
     <part name="body" element="xsd1:TradePriceRequest"/>
   </message>
   <portType name="StockQuotePortType">
     <operation name="GetLastTradePrice">
       <input message="tns:GetLastTradePriceInput"/>
```

```
        </operation>
      </portType>
      <binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
        <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="GetLastTradePrice">
          <soap:operation soapAction="http://example.com/GetLastTradePrice"/>
          <input>
            <soap:body use="literal"/>
          </input>
          <output>
            <soap:body use="literal"/>
          </output>
        </operation>
      </binding>
      <service name="StockQuoteService">
        <documentation>My first service</documentation>
        <port name="StockQuotePort" binding="tns:StockQuoteBinding">
          <soap:address location="http://example.com/stockquote"/>
        </port>
      </service>
    </definitions>
```

Observe que no exemplo acima foram utilizados os seguintes elementos: <types>, <message>, <portType> e <binding>, responsáveis por definir um documento WSDL.

3.6. UDDI

3.6.1. – Definição de UDDI

Segundo [BELLWOOD, 2002], Universal Description, Discovery and Integration, ou UDDI, é o nome do grupo de registros baseados na web que expõem informações sobre negócios ou outras entidades e suas interfaces ou APIs. Estes registros funcionam sobre múltiplos sites operadores, que podem ser utilizados por qualquer um que gostaria de disponibilizar alguma informação sobre seus negócios ou entidades.

O conceito relacionado com o UDDI provém do fato que a tecnologia dos serviços Web lida com a comunicação de aplicações, sem que seja necessária uma grande intervenção humana. Todos devem ter condições de usar qualquer serviço, independente de plataforma e linguagem de programação. E o principal conceito que dá embasamento a esse ponto é o do SOA (Service Oriented Architecture), que se define em três papéis:

- requisitante do serviço;
- provedor do serviço;
- agente do serviço (broker);

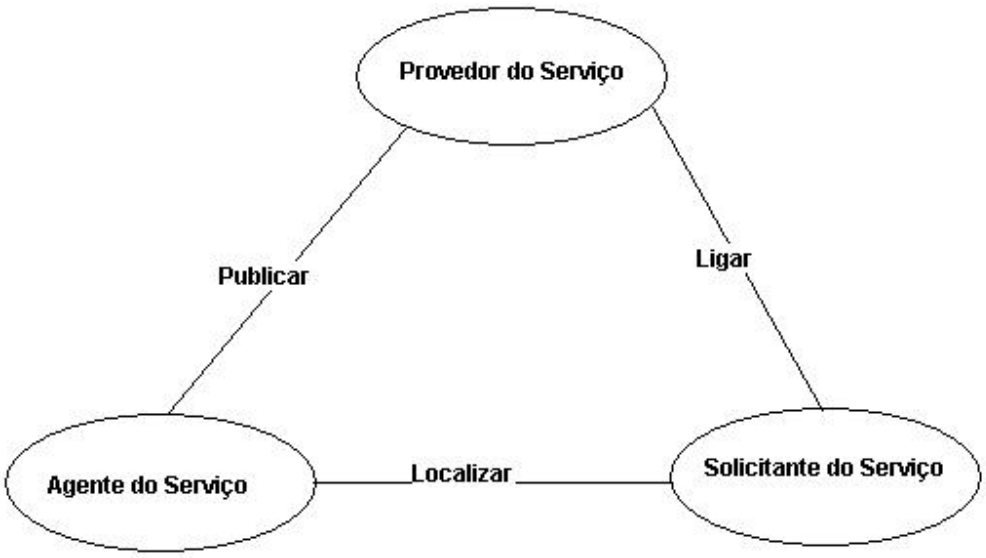

Figura 11 - Modelo SOA

O agente de serviço costuma ser representado por um registro que contém referências aos serviços disponíveis. Este registro de serviços Web é o UDDI, que contém informações sobre os serviços existentes e as empresas que os oferecem.

Os registros trocam todas as suas informações entre si, para que cada registro sempre contenha todas as informações disponíveis.

Segundo Hendricks [HENDRICKS, 2002], o UDDI consiste em quatro especificações principais:

- a especificação da estrutura de dados descreve que tipo de dados é armazenado no UDDI. Esta estrutura de dados baseia-se em XML para permitir a neutralidade em relação à plataforma e à linguagem de programação.

- a especificação da API do programador informa como se acessa um registro UDDI. Existem dois tipos de API: as funções de publicação e as funções de pesquisa.

- a especificação de replicação que contém as descrições sobre a maneira como os registros trocam informações entre si.

- a especificação do operador que se refere apenas aos que estão implementando ou executando um registro UDDI, definindo as políticas de segurança e de gerenciamento dos dados.

4. J2ME WEB SERVICES API (WSA)

4.1. Visão Geral de J2ME Web Services

A natureza de um Web Service de oferecer funcionalidades distribuídas sobre a Internet independente de sistema operacional e linguagem de programação já era muito explorada através de aplicações convencionais, utilizando-se J2SE e J2EE. Entretanto, com o surgimento de J2ME, criou-se uma nova oportunidade de mercado, onde os serviços que antes eram acessados através de computadores desktop, agora podem ser acessados de qualquer lugar.

Através do uso de tecnologias como SOAP, WSDL, XML e RPC, os desenvolvedores de dispositivos móveis podem criar aplicações clientes que são distribuídas, portáveis e que podem ser utilizadas em telefones celulares, PDAs e carregadas pelos seus clientes para qualquer lugar.

A especificação J2ME Web Services[ELLIS, 2004] (JSR-172 ou WSA) foi especialmente projetada para facilitar o desenvolvimento de pequenas, rápidas e robustas aplicações do mercado wireless. Esta especificação fornece dois pacotes opcionais baseados em XML que podem ser utilizados em dispositivos com pouca memória, como telefones celulares, handhelds, PDAs e pagers. Ao longo deste capítulo, serão explicadas qual a infra-estrutura desta especificação e suas características.

4.2. Entendendo JSR-172

O grande interesse e atividade da comunidade Java no uso de padrões de web services para construção de serviços, fez surgir a necessidade da utilização destes padrões no desenvolvimento de clientes J2ME no acesso a serviços corporativos. Ao comparar a arquitetura de Web Services de J2ME observa-se que ela segue as mesmas especificações, arquiteturas e modelos de invocação padrão de Web Services, WS-I Basic Profile 1.0. Ou seja, ela especifica:

- SOAP 1.1: que define como um dado é transportado e codificado.

- WSDL 1.1 : que define como descrever serviços remotos.

- XML 1.0, e

- XML Schema.

Deve-se observar que a JSR-172 não suporta UDDI 2.0, que define como descobrir um serviço web. Esta também não permite que uma aplicação funcione como produtora de serviço (Endpoint), mas somente como consumidora de serviços.

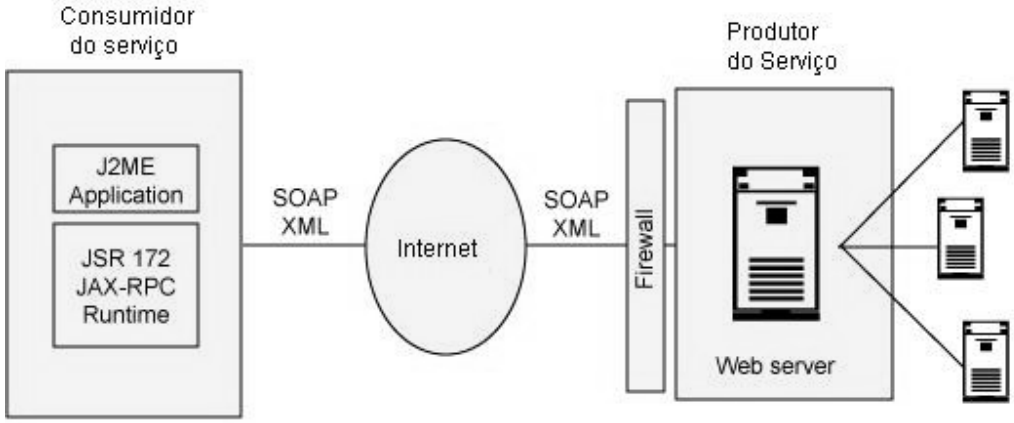

Figura 12 - Arquitetura de alto nível JSR 172 WSA. Fonte: [ORTIZ, 2005]

Segundo [ELLIS, 2004], a infra-estrutura desta especificação baseia-se em dois Pacotes Opcionais[COURTNEY, 2002] que oferecem as seguintes propostas:

- processamento básico de XML;

- reuso de conceitos de web service para utilização de serviços corporativos através de cliente J2ME;

- oferecer bibliotecas e convenções de código para programar estes dispositivos;

- aderir a padrões de web services e convenções que estão sendo consolidados pela comunidade de desenvolvedores Java;

- permitir a interoperabilidade de clientes J2ME com web services;

- permitir um modelo de comunicação do cliente J2ME com um web service consistente com outras tecnologias clientes Java, como o J2SE.

E para isso foram oferecidas as seguintes tecnologias:

- biblioteca de classes para manipulação de dados XML (parsing);

- biblioteca de classes para permitir a comunicação entre mecanismos RPC baseados em XML.

A manipulação dos documentos XML é oferecida através de um subconjunto da biblioteca JAXP 1.2 (JSR-063)[MORDANI, 2002] , e a comunicação através de um subconjunto da biblioteca JAX-RPC 1.1, que serão descritos no próximo tópico.

4.2.1. JAXP para J2ME

O objetivo deste pacote é definir um subconjunto de classes que permitem efetuar a manipulação de arquivos XML dentro do contexto de J2ME.

Este conjunto de classes foi especificado para utilizar pelo menos a configuração CLDC 1.0, a menor configuração da plataforma J2ME. Deve-se observar que esta biblioteca contém as seguintes características:

- não suporta interfaces de Simple API for XML Parsing (SAX) 1.0. Esta implementação foi substituída pela versão 2.0.

- não suporta Document Object Model (DOM) nível 1 e 2, pois o DOM é considerado muito pesado em termos de implementação e utilização de memória.

- não suporta XSLT (Extensible Stylesheet Transformation)[COUNSORTIUM, 1999]

- oferece um subconjunto de classes de SAX 2.0.

- oferece suporte a XML namespaces.

- oferece suporte a codificação UTF-8 e UTF-16.

Além destas características, JAXP também pode possibilitar a validação de documentos XML através de DTD, mas isto é restrito a dispositivos com grande capacidade de memória e processamento.

Existem três pacotes que compreendem a biblioteca JAXP para J2ME:

- javax.xml.parsers: composto por classes que obtém e referenciam implementações de parsers para uma plataforma.

- org.xml.sax: contém um subconjunto de classes e interfaces da API SAX 2.0.

- org.xml.sax.helpers: contém uma classe para aplicações entenderem a recepção de eventos de parser.

4.2.2. JAX-RPC para J2ME

A biblioteca JAX-RPC é a API Java responsável por interagir com web services através do protocolo SOAP. As funcionalidades oferecidas por esta biblioteca refletem as limitações da plataforma J2ME - tamanho de memória, processamento, além de limitações do ambiente de instalação (deploy) como largura de banda restrita e alta latência na rede.

Uma aplicação J2ME é baseada no modelo de programação cliente-servidor [ELLIS, 2004]. Para acessar um serviço disponibilizado pelo servidor, o cliente precisa gerar classes Stub que são interfaces de comunicação com o servidor.

Os Stubs são gerados em tempo de desenvolvimento através de uma ferramenta que efetua a leitura do arquivo WSDL e faz o mapeamento para classes Java.

O código gerado utiliza-se de SPI (J2ME Subset runtime Service Provider Interface) para efetuar chamadas RPC. Esta execução é utilizada pelos Stubs para se comunicar com um web services através de RPC, e é formada pelas classes Type, Element, ComplexType e Operation.

As classes Type, Element e ComplexType são usadas pelo Stub para descrever os parâmetros de entrada e o tipo de retorno de uma Remote Procedure Call (RPC). Para invocar um RPC, o stub pode efetuar os seguintes passos através do SPI:

- configurar as propriedades necessárias para configurar o RPC;
- criar um objeto que descreva os parâmetros de entrada;
- criar um objeto que descreva os parâmetros retornados;
- criar um objeto Operation representando uma invocação de RPC;

- codificar os parâmetros de entrada;

- invocar os serviços RPC do servidor;

- decodificar os valores retornados da chamada RPC.

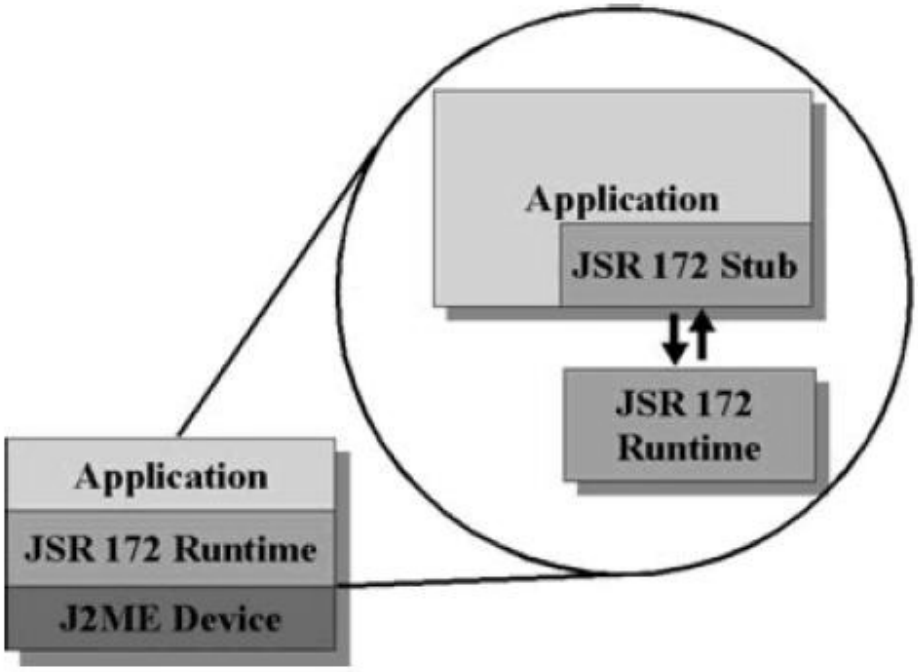

Figura 13 - Estrutura de classes da JAX-RPC Subset Runtime SPI. Fonte: [MUCHOW, 2004]

A figura 13 descreve a estrutura de classes do modelo SPI, e como elas se comunicam em uma chamada RPC.

Deve se verificar que a JSR172 Runtime esconde a complexidade do gerenciamento de conexão e codificação de dados, e o SPI desacopla os stubs de detalhes de implementação, permitindo maior portabilidade entre os stubs de diferentes fabricantes.

Além dos Stubs, a biblioteca JAX-RPC do J2ME possui suporte ao elemento soap:binding ou soap:operation de um documento WSDL, que é responsável por identificar o tipo do protocolo SOAP utilizado para localizar definições WSDL. Ou seja,

ele irá indicar quando uma operação é RPC ou orientada a documento. Nesta biblioteca de classes, as operações orientadas a documento são mapeadas para os métodos remotos correspondentes nas interfaces do servidor. Já a operação RPC, não é suportada.

A especificação JAX-RPC de J2ME[ELLIS, 2004] define o seguinte mapeamento entre tipos de um documento WSDL e Java, conforme tabela abaixo:

Tipo definido em XML	Tipo em Java
xsd:string	java.lang.String
xsd:int	int ou java.lang.Integer
xsd:long	long ou java.lang.Long
xsd:short	short ou java.lang.Short
xsd:boolean	boolean ou java.lang.Boolean
xsd:byte	byte ou java.lang.Byte
xsd:float	java.lang.String ou float ou java.lang.Float
xsd:double	java.lang.String ou double ou java.lang.Double
xsd:QName	javax.xml.namespace.QName
xsd:base64Binary	byte[]
xsd:hexBinary	byte[]

Os tipos xsd:float e xsd:double podem ser mapeados para o tipo java.lang.String, devido ao fato que a especificação de CLDC 1.0 não possui suporte aos tipos primitivos float e double. Já o mapeamento de um tipo para um objeto Java, no caso de xsd:string,

xsd:long, xsd:short, xsd:boolean, xsd:byte, xsd:float e xsd:double pode ocorrer caso o atributo nillable estiver true.

O tratamento das mensagens de falha é especificado através do elemento soap:fault no documento WSDL. Este elemento é opcional, e a biblioteca J2ME não oferece uma classe específica de exceção como a SOAPFaultException da biblioteca padrão de JAX-RPC. Ao invés disto, esta exceção pode ser mapeada para uma classe específica de exceção ou para uma java.rmi.RemoteException.

A especificação de JAX-RPC J2ME também possui suporte a todos os protocolos da camada de transporte do SOAP 1.1, como o HTTP. A segurança nesta camada vai depender da implementação utilizada na configuração e no perfil utilizado na aplicação.

No próximo capítulo serão abordados alguns serviços de segurança existentes e onde eles podem ser implantados.

5. SEGURANÇA, JAVA J2ME E WEB SERVICES

5.1. Segurança dos Web Services

A segurança na Internet é um assunto que deve tomar muita atenção dos desenvolvedores e das empresas visto que o mundo dos negócios está confiando cada vez mais na Internet como sendo o meio de transporte para suas comunicações e transações mais importantes, onde é necessário tomar cuidados importantes para evitar que o patrimônio das empresas seja apropriado indevidamente.

O W3C considerou os riscos relacionados à segurança como estando em três grupos:

– Problemas de configuração do servidor Web.

– Riscos do lado do navegador (browser).

– Interceptação de dados de rede enviados do navegador para o servidor, ou vice-versa.

Para minimizar estes riscos, existem diversos serviços de segurança que devem ser implementados para desempenhar este papel, conforme lista abaixo:

Serviço	**Descrição**
Identificação e autenticação	Um serviço de identificação e autenticação deveria saber responder, *quem é você, e como nós sabemos se a sua identidade é legítima.*
Autorização	Um serviço de autorização deve responder a pergunta: *você tem permissão para acessar esses dados ou*

	executar essa transação?
Integridade	Um serviço de integridade responde a pergunta: *os dados que você me enviou são os mesmos que eu recebi?*
Privacidade	Um serviço de privacidade responde a pergunta: *Nós podemos ter a certeza de que ninguém leu os dados que você me enviou?*
Não-repúdio	Um serviço de não-repúdio responde a pergunta: *Como provar para terceiros que você enviou os dados para eles?*

Tabela 1 - Descrição dos serviços de segurança

Para executar estes serviços de segurança, existem diversas técnicas e tecnologias essenciais, como a criptografia - a codificação da mensagem de maneira a evitar que terceiros vejam seu conteúdo -, que é uma das principais técnicas de segurança usadas na Internet, sendo, de fato, de importância fundamental para a PKI (Infra-estrutura de chaves públicas) e para o uso de certificados digitais.

Quando tratamos da segurança em relação a Web Services, devemos cuidar da segurança em diferentes camadas, desde a camada de rede de baixo nível até a de transporte, e, por fim, a camada de aplicação.

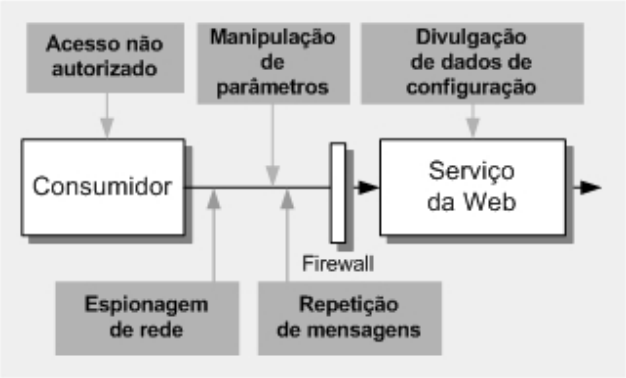

Figura 14 - Principais ameaças aos Web Services[MORDANI, 2002]

A figura acima mostra as principais ameaças e ataques aos Web Services. No capítulo seguinte será explicado as principais técnicas de segurança que podem ser empregadas para solucionar estes problemas.

5.1.1. – Segurança na camada de rede

Conforme mencionado, a segurança no contexto dos serviços Web pode ser tratada em diferentes níveis ou camadas. Na camada de rede, temos a IPSec (Internet Protocol Security) que tenta tornar o canal de rede seguro (por si só), o que é feito através da proteção à própria rede.

5.1.2. – Segurança na camada de transporte

A segurança na camada de transporte pode ser obtida utilizando vários mecanismos de segurança incorporados a partir dos próprios protocolos de transporte na Internet. O HTTP (Hypertext Transfer Protocol) e o SMTP (Simple Mail Transfer Protocol) são exemplos disso, sendo que cada um é acompanhado por seu próprio conjunto de medidas de segurança[HENDRICKS, 2002].

O esquema de autenticação do HTTP/1.0 baseia-se no modelo em que o agente de usuário, ou navegador, deve autenticar a si mesmo com um nome de usuário e uma senha, para cada domínio, ou área no servidor. Grande parte do processamento que usa um esquema de autenticação básico permanece oculto, pois a interação é manipulada de maneira transparente pelo HTTP.

Um outro método de autenticação, mais flexível, é através de formulários de HTML que ocorre na camada acima do HTTP. No caso do esquema básico do HTTP, os dados são codificados no formato base 64, um formato extremamente fácil de se decodificar. Já os formulários HTML, não são codificados, e são enviados através do http como texto puro. Uma forma de contornar esses problemas é combiná-los com o protocolo Secure Socket Layer (SSL).

O SSL segundo [HENDRICKS, 2002] fornece serviços de identificação/ autenticação, integridade e de segurança privada. Este possui dois sub-protocolos: o protocolo de registro SSL, que define o formato usado para transmitir os dados, e o protocolo de estabelecimento de handshake SSL, que utiliza o protocolo de registro SSL para trocar mensagens entre o servidor que permite o SSL, quando estabelecida à conexão pela primeira vez.

5.1.3. – Segurança na camada de aplicação / SOAP

Ao falar em segurança no SOAP, uma das maneiras mais diretas de se chegar a ela consiste em examinar os canais de transmissão usados para a troca de mensagens SOAP. O SOAP é transmitido através do http, e as duas maneiras mais populares de proteger o http é através do uso de SSL ou de VPN (Virtual Private Networks).

Com a utilização de VPN, configura-se um túnel criptografado entre duas máquinas ou redes que permitem a transmissão de dados segura entre as duas extremidades, de modo que para o usuário final, parece que os dados são transmitidos somente através de sua rede interna, porém, os dados trafegam através de redes públicas.

As soluções de VPN se baseiam, geralmente em IPSec (Internet Protocol Security), um framework de padrões abertos desenvolvidos pela IETF para garantir a privacidade e autenticação dos dados, além de autenticação do usuário[HENDRICKS, 2002]. Uma das principais vantagens de IPSec é que ele opera na camada de rede e permite que as aplicações operem de maneira transparente nas redes.

As VPNs utilizam criptografia DES de 56 bits ou 3DES de 168 bits para a privacidade dos dados e MD5 e SHA-1 para autenticação dos dados.

A segurança em nível de aplicação requer mais trabalho do que a segurança de canal. Deve-se modificar o código da aplicação para fornecer segurança tanto da parte do remetente quanto da parte do receptor. E para oferecer um maior grau de proteção ela pode ser utilizada em conjunto com SSL ou VPN.

Tipicamente, consegue-se segurança em aplicações através da utilização da criptografia dos dados que são enviados e recebidos, onde o remetente e o receptor devem compartilhar o conhecimento sobre como a criptografia é feita, resolvendo os problemas referentes à confiabilidade. Além de que o código deve ser executado de acordo com o serviço ou nome de usuário apropriado, resolvendo alguns dos problemas referentes à autorização.

Ao considerar a segurança, deve-se, observar as cinco principais áreas abrangidas no padrão SOAP:

- o formato da mensagem;

- a codificação;

- convenções de RPC;

- camada de transporte;

- natureza das mensagens com anexo;

Pois com a segurança da camada de aplicação/ SOAP, precisa-se modificar as mensagens SOAP propriamente ditas, ou seja, as extremidades da aplicação de uma mensagem SOAP precisam estar aptas a discernir o tipo de segurança usada, além de como e quando aplicá-la.

Conforme [HENDRICKS, 2002], SOAP oferece um módulo de extensão de segurança através de assinatura digital, onde os recursos de segurança podem ser incluídos na mensagem SOAP. Esta extensão especifica as regras de sintaxe e de processamento de uma entrada de cabeçalho SOAP para transportar informações de assinatura digital.

Abaixo, segue exemplo de um envelope SOAP com a extensão de assinatura digital:

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header>
    <SOAP-SEC:Signature
      xmlns:SOAP-SEC="http://schemas.xmlsoap.org/soap/security/2000-12"
      SOAP-ENV:actor="some-URI"
      SOAP-ENV:mustUnderstand="1">
      <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        <ds:SignedInfo>
          <ds:CanonicalizationMethod
            Algorithm="http://www.w3.org/TR/2000/CR-xml-c14n-20001026">
          </ds:CanonicalizationMethod>
          <ds:SignatureMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
          <ds:Reference URI="#Body">
            <ds:Transforms>
              <ds:Transform Algorithm="http://www.w3.org/TR/2000/CR-xml-c14n-
20001026"/>
```

```
            </ds:Transforms>
            <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
            <ds:DigestValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</ds:DigestValue>
          </ds:Reference>
        </ds:SignedInfo>
        <ds:SignatureValue>MC0CFFrVLtRlk=...</ds:SignatureValue>
      </ds:Signature>
    </SOAP-SEC:Signature>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body
    xmlns:SOAP-SEC="http://schemas.xmlsoap.org/soap/security/2000-12"
    SOAP-SEC:id="Body">
    <m:GetLastTradePrice xmlns:m="some-URI">
      <m:symbol>IBM</m:symbol>
    </m:GetLastTradePrice>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Pode se observar no código uma entrada de cabeçalho da assinatura, onde se assina o corpo SOAP, e a assinatura resultante, <ds:Signature> é adicionada à entrada do cabeçalho <SOAP-SEC:Signature>. O elemento <ds:SignedInfo> é a informação que está sendo assinada. As assinaturas aplicadas a um documento não protegem nenhuma informação além daquelas assinadas.

No capítulo seguinte será demonstrada uma outra tecnologia de segurança, a assinatura XML.

5.1.4. – Assinatura XML

A assinatura XML é uma iniciativa conjunta da IETF e do W3C para especificar uma sintaxe XML e regras de processamento para criação e representação digital de assinaturas. As vantagens na utilização da assinatura XML, diferente de outros padrões de assinaturas digitais, é que é baseado na independência da linguagem de programação, fácil interpretação humana e independência de fabricante. Esta

tecnologia também permite assinar digitalmente subconjuntos de um documento XML, o que permite um número maior de benefícios.

5.1.5. – Criptografia XML

A criptografia XML especifica um processo para criptografar dados e sua representação em formato XML. Os dados podem ser:

- dados arbitrários (incluindo um documento XML),
- elementos XML,
- ou conteúdos de elementos XML.

Um documento XML que utiliza a criptografia XML pode ser visto por qualquer utilizador, mas apenas o proprietário da chave de criptografia conseguirá compreender o conteúdo que foi criptografado.

5.1.6. - XKMS

O XKMS (XML Key Management Specification) fornece um protocolo baseado em XML/SOAP para distribuição e registro de chaves públicas. Ele inclui funções para informações sobre chaves, registro, verificação e revogação. O XKMS é destinado a fornecer suporte para gerenciamento de chaves para tecnologias como assinatura XML e criptografia XML, mas também pode ser utilizado para suportar outras tecnologias de chaves públicas.

O XKMS inclui, essencialmente três serviços e especificações: Key Information Service Specification (X-KISS), Key Registration Service Specification (X-KRSS), trust Assertion Service Specification (X-TASS).

A X-KISS fornece um serviço de resolução/ recuperação e verificação de chave pública. Serviços de baixo nível de X-KISS permitem que aplicações recuperem o certificado de um usuário. Já um serviço de alto nível fornece uma interface para um intervalo de informação vinculada ao dono de uma chave pública identificada.

O X-KRSS define um protocolo para o registro das informações de uma chave pública.

A especificação X-TASS define uma arquitetura e um protocolo para as intruções de recuperação, conhecidas como trust assertions, que são ligadas à chave pública.

5.2. Segurança em JAVA 2 Micro Edition

A segurança em dispositivos móveis apresenta um grande interesse e desafio. Os dispositivos móveis estão permitindo aos usuários acessar cada vez mais serviços seguros que utilizam a Internet. E muitos são os interessados pela segurança em dispositivos móveis. As operadoras de rede tem um grande interesse em medir a integridade dos dados que estão sendo enviados pela sua rede para poder oferecer cada vez mais lucrativos serviços aos seus clientes. Os fabricantes destes dispositivos gostariam de oferecer aos usuários as mais diversas experiências com seus aparelhos. Os desenvolvedores gostariam de oferecer aplicações diversificadas no mercado, desde jogos a aplicações empresariais. E os usuários gostariam que seus dispositivos fossem simples, divertidos, convenientes e fáceis de usar.

Alguns modelos de segurança foram implantados em J2ME. A especificação MIDP 2.0 definiu um modelo de segurança baseado na política de autenticação de um MIDlet e na escolha do usuário. O Java Community Process (JCP) definiu um pacote opcional

com diversos recursos de segurança, o Security and Trust Service for J2ME, que será explicado no próximo capítulo.

5.2.1. – Security and Trust Service for J2ME (SATSA) – JSR 177

A Security and Trust Service API (SATSA) [AHMAD, 2004] é um pacote opcional J2ME que oferece segurança e serviços confiáveis através da integração de um elemento seguro (Security Element - SE). Um SE, é um componente em um dispositivo J2ME que possui os seguintes benefícios:

- armazenamento seguro para proteger dados restritos, como chave pública de um usuário, certificados de chave pública, credenciais, informações pessoais e assim por diante.

- operações de criptografia com suporte a protocolo de pagamentos, integridade de dados e dados confidenciais.

- um ambiente seguro de execução para instalação de mecanismos de segurança. As aplicações podem utilizar dessa área para manipular diversos serviços como autenticação de usuário, identificação, pagamento bancário, aplicações financeiras, etc.

O SE pode ser encontrado em diversas formas. As mais comuns são os smart cards que são encontrados em telefones celulares. Temos como exemplo os cartões SIM que podem ser encontrados nos telefones GSM, os cartões UICC em telefones 3G, e os cartões RUIM em telefones CDMA. As operadoras da rede GSM utilizam a autenticação dos dados nos cartões GSM, assim como a agenda telefônica.

Além dos smart cards, um SE pode ser implementado pelo próprio dispositivo através de algum mecanismo de hardware ou software.

A API SATSA oferece as seguintes funcionalidades:

- Comunicação com um smart card: um smart card oferece um ambiente seguro programável. Eles são os serviços SE instalados que mais oferecem segurança e serviços confiáveis. Estes serviços podem ser constantemente atualizados com novas aplicações que podem ser instaladas no smart card. Existem dois métodos de acesso definidos para comunicação com o smart card, o protocolo APDU e o protocolo Java Card RMI, que serão descritos mais abaixo.

- Serviço de assinatura digital e gerenciamento básico de credenciamento de usuário: O serviço de assinatura digital permite a uma aplicação J2ME gerar assinaturas digitais conforme o formato do sistema criptográfico de mensagem (CMS – Cryptographic Message System). As assinaturas digitais são usadas para autenticar usuários ou autorizar transações utilizando criptografia com chaves públicas.

- Biblioteca de criptografia de propósito geral: A biblioteca de criptografia oferece é um sub-conjunto da API de criptografia de J2SE. Ela suporta operações básicas de criptografia como message-digests, verificação de assinatura digital (não permite a assinatura), criptografia e decriptografia.

A especificação de SATSA define quatro diferentes APIs[AHMAD, 2004]:

- SATSA-APDU: permite aplicações a comunicarem com aplicações smart-card através de um protocolo de baixo-nível.

- SATSA-JCRMI: método alternativo ao SATSA-APDU para comunicar com aplicações smart-card através de um protocolo de objetos remoto.

- SATSA-PKI: permite aplicações a utilizarem smart-card para assinar digitalmente um dado e gerenciar certificados de usuário.

- SATSA-CRYPTO: API de criptografia de propósito geral que possui a capacidade de message digests, verificação de assinatura digital e cifragem dos dados. Esta API é a única que não envolve a interação com smart-card.

6. DESENVOLVIMENTO DE UMA APLICAÇÃO QUE UTILIZA JAVA J2ME E WEB SERVICES.

6.1. Descrição da aplicação

Para comprovar o estudo realizado, será desenvolvida uma aplicação que utilize os conceitos que foram discutidos durante o trabalho. Esta aplicação, basicamente, consiste em um gerenciador de finanças pessoais, estruturado na forma de um web service disponível em uma máquina servidora, e acessado por um cliente J2ME. Alternativamente, este serviço poderia ser acessado por qualquer tipo de aplicação web ou desktop.

Neste sistema, o usuário poderá manipular suas finanças através do cadastro das atividades realizadas durante um período. Esta atividade pode ser caracterizada pelo recebimento do salário, o pagamento de uma conta, uma compra, ou qualquer atividade que o usuário do sistema tenha interesse em cadastrar.

O sistema utilizará um serviço web para solicitar informações do usuário, ou atualizar novas informações cadastradas no cliente ou no servidor. Estas informações são referentes às atividades realizadas pelo usuário e descritas acima.

As seguintes funcionalidades são previstas pelo sistema:

- cadastro de usuário;
- cadastro de contas;
- cadastro de itens de conta;

- cálculo de contas;

- sincronização com o serviço;

- configuração do sistema;

O cadastro do usuário consiste em armazenar em um mecanismo de persistência local e/ ou remoto as informações pessoais do usuário, como o seu nome, telefone, e-mail, CPF, usuário e senha. Estas informações devem ser cadastradas pelo usuário através de uma aplicação cliente. No entanto, nos dispositivos móveis, podem ser solicitadas somente algumas dessas informações.

O cadastro de conta consiste em armazenar em um mecanismo de persistência local e/ ou remoto as informações das atividades que o usuário pode realizar em um determinado período. Esta é caracterizada pelo seu nome, o seu tipo, que pode ser receita ou despesa, a freqüência da conta (diária, mensal, anual, único evento), e a data de expiração da conta.

O cadastro de itens de contas consiste no armazenamento de uma atividade realizada em uma conta. Devem ser armazenados o nome do evento (do item de conta), a data de acontecimento do evento, o valor real do evento, e o valor previsto pelo evento. Este evento pode ser exemplificado como um cadastro de salário. Onde o item de conta “valor do salário” é cadastro para a conta, pré-cadastrada “salário”. Armazena-se em valor previsto o valor X, e em valor real, o valor que realmente foi pago.

A funcionalidade cálculo de contas deve realizar a totalização de contas cadastradas em um determinado período. Ela deve informar ao usuário o saldo real e previsto entre as contas de receita e despesa. Este saldo é calculado através do somatório dos itens de conta das contas de receita e a despesa.

A sincronização com o serviço (web service) é responsável por solicitar ao serviço se existem novas contas cadastradas no servidor e que não estão cadastradas no cliente. Caso existam novas contas no servidor, é possível solicitar os dados das contas para serem cadastrados no cliente. O cliente também pode atualizar os dados do servidor, como o cadastro do usuário, cadastro de contas ou itens de contas. Esta troca de informação deve acontecer através de uma tecnologia padrão, interoperável, e que permita a comunicação entre dispositivos de diferentes tecnologias, sistemas operacionais e linguagem de programação de forma segura e confiável.

A funcionalidade de configuração do sistema é responsável por armazenar em um repositório local da aplicação cliente, informações sobre o servidor, ou personalização de interfaces, para facilitar a navegação do usuário através da interface do sistema.

Nas próximas seções, serão apresentadas a especificação do sistema desenvolvido, e conclusões finais a respeito das tecnologias envolvidas nesse trabalho.

6.2. Especificação da aplicação

6.2.1. – Escopo da aplicação

Devido à restrição de tempo para o desenvolvimento do trabalho, e objetivo do trabalho, que se foca no estudo da tecnologia J2ME, será desenvolvida somente a aplicação cliente para dispositivos móveis utilizando a tecnologia Java 2 Micro Edition.

No âmbito da aplicação servidor, ou seja, o serviço web, será implementada somente a interface de comunicação com o cliente. Os serviços de persistência de dados do servidor, e troca de informações não serão implementados.

Na aplicação cliente, as seguintes funcionalidades serão desenvolvidas com o seguinte escopo:

- Cadastro de usuário: Não será implementado.

- Cadastro de conta: Serão implementadas as funcionalidades de cadastro, edição e exclusão de contas.

- Cadastro de itens de conta: Serão implementadas as funcionalidades de cadastro, edição, exclusão e visualização de itens de conta.

- Cálculo de contas: Será implementada esta funcionalidade, porém, para efetuar o cálculo das contas será necessário selecionar quais as contas serão contabilizadas.

- Sincronização com o serviço: será implementada com as funcionalidades definidas neste escopo – atualização de conta e item conta.

Além destas funcionalidades, a camada de segurança do sistema também não será implementada.

6.2.2. – Arquitetura do sistema

A arquitetura do sistema baseia-se na arquitetura padrão de comunicação entre um cliente J2ME e um Web service.

Para o desenvolvimento da aplicação cliente foram utilizadas as seguintes tecnologias:

- J2ME Wireless Toolkit 2.2;

- Configuração: CLDC 1.1;

- Perfil: MIDP 2.0;

- Pacotes opcionais: JSR 172 (WSA);

Para o desenvolvimento da aplicação servidora foram utilizados:

- JBoss Application Server 4.0.2;

- Engine Web service: Axis 1.2 RC3;

- J2EE 1.4;

- J2SE 1.4.2_06

- Xdoclet 1.2.2;

- Maven 1.0.2;

- ANT

6.3. – Desenvolvimento da aplicação

Após a especificação dos requisitos e das tecnologias que foram utilizadas no trabalho, será mostrado neste capítulo, como ocorreu o desenvolvimento da aplicação.

A aplicação foi batizada de “Economize” devido as suas características de manipulação de finanças pessoais.

Para a construção desta aplicação foram definidas as seguintes etapas:

- definição da interface de comunicação do servidor.

- criação do arquivo WSDL baseado na interface de comunicação do servidor.

- construção da aplicação cliente.

Estas etapas serão descritas nas próximas seções.

6.3.1. – Definição da interface de comunicação do servidor

A interface de comunicação com o servidor foi definida baseada nas operações que um servidor poderiam efetuar. E para definir estas operações, inicialmente foi

modelado um conjunto de classes que representassem os objetos do domínio do problema, conforme diagrama abaixo:

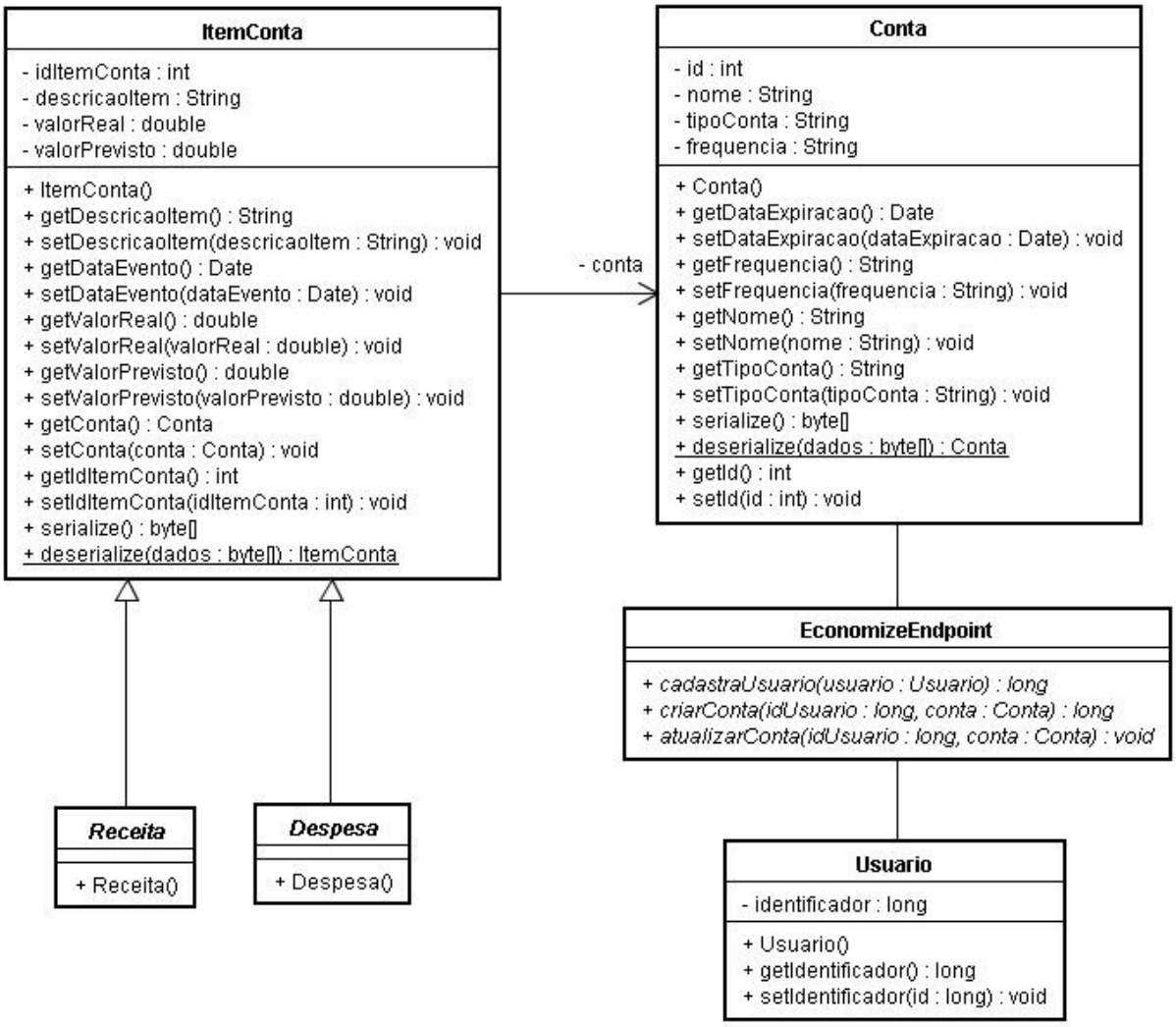

Figura 15 – Diagrama de classes do servidor

A classe EconomizeEndpoint representa as operações oferecidas através do web service. Esta classe será mapeada para o documento WSDL, que será descrito no próximo tópico.

6.3.2. – Criação do arquivo WSDL

Para a construção do documento WSDL foi utilizado uma ferramenta oferecida através do kit de farramentas do Axis[FOUNDATION, 2005].

O Axis é um engine SOAP, ou seja, um framework para a construção de processos SOAP tanto em cliente, servidores e gateways. Ele consiste de diversos sub-sistemas que trabalham em conjunto.

Ele possui uma ferramenta chamada Java2WSDL, responsável por gerar um arquivo WSDL a partir de uma classe Java. Neste trabalho, utilizou-se da ferramenta Ant[COUNSORTIUM, 2005], que é responsável por executar um conjunto de tarefas especificadas em um arquivo xml.

Abaixo, segue parte do arquivo build.xml, utilizado para a criação do WSDL:

```
    <!—Definição do classpath do AXIS-->
    <path id="axis.classpath">
      <pathelement location="${build.dir}"/>
   <fileset dir="F:\Web Service\axis\axis-1_2RC3/lib">
        <include name="*.jar"/>
      </fileset>
   <fileset dir="${jboss.server}/lib/">
        <include name="javax.servlet*.jar"/>
      </fileset>
    </path>

  <!—Definição da criação do arquivo WSDL ->
    <target name="wsdl">
      <mkdir dir="${basedir}/descriptor/ws/wsdl" />
      <java classname="org.jboss.axis.wsdl.Java2WSDL"
      classpathref="build.classpath" fork="yes">
```

```
            <arg value="-lhttp://127.0.0.1:8080/economize/Economize" />
            <arg value="-SEconomizeService" />
            <arg value="-sEconomizeEndpoint" />
            <arg value="-odescriptor/ws/wsdl/economize.wsdl" />
            <arg value="-uLITERAL" />
            <arg value="edu.ufsc.economize.servidor.ws.EconomizeEndpoint" />
          </java>
        </target>
```

Neste trecho de código, observa-se que foi especificada a localização da instalação do Axis, além dos parâmetros utilizados para criar o arquivo WSDL:

- "–l": refere-se a URI que identifica o serviço.

- "–S": refere-se ao nome do serviço.

- "-s": refere-se ao port-type, ou seja, o nome utilizado para definir um serviço do web service.

- "-o": define o caminho onde será criado o arquivo wsdl.

- "-u": refere-se ao tipo de serialização.

- "edu.ufsc.economize.servidor.ws.EconomizeEndpont": Classe que servirá de base para a criação do arquivo wsdl. A partir dela será definido as operações, e os elementos.

Na tabela abaixo, encontra-se o arquivo WSDL gerado:

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://ws.servidor.economize.ufsc.edu"
                          xmlns="http://schemas.xmlsoap.org/wsdl/"
                          xmlns:apachesoap="http://xml.apache.org/xml-soap"
                          xmlns:impl="http://ws.servidor.economize.ufsc.edu"
                          xmlns:intf="http://ws.servidor.economize.ufsc.edu"

xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
                          xmlns:tns2="http://util.servidor.economize.ufsc.edu"
                          xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"

xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
                          xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

```
 <wsdl:types>
  <schema targetNamespace="http://util.servidor.economize.ufsc.edu"
xmlns="http://www.w3.org/2001/XMLSchema">
   <complexType name="Conta">
    <sequence>
     <element name="dataExpiracao" nillable="true" type="xsd:dateTime"/>
     <element name="frequencia" nillable="true" type="xsd:string"/>
     <element name="id" type="xsd:int"/>
     <element name="nome" nillable="true" type="xsd:string"/>
     <element name="tipoConta" nillable="true" type="xsd:string"/>
    </sequence>
   </complexType>
   <complexType name="Usuario">
    <sequence/>
   </complexType>
  </schema>
 </wsdl:types>

   <wsdl:message name="atualizarContaRequest">
      <wsdl:part name="idUsuario" type="xsd:long"/>
      <wsdl:part name="conta" type="tns2:Conta"/>
   </wsdl:message>

   <wsdl:message name="atualizarContaResponse">
   </wsdl:message>

   <wsdl:message name="criarContaRequest">
      <wsdl:part name="idUsuario" type="xsd:long"/>
      <wsdl:part name="conta" type="tns2:Conta"/>
   </wsdl:message>

   <wsdl:message name="criarContaResponse">
      <wsdl:part name="criarContaResponse" type="xsd:long"/>
   </wsdl:message>

   <wsdl:message name="cadastraUsuarioRequest">
      <wsdl:part name="usuario" type="tns2:Usuario"/>
   </wsdl:message>

   <wsdl:message name="cadastraUsuarioResponse">
      <wsdl:part name="cadastraUsuarioResponse" type="xsd:long"/>
   </wsdl:message>

   <wsdl:portType name="EconomizeEndpoint">

      <wsdl:operation name="cadastraUsuario" parameterOrder="usuario">
         <wsdl:input message="impl:cadastraUsuarioRequest"
name="cadastraUsuarioRequest"/>
         <wsdl:output message="impl:cadastraUsuarioResponse"
name="cadastraUsuarioResponse"/>
      </wsdl:operation>

      <wsdl:operation name="criarConta" parameterOrder="idUsuario conta">
         <wsdl:input message="impl:criarContaRequest"
name="criarContaRequest"/>
         <wsdl:output message="impl:criarContaResponse"
name="criarContaResponse"/>
```

```
      </wsdl:operation>

      <wsdl:operation name="atualizarConta" parameterOrder="idUsuario conta">
         <wsdl:input message="impl:atualizarContaRequest"
name="atualizarContaRequest"/>
         <wsdl:output message="impl:atualizarContaResponse"
name="atualizarContaResponse"/>
      </wsdl:operation>
   </wsdl:portType>

   <wsdl:binding name="EconomizeEndpointSoapBinding"
type="impl:EconomizeEndpoint">
      <wsdlsoap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>

      <wsdl:operation name="cadastraUsuario">
         <wsdlsoap:operation soapAction=""/>
         <wsdl:input name="cadastraUsuarioRequest">
            <wsdlsoap:body use="literal"/>
         </wsdl:input>
         <wsdl:output name="cadastraUsuarioResponse">
            <wsdlsoap:body use="literal"/>
         </wsdl:output>
      </wsdl:operation>

      <wsdl:operation name="criarConta">
         <wsdlsoap:operation soapAction=""/>
         <wsdl:input name="criarContaRequest">
            <wsdlsoap:body use="literal"/>
         </wsdl:input>
         <wsdl:output name="criarContaResponse">
            <wsdlsoap:body use="literal"/>
         </wsdl:output>
      </wsdl:operation>

      <wsdl:operation name="atualizarConta">
         <wsdlsoap:operation soapAction=""/>
         <wsdl:input name="atualizarContaRequest">
            <wsdlsoap:body use="literal"/>
         </wsdl:input>
         <wsdl:output name="atualizarContaResponse">
            <wsdlsoap:body use="literal"/>
         </wsdl:output>
      </wsdl:operation>
   </wsdl:binding>

   <wsdl:service name="EconomizeService">
      <wsdl:port binding="impl:EconomizeEndpointSoapBinding"
name="EconomizeEndpoint">
         <wsdlsoap:address
location="http://127.0.0.1:8080/economize/Economize"/>
      </wsdl:port>
   </wsdl:service>

</wsdl:definitions>
```

A partir do momento que o arquivo WSDL está definido, é possível iniciar o desenvolvimento da aplicação cliente. Onde, através da ferramenta WSDL2Java é possível gerar as classes Stubs que servirão de interface de comunicação entre o cliente e o serviço baseando-se no arquivo WSDL.

No próximo capítulo, será descrito como foi realizada a construção da aplicação cliente.

6.3.3. – Construção da aplicação cliente

A aplicação cliente foi desenvolvida utilizando o Java Wireless Toolkit versão 2.2. Esta ferramenta contém um emulador para dispositivos móveis, que pode ser configurado de acordo com o requisito da aplicação. Neste caso, ele foi configurado para simular um dispositivo celular com pouca memória, pouco poder de processamento, e acesso à rede.

Para a criação do cliente web service dentro do ambiente J2ME é necessário adicionar os pacotes opcionais ao classpath da aplicação ao iniciar o emulador. Feito isso, é necessário criar as classes Stub que irão representar os serviços oferecidos pelo web service.

A criação destas classes será feita a partir da ferramenta WSCompile do pacote opcional JSR 172. Abaixo, encontra-se o comando utilizado para a criação das classes Stub:

```
C:\Documents and Settings\Lucas\Meus
documentos\workspace\EconomizeMobile\src>f:
\WTK22\bin\wscompile.bat -keep -gen:client -f:wsi -verbose config.xml
```

Para a execução deste comando, é necessário criar um arquivo xml chamado “config.xml” que identifica o caminho do arquivo WSDL e o pacote onde serão criadas as classes Stub:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration xmlns="http://java.sun.com/xml/ns/jax-rpc/ri/config">
      <wsdl name="EconomizeService" location="economize.wsdl"
packageName="edu.ufsc.economize.wsdl"/>
</configuration>
```

Com os Stubs criados, foi possível iniciar o desenvolvimento das funcionalidades executadas pelo cliente, e que já foram especificadas no capítulo 6.2.

A seguinte estrutura de classes de negócio foi definida para o cliente:

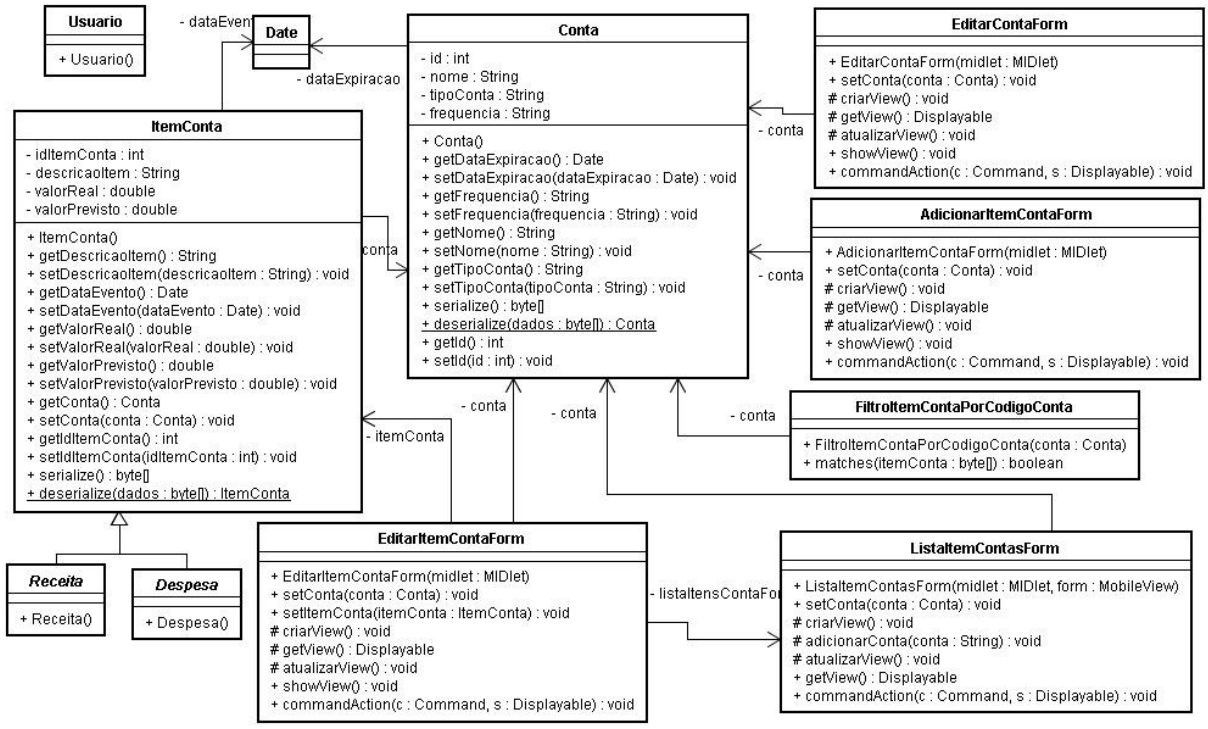

Figura 16 – Diagrama de classes de negócio do cliente

A classe Conta contém os dados referente a uma conta, ou seja, o seu identificador, o nome da conta, o tipo da conta, a data de expiração da conta, e a freqüência de repetição da conta, ou seja, qual a periodicidade na qual esta conta será

contabilizada (diariamente, mensalmente, anual). Esta classe também é responsável por serializar e deserializar seus dados transformando o objeto em um array de bytes, que será utilizado posteriormente para persistir os dados no dispositivo através da biblioteca RMS do J2ME.

A classe ItemConta possui os seguintes atributos: o identificador do item de conta, a descrição do item, o valor previsto para aquele item, e o valor real daquele item. Esta classe também possui o atributo conta, que se refere à conta a qual este item pertence.

Esta classe também possui os métodos de serialização e deserialização do objeto para um array de bytes.

Pode-se observar que ambas as classes não possuem métodos complexos, apenas de inicialização dos atributos e serialização dos dados. Estas classes se comunicam com as classes Form, que são responsáveis por definir as operações que um usuário pode realizar, como inserir, editar, listar e apagar uma conta e um item de conta. Todas as classes Form, por exemplo a classe EditarContaForm, implementam a interface MobileView e através do método showView, solicitam ao MIDlet para exibir o seu conteúdo para a interface do dispositivo, conforme o diagrama de classes abaixo.

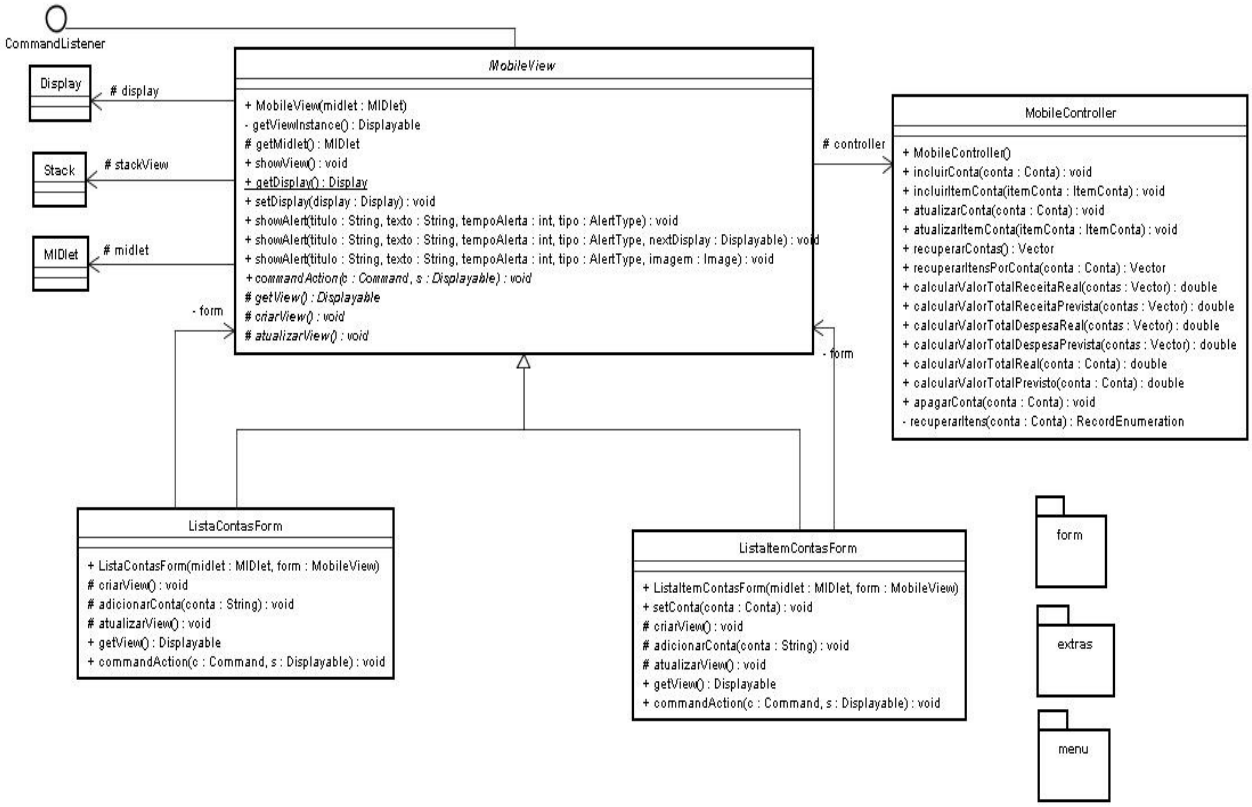

Figura 17 – Diagrama de classes do pacote view

A classe MobileController é responsável por manipular os dados que estão sendo passados pelo formulário e por delegar as operações de persistência e comunicação com a rede.

A camada de persistência é manipulada através da classe GerenciadorRMS, que utiliza a classe RecordStore, pertencente à API do J2ME, para manipular os dados no registro do dispositivo.

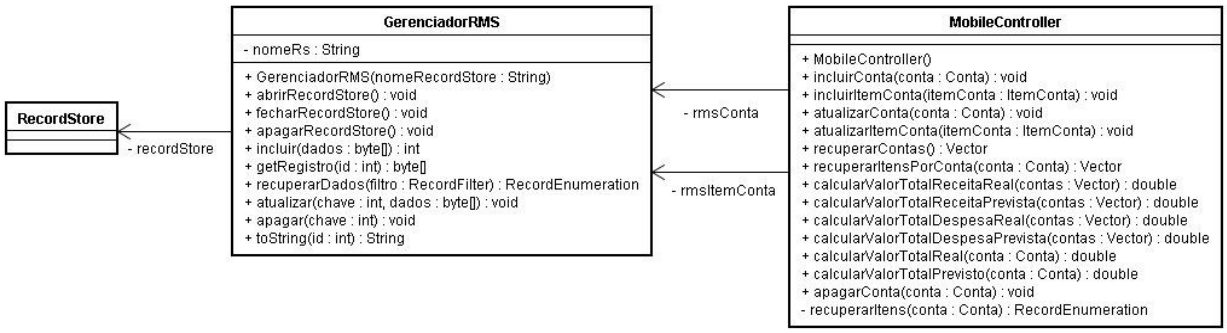

Figura 18 – Diagrama de classes da camada de persistência

Para manipular os dados através da rede, utilizou-se as classes Stubs geradas através do arquivo WSDL do sistema para tratar os dados e enviá-los através do protocolo SOAP sobre HTTP.

Nas figuras abaixo, segue o resultado da implementação da aplicação cliente:

Figura 19 – Tela de inicialização da aplicação e menu principal

Nas telas acima, é possível inicializar o MIDlet e acessar o menu principal. Através deste menu é possível acessar as opções de CONTA, ITEM DE CONTA, CALCULAR..., CONFIGURAÇÕES e SOBRE....

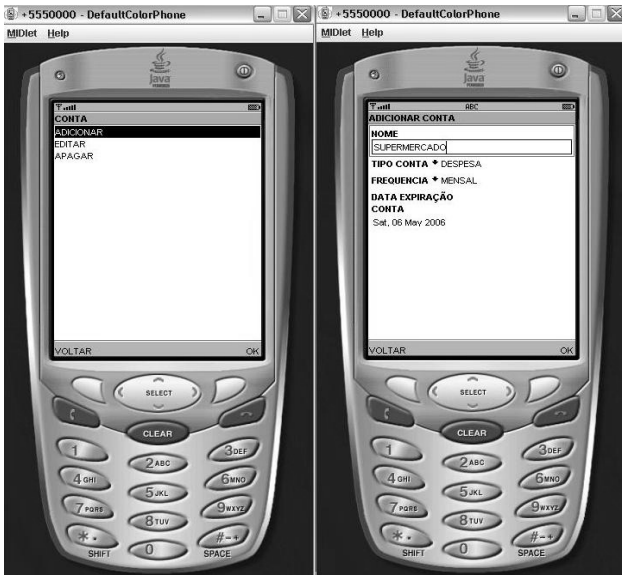

Figura 20 – Menu conta e adição de uma nova conta

Conforme a figura 20 demonstra, é possível através da opção CONTA, adicionar uma nova conta, editar uma conta existente ou apagar uma conta. Ao adicionar uma conta é necessário informar todos os dados solicitados no formulário.

Figura 21 – Adicionar item de conta

Para adicionar um Item de conta é utilizada a mesma rotina usada para adicionar uma conta, sendo diferente apenas o formulário onde serão preenchidos os dados. Primeiramente é necessário selecionar a conta à qual este item pertence, e depois preencher todos os dados.

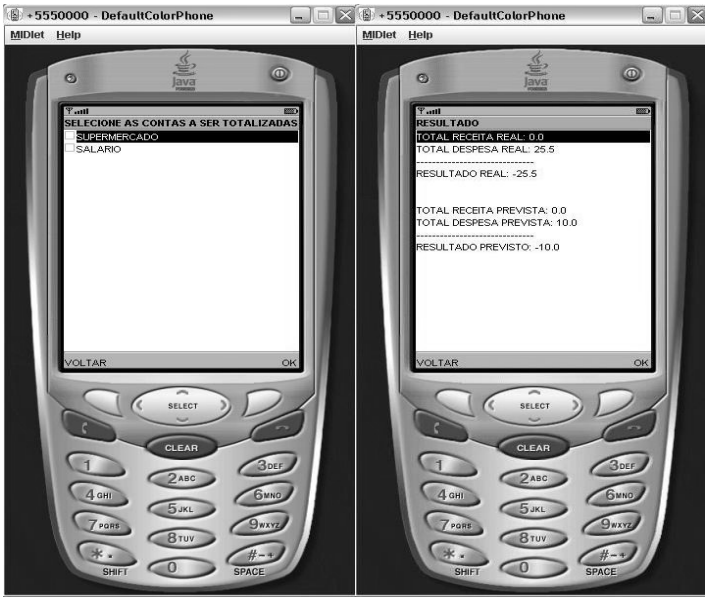

Figura 22 – Totalizar contas

Na figura 22 é demonstrada a funcionalidade de totalização de contas, onde serão selecionadas em uma tela quais as contas que devem ser totalizadas e posteriormente será demonstrado em um formulário o valor total contabilizado entre as contas selecionadas.

6.3.4. – Conclusão sobre o desenvolvimento

O desenvolvimento da aplicação Economize, proposto neste trabalho, auxiliou na aprendizagem das tecnologias propostas para estudo. Através dela pode-se verificar a dificuldade em desenvolver uma aplicação para dispositivos com capacidade de memória e processamento restritos, onde é necessário otimizar o acesso a recursos de rede, e a criação de objetos armazenados em memória.

Foi observado também, que o desenvolvimento da camada de comunicação com o Web Service utilizando a configuração CLDC de J2ME é uma operação muito complexa. A definição do arquivo WSDL é uma etapa muito importante no desenvolvimento do projeto, pois através dele serão geradas as Stubs que irão se comunicar com os objetos de negócio da aplicação. E deve ser lembrado que o arquivo WSDL utilizado em J2ME deve seguir o padrão WS-I, onde devem ser respeitadas algumas regras e restrições quanto à utilização de elementos no arquivo XML para manter o padrão de interoperabilidade.

Porém, mesmo com a dificuldade observada no desenvolvimento de algumas tarefas, a aplicação atingiu os seus objetivos. Através dela foi possível comprovar a possibilidade de desenvolver uma aplicação cliente web service em um dispositivo que se enquadra na configuração CLDC de J2ME. Observa-se que a arquitetura J2ME

oferece uma rica biblioteca de classes e ferramentas que auxiliam no desenvolvimento desta categoria de aplicação.

7. CONCLUSÕES E PERSPECTIVAS FUTURAS

A arquitetura de Web Services está se difundindo cada vez mais. A possibilidade de criar serviços que podem ser distribuídos pela Internet sem a necessidade de conhecer a sua implementação é o principal aspecto positivo desta arquitetura. Os consumidores destes serviços necessitam conhecer somente a interface do serviço. Assim, os web services podem ser descritos como componentes, que são acessados através de protocolos Web amplamente difundidos, como o HTTP, e que utilizam formatos universais para a representação de dados (XML). A arquitetura de web services não define um conjunto de tecnologias, mas padrões para transporte, descrição e descoberta de serviços, que são implementados utilizando SOAP, WSDL e UDDI, respectivamente.

A arquitetura J2ME fez o Java retornar aos seus princípios, ou seja, ser utilizado em dispositivos portáteis. Esta característica fez o Java crescer muito no mercado abrindo novas portas para esta tecnologia.

Neste trabalho demonstramos como é possível desenvolver clientes de web services para dispositivos móveis com suporte para J2ME. Isto foi conseguido através do desenvolvimento de uma aplicação cliente J2ME que acessa serviços oferecidos por um Web Service. Esta aplicação foi implementada utilizando a configuração CLDC de J2ME, e o perfil MIDP 2.0, que atendem aplicações dos dispositivos com poucos recursos de processamento, memória, e conexão intermitente de rede, e abrangem os dispositivos mais populares hoje em dia, como celulares e PDAs.

Para que esta aplicação se comunicasse com um web service, foi necessário utilizar o pacote opcional de J2ME para web services, o WSA. Este pacote oferece

suporte a arquitetura básica de web services (WS-I Basic Profile 1.0). Então, para a construção do web service foi necessário restringir-se a esta especificação, ou seja, chegamos à conclusão que uma aplicação J2ME não é compatível com todos os web services, o que não pode ser considerado uma falha desta especificação, mas sim uma condição existente graças à limitação de recursos de hardware dos dispositivos.

Este pacote opcional deve ser instalado no kit de ferramentas de desenvolvimento de J2ME (WTK) e oferece, além da biblioteca de classes, diversas ferramentas para o desenvolvimento de clientes de web services, como um gerador de classes Stub - baseados em arquivos WSDL - que são utilizadas para representar o serviço do web service na plataforma cliente.

A grande dificuldade para o desenvolvimento deste trabalho foi a especificação do serviço, pois a partir desta especificação foi possível criar o arquivo WSDL que serviu de base para a comunicação entre o cliente e o servidor do web service.

O desenvolvimento da aplicação J2ME possui características diferentes de uma aplicação desktop convencional, pois é necessário cuidar de aspectos de desempenho e de interfaceamento com o usuário que não são necessários em uma aplicação convencional que não possui tais requisitos.

Portanto, com este trabalho, consegue-se comprovar a possibilidade de desenvolver aplicações que interagem com web services através da comunicação por redes sem fio, e com capacidade de hardware limitada, utilizando as tecnologias descritas neste trabalho.

Ainda deve-se observar que existem muitas áreas a serem exploradas no âmbito destas tecnologias. A necessidade de construir aplicações seguras é um desafio para todas empresas, clientes e desenvolvedores, e o estudo de novas técnicas de

segurança para web services e J2ME são temas para trabalhos futuros, além do estudo de Web services com outras configurações J2ME. Outra possibilidade de trabalho futuro consiste no desenvolvimento de um suporte para disponibilização de web services, e não somente seus clientes, em dispositivos móveis.

8. BIBLIOGRAFIA

1. AHMAD, Saqib. *Security and Trust Services API (SATSA) for Java 2 Plataform, Micro Edition. Version 1.0, Specification* – 28/07/2004. <http://jcp.org/en/jsr/detail?id=177>. Acessado em 19/05/2005.

2. ALMEIDA, Leandro Batista de. *Introdução à J2ME e programação MIDP*. Brasil, Revista Mundo JAVA - Número 5. 2004.

3. BELLWOOD, Tom. *UDDI Version 2.04 API Specification* – 19/07/2002 <http://uddi.org/pubs/ProgrammersAPI_v2.htm>. Acessado em 28/11/2004.

4. BOOTH, David. HAAS, Hugo. MCCABE, Francis. NEWCOMER, Eric. CHAMPION, Michael. FERRIS, Chris. ORCHARD, David. *Web Service Architecture* – W3C Working Group Note 11/02/2004. <http://www.w3.org/TR/ws-arch>. Acessado em 14/03/2005.

5. BOX, Don. EHNEBUSKE, David. KAKIVAYA, Gopal. LAYMAN, Andrew. MENDELSOHN, Noah. NIELSEN, Henrik Frystyk. THATTE Satish. WINER, Dave. *Simple Object Access Protocol (SOAP) 1.1.* – 08/05/2000 <http://www.w3.org/TR/SOAP>. Acessado em 28/11/2004.

6. CHAPPELL, David. JEWELL, Tyler. *Java Web Services – Using Java in Service-Oriented Architectures.* O'Reilly - Edição 1 - 03/2002.

7. CHRISTENSEN, Erik. CURBERA, Francisco. MEREDITH, Greg. WEERAWARANA, Sanjiva. *Web Services Description Language (WSDL) 1.1* . – 15/03/2001 <http://www.w3.org/TR/wsdl.html>. Acessado em 28/11/2004.

8. CONSORTIUM, World Wide Web – *Document Object Model (DOM) Level 1 Specification – 01/10/1998.* <http://www.w3.org/TR/REC-DOM-Level-1/> Acessado em 12/06/2005.

9. CONSORTIUM, World Wide Web – *XML Path Language (Xpath) Version 1.0 – 16/11/1999.* <http://www.w3.org/TR/xpath/> Acessado em 12/06/2005.

10. CONSORTIUM, World Wide Web – *XML Schema Part 0: Primer Second Edition – 28/10/2004.* <http://www.w3.org/TR/xmlschema-0/> Acessado em 12/06/2005.

11. CONSORTIUM, World Wide Web – *XSL Transformations (XSLT). – 16/11/1999.* <http://www.w3.org/TR/xslt> Acessado em 12/06/2005.

12. COURTNEY, Jon. *JSR 68: J2ME Platform Specification* – 09/07/2002. <http://jcp.org/en/jsr/detail?id=68 >. Acessado em 14/05/2005.

13. EASTLAKE, Donald E., NILES, Kitty. *Secure XML: The New Syntax for Signatures and Encryption* – Addison Wesley – 19 Julho 2002.

14. ELLIS, Jon. YOUNG, Mark. *JSR 172: J2ME Web Services Specification* – 02/03/2004. < http://jcp.org/en/jsr/detail?id=172 >. Acessado em 14/05/2005.

15. FIELDING, R.; GETTYS, J.; MOGUL, J.; FRYSTYK, H.; MASINTER, L.; LEACH, P.; BERNERS-LEE, T.; *Hypertext Transfer Protocol -- HTTP/1.1* – Junho 1999 <http://www.ietf.org/rfc/rfc2616>. Acessado em 28/11/2004.

16. FOUNDATION, The Apache Software. *Apache Ant.* <http://ant.apache.org > Acessado em 20/05/2005.

17. FOUNDATION, The Apache Software. *Web Services – Axis.* <http://ws.apache.org/axis/overview.html> Acessado em 20/05/2005.

18. HENDRICKS, Mack. GALBRAITH, Ben. IRANI, Romin. MILBERNY, James. MODY, Tarak. TOST, Andre. BASHA, Jeelani S. . CABLE, Scott. *Profissional Java Web Services* – Alta Books – 2002.

19. INC, Sun Microsystems. *SATSA Developer's Guide. SATSA Reference Implementation* – 12/2004. <http://java.sun.com/j2me/docs/satsa-dg/>. Acessado em 19/05/2005.

20. KNUDSEN, Jonathan. *Wireless Developing with J2ME, Second Edition* – Apress – 2003.

21. MICROSOFT. Criando Web Services Seguros – 20/05/2004. <http://www.microsoft.com/brasil/security/guidance/topics/devsec/secmod85.mspx> Acessado em 15/05/2005

22. MORDANI, Rajiv. JSR 63: Java API for XML Processing 1.1 – 10/09/2002. <http://jcp.org/en/jsr/detail?id=63 >. Acessado em 14/05/2005.

23. MUCHOW, John W. *Core J2ME – Tecnologia & MIDP* – Pearson Makron Books – 2004.

24. NETWORK WORKING GROUP. *Hypertext Transfer Protocol – HTTP/1.1* – 06/1999. <http://www.w3.org/Protocols/rfc2616/rfc2616.html>. Acessado em 15/05/2005.

25. ORTIZ, C. Henrique. *Introduction to J2ME Web Services* – 04/2004. <http://developers.sun.com/techtopics/mobility/apis/articles/wsa/>. Acessado em 17/05/2005.

26. QUIN, Lean. *Extensible Markup Language (XML)* – 07/04/2005 <http://www.w3.org/XML/>. Acessado em 15/05/2005.

27. RIGS, Roger. TAIVALSAARI, Antero. PEURSEN, Jim Van. HUOPANIEMI, Jyri. PATEL, Mark. UOTILA, Aleksi. EDITOR, Jim Holliday. *Programming Wireless Devices with the Java 2 Plataform, Micro Edition, Second Edition* – Addison Wesley – 2003.

28. RODRIGUES, Nando. Desenvolvedoras Investem em integração de soluções. *Revista Computerworld - Edição 393 – 25/09/2003* < http://computerworld.uol.com.br/AdPortalv5/adCmsDocumentShow.aspx?DocumentID=76278>. Acessado em 02/11/2004.

29. SKONNARD, Aaron. *The Birth of Web Services.* – 10/2002. <http://msdn.microsoft.com/webservices/understanding/webservicebasics/default.aspx?pull=/msdnmag/issues/02/10/xmlfiles/default.aspx>. Acessado em 24/11/2004

9. ANEXOS

```
/*
 * Conta.java
 *
 * Created on 11 de Abril de 2005, 21:36
 */
package edu.ufsc.economize.business;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.util.Date;

/**
 * @author Lucas
 */
public class Conta {
    private int id;
    private String nome;
    private String tipoConta;
    private String frequencia;
    private Date dataExpiracao;

    /** Creates a new instance of Conta */
    public Conta() {}
    /**
     * @return Returns the dataExpiracao.
     */
    public Date getDataExpiracao() {
        return dataExpiracao;
    }
    /**
     * @param dataExpiracao
     *            The dataExpiracao to set.
     */
    public void setDataExpiracao(Date dataExpiracao) {
        this.dataExpiracao = dataExpiracao;
    }
    /**
     * @return Returns the frequencia.
     */
    public String getFrequencia() {
        return frequencia;
    }
    /**
     * @param frequencia
     *            The frequencia to set.
     */
    public void setFrequencia(String frequencia) {
        this.frequencia = frequencia;
    }
    /**
```

```
     * @return Returns the nome.
     */
    public String getNome() {
        return nome;
    }
    /**
     * @param nome
     *            The nome to set.
     */
    public void setNome(String nome) {
        this.nome = nome;
    }
    /**
     * @return Returns the tipoConta.
     */
    public String getTipoConta() {
        return tipoConta;
    }
    /**
     * @param tipoConta
     *            The tipoConta to set.
     */
    public void setTipoConta(String tipoConta) {
        this.tipoConta = tipoConta;
    }
    /**
     * Método responsável por serializar uma Conta.
     *
     * @param outputStream
     * @throws IOException
     */
    public byte[] serialize() throws IOException {
        ByteArrayOutputStream byteStream = new ByteArrayOutputStream();
        DataOutputStream outputStream = new DataOutputStream(byteStream);
        outputStream.writeUTF(this.nome);
        outputStream.writeUTF(tipoConta);
        outputStream.writeUTF(frequencia);
        outputStream.writeLong(dataExpiracao.getTime());
        return byteStream.toByteArray();
    }
    /**
     * Método responsável por deserializar uma conta.
     *
     * @param inputStream
     * @return
     * @throws IOException
     */
    public static Conta deserialize(byte[] dados) throws IOException {
        ByteArrayInputStream byteStream = new ByteArrayInputStream(dados);
        DataInputStream inputStream = new DataInputStream(byteStream);
        Conta conta = new Conta();
        conta.setNome(inputStream.readUTF());
        conta.setTipoConta(inputStream.readUTF());
        conta.setFrequencia(inputStream.readUTF());
        conta.setDataExpiracao(new Date(inputStream.readLong()));
        return conta;
    }
```

```
    /**
     * @return Returns the id.
     */
    public int getId() {
        return id;
    }
    /**
     * @param id
     *            The id to set.
     */
    public void setId(int id) {
        this.id = id;
    }
}



/*
 * MobileController.java
 *
 * Created on 11 de Abril de 2005, 21:36
 */


package edu.ufsc.economize.controller;

import java.io.IOException;
import java.util.Vector;
import javax.microedition.rms.InvalidRecordIDException;
import javax.microedition.rms.RecordEnumeration;
import javax.microedition.rms.RecordFilter;
import javax.microedition.rms.RecordStoreException;
import javax.microedition.rms.RecordStoreFullException;
import javax.microedition.rms.RecordStoreNotFoundException;
import javax.microedition.rms.RecordStoreNotOpenException;
import edu.ufsc.economize.business.Conta;
import edu.ufsc.economize.business.ItemConta;
import edu.ufsc.economize.persistencia.GerenciadorRMS;
import edu.ufsc.economize.view.extras.FiltroItemContaPorCodigoConta;

public class MobileController {
    private static GerenciadorRMS rmsConta;
    private static GerenciadorRMS rmsItemConta;

    public MobileController() {
        super();
        rmsConta = new GerenciadorRMS("db_conta");
        rmsItemConta = new GerenciadorRMS("db_item_conta");
    }
    /**
     * Método responsável por incluir uma conta a base de dados.
     *
     * @param conta
     * @throws RecordStoreFullException
     * @throws RecordStoreNotFoundException
     * @throws RecordStoreException
     * @throws IOException
```

```
     */
    public void incluirConta(Conta conta) throws RecordStoreFullException,
            RecordStoreNotFoundException, RecordStoreException, IOException {
        rmsConta.abrirRecordStore();
        int id = rmsConta.incluir(conta.serialize());
        conta.setId(id);
    }
    /**
     * Método responsável por adicionar um itemConta a base de dados RMS.
Porém,
     * ao adicionar, ele verifica se existe uma conta para este itemConta.
     *
     * @param itemConta
     * @throws RecordStoreException
     * @throws RecordStoreNotFoundException
     * @throws RecordStoreFullException
     * @throws IOException
     */
    public void incluirItemConta(ItemConta itemConta) throws
RecordStoreFullException,
            RecordStoreNotFoundException, RecordStoreException, IOException {
        rmsConta.abrirRecordStore();
        rmsConta.getRegistro(itemConta.getConta().getId());
        rmsItemConta.abrirRecordStore();
        int id = rmsItemConta.incluir(itemConta.serialize());
        itemConta.setIdItemConta(id);
    }
    /**
     * Método responsável por incluir uma conta a base de dados.
     *
     * @param conta
     * @throws RecordStoreFullException
     * @throws RecordStoreNotFoundException
     * @throws RecordStoreException
     * @throws IOException
     */
    public void atualizarConta(Conta conta) throws RecordStoreFullException,
            RecordStoreNotFoundException, RecordStoreException, IOException {
        rmsConta.abrirRecordStore();
        rmsConta.toString(conta.getId());
        rmsConta.atualizar(conta.getId(), conta.serialize());
    }
    public void atualizarItemConta(ItemConta itemConta) throws
RecordStoreNotOpenException,
            InvalidRecordIDException, RecordStoreFullException,
RecordStoreException, IOException {
        rmsItemConta.abrirRecordStore();
        rmsItemConta.toString(itemConta.getIdItemConta());
        rmsItemConta.atualizar(itemConta.getIdItemConta(),
itemConta.serialize());
    }
    /**
     * Método responsável por retornar todas as contas de uma base de dados.
     *
     * @return <code>Vector</code> Retorna uma coleção de <code>Conta</code>
     * @throws RecordStoreFullException
     * @throws RecordStoreNotFoundException
```

```
     * @throws RecordStoreException
     * @throws IOException
     */
    public Vector recuperarContas() throws RecordStoreFullException,
RecordStoreNotFoundException,
            RecordStoreException, IOException {
        rmsConta.abrirRecordStore();
        RecordEnumeration contas = rmsConta.recuperarDados(null);
        Vector collectionContas = new Vector();
        while (contas.hasNextElement()) {
            int id = contas.nextRecordId();
            Conta conta = Conta.deserialize(rmsConta.getRegistro(id));
            contas.keepUpdated(true);
            conta.setId(id);
            collectionContas.addElement(conta);
        }
        contas.destroy();
        return collectionContas;
    }
    /**
     * Método responsável por retornar uma coleção de ItemConta que possuem
     * chave para a conta.
     *
     * @param conta
     * @return
     * @throws RecordStoreFullException
     * @throws RecordStoreNotFoundException
     * @throws RecordStoreException
     * @throws IOException
     */
    public Vector recuperarItensPorConta(Conta conta) throws
RecordStoreFullException,
            RecordStoreNotFoundException, RecordStoreException, IOException {
        rmsItemConta.abrirRecordStore();
        RecordEnumeration itensConta = rmsItemConta.recuperarDados(null);
        Vector collectionItensConta = new Vector();
        while (itensConta.hasNextElement()) {
            int id = itensConta.nextRecordId();
            ItemConta itemConta =
ItemConta.deserialize(rmsItemConta.getRegistro(id));
            itensConta.keepUpdated(true);
            itemConta.setIdItemConta(id);
            if (conta.getId() == itemConta.getConta().getId()) {
                collectionItensConta.addElement(itemConta);
            }
        }
        itensConta.destroy();
        return collectionItensConta;
    }
    /**
     * Método responsável por calcular o valor total das receitas reais das
     * contas da coleção.
     *
     * @param contas
     * @return
     * @throws RecordStoreFullException
     * @throws RecordStoreNotFoundException
```

```
     * @throws RecordStoreException
     * @throws IOException
     */
    public double calcularValorTotalReceitaReal(Vector contas) throws
RecordStoreFullException,
            RecordStoreNotFoundException, RecordStoreException, IOException {
        double total = 0;
        for (int i = 0; i < contas.size(); i++) {
            Conta conta = (Conta) contas.elementAt(i);
            if (conta.getTipoConta().equalsIgnoreCase("RECEITA")) {
                total = +this.calcularValorTotalReal(conta);
            }
        }
        return total;
    }
    /**
     * Método responsável por calcular o valor total das receitas previstas
das
     * contas da coleção.
     *
     * @param contas
     * @return
     * @throws RecordStoreFullException
     * @throws RecordStoreNotFoundException
     * @throws RecordStoreException
     * @throws IOException
     */
    public double calcularValorTotalReceitaPrevista(Vector contas) throws
RecordStoreFullException,
            RecordStoreNotFoundException, RecordStoreException, IOException {
        double total = 0;
        for (int i = 0; i < contas.size(); i++) {
            Conta conta = (Conta) contas.elementAt(i);
            if (conta.getTipoConta().equalsIgnoreCase("RECEITA")) {
                total = +this.calcularValorTotalPrevisto(conta);
            }
        }
        return total;
    }
    /**
     * Método responsável por calcular o valor total das despesas reais das
     * contas da coleção.
     *
     * @param contas
     * @return
     * @throws RecordStoreFullException
     * @throws RecordStoreNotFoundException
     * @throws RecordStoreException
     * @throws IOException
     */
    public double calcularValorTotalDespesaReal(Vector contas) throws
RecordStoreFullException,
            RecordStoreNotFoundException, RecordStoreException, IOException {
        double total = 0;
        for (int i = 0; i < contas.size(); i++) {
            Conta conta = (Conta) contas.elementAt(i);
            if (conta.getTipoConta().equalsIgnoreCase("DESPESA")) {
```

```
                total = +this.calcularValorTotalReal(conta);
            }
        }
        return total;
    }
    /**
     * Método responsável por calcular o valor total das despesas previstas
das
     * contas da coleção.
     *
     * @param contas
     * @return
     * @throws RecordStoreFullException
     * @throws RecordStoreNotFoundException
     * @throws RecordStoreException
     * @throws IOException
     */
    public double calcularValorTotalDespesaPrevista(Vector contas) throws
RecordStoreFullException,
            RecordStoreNotFoundException, RecordStoreException, IOException {
        double total = 0;
        for (int i = 0; i < contas.size(); i++) {
            Conta conta = (Conta) contas.elementAt(i);
            if (conta.getTipoConta().equalsIgnoreCase("DESPESA")) {
                total = +this.calcularValorTotalPrevisto(conta);
            }
        }
        return total;
    }
    /**
     * Método responsável por calcular o valor total real de uma conta.
     *
     * @param conta
     * @return
     * @throws RecordStoreFullException
     * @throws RecordStoreNotFoundException
     * @throws RecordStoreException
     * @throws IOException
     */
    public double calcularValorTotalReal(Conta conta) throws
RecordStoreFullException,
            RecordStoreNotFoundException, RecordStoreException, IOException {
        double total = 0;
        RecordEnumeration itensConta = recuperarItens(conta);
        while (itensConta.hasNextElement()) {
            ItemConta item = ItemConta.deserialize(itensConta.nextRecord());
            total += item.getValorReal();
        }
        return total;
    }
    /**
     * Método responsável por calcular o valor total previsto de uma conta
     *
     * @param conta
     * @return
     * @throws RecordStoreFullException
     * @throws RecordStoreNotFoundException
```

```
     * @throws RecordStoreException
     * @throws IOException
     */
    public double calcularValorTotalPrevisto(Conta conta) throws
RecordStoreFullException,
            RecordStoreNotFoundException, RecordStoreException, IOException {
        double total = 0;
        RecordEnumeration itensConta = recuperarItens(conta);
        while (itensConta.hasNextElement()) {
            ItemConta item = ItemConta.deserialize(itensConta.nextRecord());
            total += item.getValorPrevisto();
        }
        return total;
    }
    /**
     * Método responsável por apagar uma Conta e todos os ItemConta
relacionados
     * a esta conta.
     *
     * @param conta
     * @throws RecordStoreFullException
     * @throws RecordStoreNotFoundException
     * @throws RecordStoreNotOpenException
     * @throws RecordStoreException
     */
    public void apagarConta(Conta conta) throws RecordStoreFullException,
            RecordStoreNotFoundException, RecordStoreNotOpenException,
RecordStoreException {
        RecordEnumeration itensConta = this.recuperarItens(conta);
        itensConta.keepUpdated(true);
        while (itensConta.hasNextElement()) {
            rmsItemConta.apagar(itensConta.nextRecordId());
        }
        itensConta.destroy();
        rmsConta.abrirRecordStore();
        rmsConta.apagar(conta.getId());
    }
    /**
     * Este método é responsável por recuperar um enumerado de itens que
possuem
     * uma chave para a conta.
     *
     * @param conta
     * @return
     * @throws RecordStoreFullException
     * @throws RecordStoreNotFoundException
     * @throws RecordStoreException
     * @throws RecordStoreNotOpenException
     */
    private RecordEnumeration recuperarItens(Conta conta) throws
RecordStoreFullException,
            RecordStoreNotFoundException, RecordStoreException,
RecordStoreNotOpenException {
        rmsItemConta.abrirRecordStore();
        RecordFilter filtroPorContas = new
FiltroItemContaPorCodigoConta(conta);
```

```
        RecordEnumeration itensConta =
rmsItemConta.recuperarDados(filtroPorContas);
        return itensConta;
    }
}

/*
 * EconomizeMidlet.java
 *
 * Created on 11 de Abril de 2005, 21:50
 */
package edu.ufsc.economize.midlet;

import javax.microedition.lcdui.Display;
import javax.microedition.midlet.MIDlet;
import edu.ufsc.economize.view.MobileView;
import edu.ufsc.economize.view.menu.MenuPrincipal;

/**
 * Classe que implementa o Midlet da aplicação. Esta classe é a única classe
que
 * consegue acessar o display do dispositivo. A partir dela será enviada as
 * ações para outras classes da Interface.
 *
 * @author Lucas
 * @version
 */
public class EconomizeMidlet extends MIDlet {
    public void startApp() {
        // ao invés de inicar o menu principal, poderia criar uma SplashScreen
        // com uma figura.
        try {
            MobileView view = new MenuPrincipal(this);
            view.setDisplay(Display.getDisplay(this));
            view.showView();
        } catch (RuntimeException e) {
            e.printStackTrace();
        }
    }
    public void pauseApp() {
    }
    public void destroyApp(boolean unconditional) {
        // o midlet só envia a mensagem para o display solicitando encerrando
a
        // aplicação
        notifyDestroyed();
    }
}


/*
 * GerenciadorRMS.java
 *
 * Created on 11 de Abril de 2005, 21:50
 */
package edu.ufsc.economize.persistencia;
```

```
import javax.microedition.rms.InvalidRecordIDException;
import javax.microedition.rms.RecordEnumeration;
import javax.microedition.rms.RecordFilter;
import javax.microedition.rms.RecordStore;
import javax.microedition.rms.RecordStoreException;
import javax.microedition.rms.RecordStoreFullException;
import javax.microedition.rms.RecordStoreNotFoundException;
import javax.microedition.rms.RecordStoreNotOpenException;

/**
 * Classe wrapper que manipula um registro RMS (Record Management System).
 *
 * @author Lucas
 */
public class GerenciadorRMS {
    private RecordStore recordStore;
    private String nomeRs;

    /**
     * Contrutor do GerenciadorRMS
     *
     * @param nomeRecordStore
     */
    public GerenciadorRMS(String nomeRecordStore) {
        super();
        this.nomeRs = nomeRecordStore;
    }
    /**
     * Abre ou cria o RecordStore
     *
     * @throws RecordStoreFullException
     * @throws RecordStoreNotFoundException
     * @throws RecordStoreException
     */
    public void abrirRecordStore() throws RecordStoreFullException,
RecordStoreNotFoundException,
            RecordStoreException {
        recordStore = RecordStore.openRecordStore(this.nomeRs, true);
    }
    /**
     * Fecha o RecordStore
     *
     * @throws RecordStoreNotOpenException
     * @throws RecordStoreException
     */
    public void fecharRecordStore() throws RecordStoreNotOpenException,
RecordStoreException {
        this.recordStore.closeRecordStore();
    }
    /**
     * Apaga o RecordStore com todos os seus registros.
     *
     * @throws RecordStoreNotFoundException
     * @throws RecordStoreException
     */
    public void apagarRecordStore() throws RecordStoreNotFoundException,
RecordStoreException {
```

```
            RecordStore.deleteRecordStore(this.nomeRs);
        }
        /**
         * Inclui um novo registro no RecordStore
         *
         * @param dados
         * @return
         * @throws RecordStoreNotOpenException
         * @throws RecordStoreFullException
         * @throws RecordStoreException
         */
        public int incluir(byte[] dados) throws RecordStoreNotOpenException,
RecordStoreFullException,
                RecordStoreException {
            int id = this.recordStore.addRecord(dados, 0, dados.length);
            return id;
        }
        /**
         * Recupera um registro no RecordStore
         *
         * @param dados
         * @return
         * @throws RecordStoreNotOpenException
         * @throws RecordStoreFullException
         * @throws RecordStoreException
         */
        public byte[] getRegistro(int id) throws RecordStoreNotOpenException,
RecordStoreFullException,
                RecordStoreException {
            return this.recordStore.getRecord(id);
        }
        /**
         * Recupera todos os dados de um RecordStore e retorna um
RecordEnumeration
         * com os dados.
         *
         * @return
         * @throws RecordStoreNotOpenException
         */
        public RecordEnumeration recuperarDados(RecordFilter filtro) throws
RecordStoreNotOpenException {
            // os parâmetros são null pois não está sendo aplicado nenhum filtro.
            return this.recordStore.enumerateRecords(filtro, null, true);
        }
        /**
         * Atualiza os dados do RecordStore da chave informada.
         *
         * @param chave
         * @param dados
         * @throws RecordStoreNotOpenException
         * @throws InvalidRecordIDException
         * @throws RecordStoreFullException
         * @throws RecordStoreException
         */
        public void atualizar(int chave, byte[] dados) throws
RecordStoreNotOpenException,
```

```
            InvalidRecordIDException, RecordStoreFullException,
RecordStoreException {
        this.recordStore.setRecord(chave, dados, 0, dados.length);
    }
    /**
     * Remove o registro do RecordStore.
     *
     * @param chave
     * @throws RecordStoreNotOpenException
     * @throws InvalidRecordIDException
     * @throws RecordStoreException
     */
    public void apagar(int chave) throws RecordStoreNotOpenException,
InvalidRecordIDException,
            RecordStoreException {
        this.recordStore.deleteRecord(chave);
    }
    /**
     * @param id
     * @return
     */
    public String toString(int id) {
        byte[] dados = new byte[100];
        try {
            this.recordStore.getRecord(id, dados, 0);
            String a = new String(dados);
            return a;
        } catch (InvalidRecordIDException e) {
            e.printStackTrace();
        } catch (RecordStoreException e) {
            e.printStackTrace();
        }
        return null;
    }
}



// This class was generated by 172 StubGenerator.
// Contents subject to change without notice.
// @generated
package edu.ufsc.economize.stub;

import javax.xml.rpc.JAXRPCException;
import javax.xml.namespace.QName;
import javax.microedition.xml.rpc.Operation;
import javax.microedition.xml.rpc.Type;
import javax.microedition.xml.rpc.ComplexType;
import javax.microedition.xml.rpc.Element;

public class EconomizeEndpoint_Stub implements
edu.ufsc.economize.stub.EconomizeEndpoint,
        javax.xml.rpc.Stub {
    private String[] _propertyNames;
    private Object[] _propertyValues;

    public EconomizeEndpoint_Stub() {
```

```
        _propertyNames = new String[] { ENDPOINT_ADDRESS_PROPERTY };
        _propertyValues = new Object[] {
"http://127.0.0.1:8080/economize/Economize" };
    }
    public void _setProperty(String name, Object value) {
        int size = _propertyNames.length;
        for (int i = 0; i < size; ++i) {
            if (_propertyNames[i].equals(name)) {
                _propertyValues[i] = value;
                return;
            }
        }
        // Need to expand our array for a new property
        String[] newPropNames = new String[size + 1];
        System.arraycopy(_propertyNames, 0, newPropNames, 0, size);
        _propertyNames = newPropNames;
        Object[] newPropValues = new Object[size + 1];
        System.arraycopy(_propertyValues, 0, newPropValues, 0, size);
        _propertyValues = newPropValues;
        _propertyNames[size] = name;
        _propertyValues[size] = value;
    }
    public Object _getProperty(String name) {
        for (int i = 0; i < _propertyNames.length; ++i) {
            if (_propertyNames[i].equals(name)) { return _propertyValues[i]; }
        }
        if (ENDPOINT_ADDRESS_PROPERTY.equals(name) ||
USERNAME_PROPERTY.equals(name)
                || PASSWORD_PROPERTY.equals(name)) { return null; }
        if (SESSION_MAINTAIN_PROPERTY.equals(name)) { return new
java.lang.Boolean(false); }
        throw new JAXRPCException("Stub does not recognize property: " +
name);
    }
    protected void _prepOperation(Operation op) {
        for (int i = 0; i < _propertyNames.length; ++i) {
            op.setProperty(_propertyNames[i], _propertyValues[i].toString());
        }
    }

    //
    // Begin user methods
    //
    //
    // End user methods
    //
    static {
        // Create all of the Type's that this stub uses, once.
    }
}
```

```
// This class was generated by 172 StubGenerator.
// Contents subject to change without notice.
// @generated
package edu.ufsc.economize.stub;

public interface EconomizeEndpoint extends java.rmi.Remote {}


/*
 * IConstants.java
 *
 * Created on 14 de Abril de 2005, 00:24
 */
package edu.ufsc.economize.util;

/**
 * @author Lucas
 */
public interface IConstantsMenuPrincipal {
    public final int MENU_PRINCIPAL_CONTA = 0;
    public final int MENU_PRINCIPAL_ITEM_CONTA = 1;
    public final int MENU_PRINCIPAL_CALCULAR = 2;
    public final int MENU_SINCRONIZAR_CONTA = 3;
    public final int MENU_ENCERRAR_CONTA = 4;
}



/*
 * MobileView.java
 *
 * Created on 13 de Abril de 2005, 23:11
 */
package edu.ufsc.economize.view;

import java.util.Stack;
import javax.microedition.lcdui.Alert;
import javax.microedition.lcdui.AlertType;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Image;
import javax.microedition.midlet.MIDlet;
import edu.ufsc.economize.controller.MobileController;

/**
 * @author Lucas
 */
public abstract class MobileView implements CommandListener {
    // atributos
    protected MIDlet midlet;
    protected static Display display;
    protected MobileController controller;
    protected static Stack stackView;

    // construtor
```

```
    /**
     * @param midlet
     */
    public MobileView(MIDlet midlet) {
        this.midlet = midlet;
        if (controller == null) {
            controller = new MobileController();
        }
        if (stackView == null) {
            stackView = new Stack();
        }
    }
    /**
     * Método responsável por retornar uma instância da view atual.
     */
    private Displayable getViewInstance() throws Exception {
        Displayable disp = getView();
        if (disp == null) {
            criarView();
        } else {
            atualizarView();
        }
        disp = getView();
        disp.setCommandListener((CommandListener) this);
        return disp;
    }
    /**
     * Retorna o MIDlet associado a uma instância desta classe.
     *
     * @return
     */
    protected MIDlet getMidlet() {
        return this.midlet;
    }
    /**
     * Método responsável por mostrar a view atual
     */
    public void showView() {
        try {
            Displayable disp = getViewInstance();
            display.setCurrent(disp);
        } catch (Exception e) {
            e.printStackTrace();
            showAlert("Erro ao mostrar tela", e.getMessage(), Alert.FOREVER,
AlertType.ERROR);
        }
    }
    public static Display getDisplay() {
        return display;
    }
    public void setDisplay(Display display) {
        MobileView.display = display;
    }
    /**
     * Método responsável por mostrar uma mensagem de Alerta no display.
     *
     * @param msg
```

```
     * @param tempoAlerta
     * @param tipo
     */
    public void showAlert(String titulo, String texto, int tempoAlerta,
AlertType tipo) {
        Alert errorAlert = new Alert(titulo);
        errorAlert.setTimeout(tempoAlerta);
        errorAlert.setString(texto);
        errorAlert.setType(tipo);
        display.setCurrent(errorAlert);
    }
    /**
     * Método responsável por mostrar uma mensagem de Alerta no display.
     *
     * @param msg
     * @param tempoAlerta
     * @param tipo
     */
    public void showAlert(String titulo, String texto, int tempoAlerta,
AlertType tipo,
            Displayable nextDisplay) {
        Alert errorAlert = new Alert(titulo);
        errorAlert.setTimeout(tempoAlerta);
        errorAlert.setString(texto);
        errorAlert.setType(tipo);
        display.setCurrent(errorAlert, nextDisplay);
    }
    /**
     * Método responsável por mostrar uma mensagem de Alerta no display.
     *
     * @param titulo
     * @param texto
     * @param tempoAlerta
     * @param tipo
     * @param imagem
     * @param nextDisplay
     *            TODO
     */
    public void showAlert(String titulo, String texto, int tempoAlerta,
AlertType tipo, Image imagem) {
        Alert errorAlert = new Alert(titulo, texto, imagem, tipo);
        errorAlert.setTimeout(tempoAlerta);
        display.setCurrent(errorAlert);
    }
    // métodos que devem ser implementados na subclasse.
    public abstract void commandAction(Command c, Displayable s);
    protected abstract Displayable getView();
    protected abstract void criarView();
    protected abstract void atualizarView();
}



package edu.ufsc.economize.view.extras;

import java.util.Vector;
import javax.microedition.lcdui.Image;
```

```
import javax.microedition.lcdui.List;
import edu.ufsc.economize.business.Conta;

public class ContasList extends List {
    protected Vector contas;

    public ContasList(String arg0, int arg1) {
        super(arg0, arg1);
        contas = new Vector();
    }
    /**
     * Método responsável por adicionar a lista uma conta.
     *
     * @param arg0
     * @param conta
     * @param arg1
     * @return
     */
    public int append(Conta conta, Image arg1) {
        int indice = super.append(conta.getNome(), arg1);
        contas.insertElementAt(conta, indice);
        return indice;
    }
    public Conta getContaBySelectedIndex() {
        int indice = super.getSelectedIndex();
        return (Conta) contas.elementAt(indice);
    }
}
```

```
package edu.ufsc.economize.view.extras;

import javax.microedition.lcdui.ChoiceGroup;

public class EconomizeChoiceGroup extends ChoiceGroup {
    public EconomizeChoiceGroup(String label, int choiceType) {
        super(label, choiceType);
    }
    public void setSelectedIndex(String valorDefault, boolean selected) {
        for (int indice = 0; indice < this.size(); indice++) {
            if (getString(indice).equals(valorDefault)) {
                super.setSelectedIndex(indice, selected);
            }
        }
    }
}
```

```
package edu.ufsc.economize.view.extras;

import java.io.IOException;
import javax.microedition.rms.RecordFilter;
import edu.ufsc.economize.business.Conta;
```

```
import edu.ufsc.economize.business.ItemConta;

public class FiltroItemContaPorCodigoConta implements RecordFilter {
    private Conta conta;

    public FiltroItemContaPorCodigoConta(Conta conta) {
        super();
        this.conta = conta;
    }
    public boolean matches(byte[] itemConta) {
        try {
            ItemConta item = ItemConta.deserialize(itemConta);
            if (item.getConta().getId() == this.conta.getId()) { return true;
}
        } catch (IOException e) {
            e.printStackTrace();
            return false;
        }
        return false;
    }
}
```

```
package edu.ufsc.economize.view.extras;

import java.util.Vector;
import javax.microedition.lcdui.Image;
import javax.microedition.lcdui.List;
import edu.ufsc.economize.business.ItemConta;

public class ItemContaList extends List {
    protected Vector itensConta;

    public ItemContaList(String arg0, int arg1) {
        super(arg0, arg1);
        itensConta = new Vector();
    }
    /**
     * Método responsável por adicionar a lista uma conta.
     *
     * @param arg0
     * @param itemConta
     * @param arg1
     * @return
     */
    public int append(ItemConta itemConta, Image arg1) {
        int indice = super.append(itemConta.getDescricaoItem(), arg1);
        itensConta.insertElementAt(itemConta, indice);
        return indice;
    }
    public ItemConta getItemContaBySelectedIndex() {
        int indice = super.getSelectedIndex();
        return (ItemConta) itensConta.elementAt(indice);
    }
}
```

```
package edu.ufsc.economize.view.form.calculo;

import java.util.Vector;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.List;
import javax.microedition.midlet.MIDlet;
import edu.ufsc.economize.view.MobileView;

public class TotalizadorContasForm extends MobileView {
    private Vector contasSelecionadas;
    private double totalReceitaReal;
    private double totalReceitaPrevista;
    private double totalDespesaReal;
    private double totalDespesaPrevista;
    private double resultadoGeralReal;
    private double resultadoGeralPrevisto;
    private static List calculosList;
    private static Command voltarCommand;
    private static Command okCommand;
    private static Command selecioneCommand;

    public TotalizadorContasForm(MIDlet midlet, Vector contasSelecionadas) {
        super(midlet);
        this.contasSelecionadas = contasSelecionadas;
    }
    protected void criarView() {
        voltarCommand = new Command("VOLTAR", Command.BACK, 0);
        okCommand = new Command("OK", Command.SCREEN, 1);
        calculosList = new List("RESULTADO ", List.IMPLICIT);
        this.calcular();
        calculosList.append("TOTAL RECEITA REAL: " + totalReceitaReal, null);
        calculosList.append("TOTAL DESPESA REAL: " + totalDespesaReal, null);
        calculosList.append("------------------------------", null);
        calculosList.append("RESULTADO REAL: " + resultadoGeralReal, null);
        calculosList.append("                                ", null);
        calculosList.append("                                ", null);
        calculosList.append("TOTAL RECEITA PREVISTA: " + totalReceitaPrevista,
null);
        calculosList.append("TOTAL DESPESA PREVISTA: " + totalDespesaPrevista,
null);
        calculosList.append("------------------------------", null);
        calculosList.append("RESULTADO PREVISTO: " + resultadoGeralPrevisto,
null);
        calculosList.addCommand(okCommand);
        calculosList.addCommand(voltarCommand);
    }
    private void calcular() {
        try {
            totalReceitaReal = super.controller
                    .calcularValorTotalReceitaReal(this.contasSelecionadas);
            totalReceitaPrevista = super.controller

.calcularValorTotalReceitaPrevista(this.contasSelecionadas);
            totalDespesaReal = super.controller
                    .calcularValorTotalDespesaReal(this.contasSelecionadas);
            totalDespesaPrevista = super.controller
```

```
.calcularValorTotalDespesaPrevista(this.contasSelecionadas);
            resultadoGeralReal = totalReceitaReal - totalDespesaReal;
            resultadoGeralPrevisto = totalReceitaPrevista -
totalDespesaPrevista;
        } catch (Exception e) {
            e.printStackTrace();
            calculosList.append("ERRO! " + e.getMessage(), null);
        }
    }
    protected Displayable getView() {
        return calculosList;
    }
    protected void atualizarView() {
        criarView();
    }
    public void commandAction(Command c, Displayable s) {
        if (c == voltarCommand) {
            ((MobileView) MobileView.stackView.pop()).showView();
        } else if ((c == okCommand) || (c == List.SELECT_COMMAND)) {
            ((MobileView) MobileView.stackView.pop()).showView();
        }
    }
}
```

```
package edu.ufsc.economize.view.form.conta;

import javax.microedition.lcdui.Alert;
import javax.microedition.lcdui.AlertType;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Form;
import javax.microedition.midlet.MIDlet;
import edu.ufsc.economize.business.Conta;
import edu.ufsc.economize.view.MobileView;

public class AdicionarContaForm extends MobileView {
    private static Form contaForm;
    private static Command voltarCommand;
    private static Command okCommand;

    public AdicionarContaForm(MIDlet midlet) {
        super(midlet);
    }
    protected void criarView() {
        voltarCommand = new Command("VOLTAR", Command.BACK, 0);
        okCommand = new Command("OK", Command.SCREEN, 1);
        contaForm = new ContaForm("ADICIONAR CONTA");
        // relaciono os comandos com o form
        contaForm.addCommand(voltarCommand);
        contaForm.addCommand(okCommand);
    }
    protected Displayable getView() {
        return contaForm;
    }
```

```
    protected void atualizarView() {
        criarView();
    }
    public void commandAction(Command c, Displayable s) {
        if (c == voltarCommand) {
            ((MobileView) MobileView.stackView.pop()).showView();
        } else if (c == okCommand) {
            adicionarConta();
            ((MobileView) MobileView.stackView.pop()).showView();
        }
    }
    /**
     * Método privado que faz a validação dos dados do formulário da conta e
     * adiciona a conta ao registro.
     */
    private void adicionarConta() {
        try {
            ((ContaForm) contaForm).validate();
        } catch (Exception e) {
            showAlert("Erro ao adicionar conta", e.getMessage(),
Alert.FOREVER, AlertType.ERROR,
                    contaForm);
            e.printStackTrace();
        }
        Conta conta = ((ContaForm) contaForm).criarContaByForm();
        try {
            super.controller.incluirConta(conta);
            showAlert("Sucesso", "Conta " + conta.getNome() + " incluída com
sucesso!",
                    Alert.FOREVER, AlertType.INFO);
        } catch (Exception e) {
            showAlert("Erro ao incluir conta no BD", e.getMessage(),
Alert.FOREVER, AlertType.ERROR);
            e.printStackTrace();
        }
    }
}



package edu.ufsc.economize.view.form.conta;

import java.util.Vector;
import javax.microedition.lcdui.Alert;
import javax.microedition.lcdui.AlertType;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.List;
import javax.microedition.midlet.MIDlet;
import edu.ufsc.economize.business.Conta;
import edu.ufsc.economize.view.MobileView;

public class ApagarContaForm extends MobileView {
    private static List contasApagar;
    private Vector contas;
    private static Command voltarCommand;
    private static Command okCommand;
```

```
    private static Command selecioneCommand;

    public ApagarContaForm(MIDlet midlet) {
        super(midlet);
    }
    protected void criarView() {
        voltarCommand = new Command("VOLTAR", Command.BACK, 0);
        okCommand = new Command("OK", Command.SCREEN, 1);
        contasApagar = new List("SELECIONE AS CONTAS QUE SERAO APAGADAS",
List.MULTIPLE);
        // esta ordem deverá ser seguida no método commandAction
        try {
            contas = super.controller.recuperarContas();
            for (int i = 0; i < contas.size(); i++) {
                Conta conta = (Conta) contas.elementAt(i);
                contasApagar.append(conta.getNome(), null);
            }
            contasApagar.addCommand(okCommand);
        } catch (Exception e) {
            contasApagar.append("Não existe conta a ser totalizada.", null);
            e.printStackTrace();
        }
        contasApagar.addCommand(voltarCommand);
        contasApagar.addCommand(okCommand);
    }
    protected Displayable getView() {
        return contasApagar;
    }
    protected void atualizarView() {
        criarView();
    }
    public void commandAction(Command c, Displayable s) {
        if (c == voltarCommand) {
            ((MobileView) MobileView.stackView.pop()).showView();
        } else if ((c == okCommand) || (c == List.SELECT_COMMAND)) {
            boolean selecionados[] = new boolean[contasApagar.size()];
            contasApagar.getSelectedFlags(selecionados);
            try {
                for (int i = 0; i < selecionados.length; i++) {
                    if (selecionados[i]) {
                        super.controller.apagarConta((Conta)
contas.elementAt(i));
                    }
                }
                ((MobileView) MobileView.stackView.pop()).showView();
            } catch (Exception e) {
                e.printStackTrace();
                super.showAlert("Erro ao apagar Conta", e.getMessage(),
Alert.FOREVER,
                        AlertType.ERROR);
            }
        }
    }
}
```

```
package edu.ufsc.economize.view.form.conta;

import java.util.Date;
import javax.microedition.lcdui.ChoiceGroup;
import javax.microedition.lcdui.DateField;
import javax.microedition.lcdui.Form;
import javax.microedition.lcdui.Item;
import javax.microedition.lcdui.TextField;
import edu.ufsc.economize.business.Conta;
import edu.ufsc.economize.view.extras.EconomizeChoiceGroup;

public class ContaForm extends Form {
    private static TextField nomeContaTxtField;
    private static EconomizeChoiceGroup tipoContaChGroup;
    private static EconomizeChoiceGroup frequenciaChGroup;
    private static DateField dataExpiracaoDtField;

    public ContaForm(String arg0) {
        super(arg0);
        this.criarDefaultForm();
    }
    public ContaForm(String arg0, Item[] arg1) {
        super(arg0, arg1);
    }
    public ContaForm(String nomeForm, Conta conta) {
        super(nomeForm);
        if (conta != null) {
            criarFormByConta(conta);
        }
    }
    protected void criarDefaultForm() {
        criarTxtFieldNomeConta("");
        criarChoiceGroupTipoConta(null);
        criarChoiceGroupFrequencia(null);
        criarDateFieldDataExpiracaoConta(null);
    }
    protected void criarFormByConta(Conta conta) {
        criarTxtFieldNomeConta(conta.getNome());
        criarChoiceGroupTipoConta(conta.getTipoConta());
        criarChoiceGroupFrequencia(conta.getFrequencia());
        criarDateFieldDataExpiracaoConta(conta.getDataExpiracao());
    }
    private void criarTxtFieldNomeConta(String valor) {
        nomeContaTxtField = new TextField("NOME", valor, 20, TextField.ANY);
        nomeContaTxtField.setLayout(TextField.LAYOUT_CENTER);
        this.append(nomeContaTxtField);
    }
    private void criarChoiceGroupTipoConta(String tipoContaDefault) {
        tipoContaChGroup = new EconomizeChoiceGroup("TIPO CONTA",
ChoiceGroup.POPUP);
        tipoContaChGroup.append("RECEITA", null);
        tipoContaChGroup.append("DESPESA", null);
        if (tipoContaDefault == null) {
            tipoContaDefault = "RECEITA";
        }
        tipoContaChGroup.setSelectedIndex(tipoContaDefault, true);
        this.append(tipoContaChGroup);
```

```
    }
    private void criarChoiceGroupFrequencia(String frequenciaDefault) {
        frequenciaChGroup = new EconomizeChoiceGroup("FREQUENCIA",
ChoiceGroup.POPUP);
        frequenciaChGroup.append("MENSAL", null);
        frequenciaChGroup.append("ANUAL", null);
        frequenciaChGroup.append("DIARIA", null);
        frequenciaChGroup.append("UNICO EVENTO", null);
        if (frequenciaDefault == null) {
            frequenciaDefault = "MENSAL";
        }
        frequenciaChGroup.setSelectedIndex(frequenciaDefault, true);
        this.append(frequenciaChGroup);
    }
    private void criarDateFieldDataExpiracaoConta(Date dataDefault) {
        dataExpiracaoDtField = new DateField("DATA EXPIRAÇÃO CONTA",
DateField.DATE);
        if (dataDefault != null) {
            dataExpiracaoDtField.setDate(dataDefault);
        }
        dataExpiracaoDtField.setLayout(DateField.LAYOUT_BOTTOM);
        this.append(dataExpiracaoDtField);
    }
    public Conta criarContaByForm() {
        Conta conta = new Conta();
        conta.setNome(nomeContaTxtField.getString());

conta.setTipoConta(tipoContaChGroup.getString(tipoContaChGroup.getSelectedInde
x()));

conta.setFrequencia(frequenciaChGroup.getString(frequenciaChGroup.getSelectedI
ndex()));
        conta.setDataExpiracao(dataExpiracaoDtField.getDate());
        return conta;
    }
    public void validate() throws Exception {
        if (nomeContaTxtField.getString().trim().equals("")) {
            throw new Exception("Nome de conta inválido.");
        } else if (dataExpiracaoDtField.getDate() == null) { throw new
Exception(
                "Data de expiração da conta inválida."); }
    }
}



package edu.ufsc.economize.view.form.conta;

import javax.microedition.lcdui.Alert;
import javax.microedition.lcdui.AlertType;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.List;
import javax.microedition.midlet.MIDlet;
import edu.ufsc.economize.business.Conta;
import edu.ufsc.economize.view.MobileView;
```

```
public class EditarContaForm extends MobileView {
    private Conta conta;
    private static ContaForm contaForm;
    private static ListaContasForm listaContasForm;
    private static Command voltarCommand;
    private static Command okCommand;
    private static Command selecioneCommand;

    public EditarContaForm(MIDlet midlet) {
        super(midlet);
        this.conta = null;
    }
    public void setConta(Conta conta) {
        this.conta = conta;
    }
    protected void criarView() {
        voltarCommand = new Command("VOLTAR", Command.BACK, 0);
        okCommand = new Command("OK", Command.SCREEN, 1);
        contaForm = new ContaForm("EDITAR CONTA", this.conta);
        contaForm.addCommand(voltarCommand);
        contaForm.addCommand(okCommand);
    }
    protected Displayable getView() {
        return contaForm;
    }
    protected void atualizarView() {
        criarView();
    }
    public void showView() {
        if (conta == null) {
            listaContasForm = new ListaContasForm(midlet, this);
            listaContasForm.showView();
        } else {
            super.showView();
        }
    }
    public void commandAction(Command c, Displayable s) {
        if (c == voltarCommand) {
            ((MobileView) MobileView.stackView.pop()).showView();
        } else if ((c == okCommand) || (c == List.SELECT_COMMAND)) {
            try {
                contaForm.validate();
                try {
                    Conta contaAtualizada = contaForm.criarContaByForm();
                    contaAtualizada.setId(conta.getId());
                    super.controller.atualizarConta(contaAtualizada);
                    ((MobileView) MobileView.stackView.pop()).showView();
                } catch (Exception e) {
                    e.printStackTrace();
                    showAlert("Erro ao atualizar conta no BD", e.getMessage(),
Alert.FOREVER,
                            AlertType.ERROR);
                }
            } catch (Exception e) {
                showAlert("Erro ao editar conta", e.getMessage(),
Alert.FOREVER, AlertType.ERROR);
                e.printStackTrace();
```

```
            }
        }
    }
}
```

```
package edu.ufsc.economize.view.form.conta;

import java.util.Vector;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.List;
import javax.microedition.midlet.MIDlet;
import edu.ufsc.economize.business.Conta;
import edu.ufsc.economize.view.MobileView;
import edu.ufsc.economize.view.extras.ContasList;
import edu.ufsc.economize.view.form.itemconta.AdicionarItemContaForm;
import edu.ufsc.economize.view.form.itemconta.EditarItemContaForm;

public class ListaContasForm extends MobileView {
    private static ContasList listaContas;
    private MobileView form;
    private static Command voltarCommand;
    private static Command okCommand;

    public ListaContasForm(MIDlet midlet, MobileView form) {
        super(midlet);
        this.form = form;
    }
    protected void criarView() {
        voltarCommand = new Command("VOLTAR", Command.BACK, 0);
        okCommand = new Command("OK", Command.SCREEN, 1);
        listaContas = new ContasList("SELECIONE UMA CONTA", List.IMPLICIT);
        try {
            Vector contas = super.controller.recuperarContas();
            for (int i = 0; i < contas.size(); i++) {
                Conta conta = (Conta) contas.elementAt(i);
                listaContas.append(conta, null);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        listaContas.addCommand(voltarCommand);
        listaContas.addCommand(okCommand);
    }
    protected void adicionarConta(String conta) {
        listaContas.append(conta, null);
    }
    protected void atualizarView() {
        criarView();
    }
    public Displayable getView() {
        return listaContas;
    }
    public void commandAction(Command c, Displayable s) {
        if (c == okCommand) {
```

```
            if (form instanceof EditarContaForm) {
                ((EditarContaForm)
form).setConta(listaContas.getContaBySelectedIndex());
                ((EditarContaForm) form).showView();
            } else if (form instanceof AdicionarItemContaForm) {
                ((AdicionarItemContaForm)
form).setConta(listaContas.getContaBySelectedIndex());
                ((AdicionarItemContaForm) form).showView();
            } else if (form instanceof EditarItemContaForm) {
                ((EditarItemContaForm)
form).setConta(listaContas.getContaBySelectedIndex());
                ((EditarItemContaForm) form).showView();
            }
        } else if (c == voltarCommand) {
            ((MobileView) MobileView.stackView.pop()).showView();
        }
    }
}
```

Desenvolvimento de aplicações em dispositivos móveis com J2ME e integração com Web Services

Lucas de Souza Reis Gomes

Universidade Federal de Santa Catarina
lucassrg@inf.ufsc.br

RESUMO

Neste artigo, descrevo em linhas gerais a tecnologia J2ME, como é composta sua arquitetura, visto que ela é utilizada no desenvolvimento de aplicações móveis, além de estudar a arquitetura de Web Services, utilizada para oferecer e acessar serviços web. E para finalizar, também descrevo o pacote opcional JSR-172 que é utilizado para a comunicação entre um Web Service e uma aplicação J2ME..

PALAVRAS-CHAVE

J2ME, Web Services, CLDC, JSR-172.

1.INTRODUÇÃO

O desenvolvimento de aplicações distribuídas sempre foi um campo muito importante desde que os sistemas migraram de mainframes para computadores pessoais, estações de trabalho ou dispositivos móveis que podem ser acessados através de uma rede. A integração entre aplicações sempre foi um grande problema para desenvolvedores de sistemas. Em sistemas legados, isto ocorria devido às limitações tecnológicas das plataformas em que foram desenvolvidos. Já em sistemas mais modernos, como os ERP (Enterprise Resource Management), CRM (Customer Relationship Management), ou Supply Chain, esta dificuldade também está presente, principalmente quando são desenvolvidos por fabricantes diferentes.

Devido à necessidade de integração dos sistemas, surgiu o conceito de operação conjunta, onde sistemas heterogêneos devem ser capazes de estabelecer comunicação e compartilhar dados, sem estarem ligados fisicamente entre si, através de uma comunicação transparente. Através deste conceito, surgiu a idéia de web services, onde serviços são disponibilizados e podem ser acessados remotamente sem intervenção humana, e utilizam-se de protocolos padrões para definir sua arquitetura.

O mercado de Web Services está explodindo em novas oportunidades, e a cada dia os desenvolvedores criam dispositivos mais poderosos e novas aplicações. Os telefones celulares e os handhelds (PDAs) estão se tornando populares entre os consumidores de todo o mundo, onde é possível cada vez mais utilizá-los em atividades cotidianas.

Antes restritos a executar aplicações simples como calculadoras e calendários, hoje estes dispositivos estão ganhando aplicações mais complexas como streaming de vídeo, tornando possível para viajantes ou pessoas de negócios que estão sempre ocupadas, assistir novos clipes em tempo real ou apresentações de negócios a partir de um táxi ou em um hotel.

Por outro lado, o crescimento do mercado de telefonia móvel e a demanda por aplicações que possam ser executadas em dispositivos portáteis, com a possibilidade de comunicação com outras aplicações, utilizando uma rede wireless, motivou o estudo da arquitetura Java J2ME. Esta possui as mesmas características da plataforma Java convencional (J2SE – Java 2 Standard Edition), e pode ser executada em qualquer plataforma que possua uma máquina virtual instalada, sem a necessidade de se prender a um fabricante ou a uma tecnologia, além de ser uma linguagem de padrão aberto e que oferece um conjunto de tecnologias e APIs.

A arquitetura Java J2ME possui características diferentes da arquitetura tradicional do JAVA, a J2SE, pois ela é otimizada para o dispositivo para a qual ela foi desenvolvida.

2. A ARQUITETURA DE J2ME

Reconhecendo que uma tecnologia não é a melhor opção para todas as áreas da indústria, a Sun Microsystems[COURTNEY, 2002] agrupou a plataforma Java 2 em três edições, cada uma com um comportamento específico para seu mercado:

- Java 2 Enterprise Edition (J2EE) – utilizada em servidores corporativos.
- Java 2 Standard Edition (J2SE) – para uso residencial e mercado desktop.

- Java 2 Micro Edition (J2ME) – para dispositivos pequenos e com poucos recursos de memória e processamento.

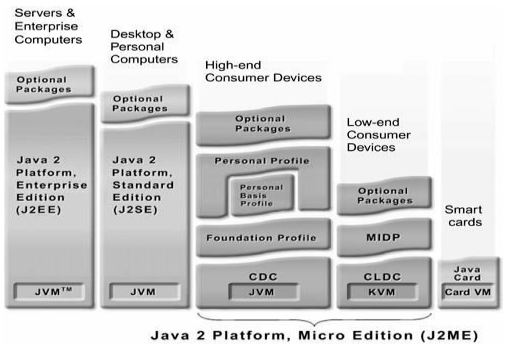

Figura 23 - Edições da plataforma Java 2 e seu mercado [RIGS, 2003]

A plataforma Java 2 Micro Edition está focada para um mercado muito amplo que está crescendo rapidamente, e abrange um grande número de dispositivos, desde pagers, aparelhos celulares, codificadores de televisão, até computadores pessoais. E assim como as outras plataformas Java, J2ME manteve as características de portabilidade de código, segurança, comunicação com rede, e grande escalabilidade.

J2ME pode ser dividida em duas categorias que agrupam dispositivos com características semelhantes. Dispositivos de alto-nível são representados pela configuração CDC (Connected Device Configuration). Exemplos destes dispositivos são os computadores de bordo de automóveis, conversores de TV por assinatura (set-top-boxes) e demais dispositivos que possuem uma rica capacidade de interfaces com usuário, tamanho de memória de pelo menos 2 a 4 megabytes, persistência e conexões de rede de alta velocidade. Já os dispositivos de baixo-nível são representados pela configuração CLDC (Connected Limited Device Configuration), que é utilizada em telefones celulares, pagers e outros dispositivos que possuem uma interface simples para o usuário (se comparada a um computador pessoal), pouca capacidade de memória (de 128 a 256 kilobytes), redes com baixa taxa de transferência e conexão intermitente. Muitos desses dispositivos são alimentados por baterias.

O mercado consumidor necessita medir a flexibilidade com a qual uma tecnologia é empregada ao desenvolver uma aplicação. Isto se mostra necessário devido ao grande número de tipos de dispositivos, configurações de hardware, ao modo como as aplicações são operadas pelo dispositivo (através do teclado, comando de voz, etc), ao grande número de aplicações e funcionalidades, e à necessidade de adaptação a mudanças e crescimento em escala.

A arquitetura J2ME foi projetada para ser modular e escalável para suportar estas necessidades. Contudo, para permitir isso, o ambiente J2ME oferece diversas tecnologias de máquina virtual, cada uma otimizada para diferentes tipos de processador, e memórias encontradas no mercado consumidor. E para suportar essa personalização e extensibilidade foram definidos pela arquitetura J2ME três conceitos: Configuração, Perfil e Pacotes opcionais.

2.1 Configuração

A configuração define o mínimo que um desenvolvedor pode esperar de um dispositivo, agrupando-os de acordo com a semelhança entre seus requisitos de memória e de hardware[MUCHOW, 2004]. Ou seja, uma configuração especifica:

- quais as funcionalidades suportadas pela linguagem Java;
- quais as funcionalidades suportadas pela máquina virtual;
- quais as bibliotecas Java e APIs são suportadas.

O ambiente J2ME pode ser instalado sobre mais de uma configuração. Cada configuração especifica a linguagem, a máquina virtual e um conjunto de APIs que será implementado sobre uma família de dispositivos. Esta camada é menos visível ao usuário, porém, é muito importante para os desenvolvedores de perfil.

Foram definidos dois padrões de configuração J2ME, o CLDC (Connected, Limited Device Configuration) e o CDC (Connected Device Configuration)[COURTNEY, 2002].

2.1.1 CLDC

A configuração CLDC, segundo [RIGS, 2003], é focada para os dispositivos considerados de uso pessoal, móvel, operado por bateria como telefones celulares, pagers, organizadores pessoais e outros aparelhos de tamanho semelhante. Esta configuração inclui algumas novas bibliotecas que não fazem parte da arquitetura J2SE e que foram desenvolvidas especificamente para suprir algumas necessidades dos dispositivos com poucos recursos.

As funcionalidades que não são abrangidas pelo CLDC podem ser implementadas através do perfil para manter a interoperabilidade entre os dispositivos que utilizam essa configuração, e não exceder a limitação de memória definida na especificação ou excluir alguma categoria de dispositivo

2.1.2 CDC

A configuração CDC é focada para dispositivos como aparelhos conversores de TV a cabo, sistemas navegadores de carro e PDAs, que devem possuir capacidade para suportar todas as funcionalidades da máquina virtual Java (JVM)[COURTNEY, 2002]. Quando se falou em CLDC e pequenos dispositivos, era indispensável pensar em termos de memória volátil e não volátil. Porém, ao tratar-se de CDC, este requisito não é tão importante, pois esta configuração especifica a necessidade que um dispositivo tenha no mínimo 512KB de memória ROM, 256KB de memória RAM, e algum tipo de conexão de rede.

2.2 Perfil

O perfil é a camada no topo da configuração, responsável por adicionar APIs e especificações necessárias para desenvolver aplicações para famílias específicas de dispositivos[RIGS, 2003].

Diversos perfis estão sendo desenvolvidos sobre a JCP (Java Community Process), que são agrupados conforme a configuração sobre a qual ele é especificado. Os perfis MIDP (Mobile Information Device Profile) e IMP (Information Module Profile) são especificados sobre CLDC e os perfis FP (Foundation Profile), o PP (Personal Profile) e o PBP (Personal Basis Profile) sobre CDC.

O perfil IMP foi desenvolvido para aplicações de dispositivos que utilizam rede mas não possuem um rico display gráfico, ou com recursos muito limitados como os aparelhos de emergência, medidores de estacionamento, módulos wireless de sistemas de alarme residenciais, dispositivos medidores de indústrias e assim por diante. O IMP é um subconjunto do perfil MIDP versão 1.0.

O perfil MIDP é o mais utilizado atualmente pois ele atende aplicações dos dispositivos que são mais populares hoje em dia, como celulares e PDAs. Este perfil será mais detalhado adiante.

O Foundation Profile tem dois propósitos. O primeiro seria definir um perfil para dispositivos que possuem suporte a operações de rede, mas não utilizam interface gráfica. E o segundo propósito seria para servir como base de outros perfis com mais funcionalidades.

O Personal Profile é utilizado em dispositivos com total suporte a bibliotecas de interface gráfica ou applets, como os PDAs de alto nível, comunicadores e console de jogos. Este perfil utiliza a biblioteca completa de AWT (Java Abstract Window Toolkit) e possui um grande suporte ao desenvolvimento de aplicações baseadas em applets que são os mesmos que rodam em aplicações Desktop convencionais.

O Personal Basis Profile é utilizado em dispositivos com recursos reduzidos que necessitam de um framework de interface gráfica mais leve, sem todas as funcionalidades do AWT. Ele possui suporte ao modelo de programação baseado em *xlet*, e pode ser utilizado em aplicações de televisão interativa, na indústria automotiva, ou em um mercado consumidor definido.

2.3 Pacotes opcionais

Após a definição da configuração CLDC e do padrão MIDP, observou-se que além das configurações e dos perfis, existiam bibliotecas que não pertenciam a simples categorias ou famílias de dispositivos. Assim, para permitir uma melhor localização de bibliotecas que seriam aproveitadas por um grande número de dispositivos e famílias de dispositivos e que não estavam restritos a uma configuração, foi criado o conceito de pacotes opcionais, que nada mais é que um conjunto de bibliotecas que pode ser utilizado para estender um perfil, com a vantagem de ter suas funcionalidades independentes de qualquer perfil. Espera-se que os pacotes opcionais sirvam como um guia da evolução dos perfis, visto que bibliotecas são desenvolvidas conforme novas tecnologias vão surgindo. A tendência é que tão logo estas bibliotecas fiquem maduras, com o surgimento de novas versões, elas sejam incorporadas aos perfis.

Segue, abaixo, os pacotes opcionais especificados até o momento:

JSR 120: Wireless Messaging API - define um conjunto de bibliotecas que padronizam recursos de comunicação. Utilizam: SMS (Short Message Service) e CBS (Cell Broadcast Service).

JSR 135: Mobile Media API (MMAPI) - define um conjunto de biblioteca de multimídia, providenciando acesso e controle a recursos básicos de áudio, multimídia e arquivos.

JSR 172: J2ME Web Service Specification - define padrões de acesso para acessar Web Services. Definindo uma infra-estrutura para processamento de XML, reuso de conceitos de Web Service no acesso a servidores, permitir interoperabilidade entre clientes J2ME e Web services.

JSR 177: Security and Trust Services for J2ME - Define um conjunto de bibliotecas que definem serviços de segurança aos dispositivos (serviços confiáveis).

JSR 179: Location API for J2ME - Define um conjunto de bibliotecas que permite aos desenvolvedores escrever aplicações que produzam informações sobre a localização física da aplicação, utilizando mecanismos de GPS e E-OTD.

JSR 180: Session Initiation Protocol (SIP) for J2ME - O SIP é utilizado para estabelecer e gerenciar sessões de IP multimídia – o mesmo mecanismo usado em serviços de mensagem instantânea.

JSR 184: Mobile 3D Graphics for J2ME - Define uma biblioteca leve e interativa de gráficos 3D que consume pouco processamento e memória.

JSR 190: Event Tracking API for J2ME - Define um padrão para envio de eventos para um servidor de eventos via um protocolo padrão. Estes eventos podem ser utilizados para pagamento de contas, notificação de atualizações, revisões, etc.

3. A ARQUITETURA DE WEB SERVICES

Existem diversas definições de web services, segundo [HENDRICKS, 2002], um Web service é um pedaço de lógica de negócio localizado em algum lugar da Internet, que é acessível através de protocolos padrões da Internet como o HTTP ou SMTP.

Segundo [RODRIGUES, 2003], web services são aplicações que fazem uso de um registro e padrões de comunicação para trabalhar de uma forma dinâmica (onde uma aplicação fornece transações, mensagens ou serviços para outras aplicações). Estas aplicações utilizam um formato definido (XML ou algum variante de XML) para apresentação de informação e dados; e eles utilizam alguns padrões da arquitetura de Web service para encontrar serviços (UDDI), negociar como enviar e receber informações (WSDL), iniciar sessões de comunicação (SOAP), e transferir informação sobre a Internet (HTTP).

Uma das maiores promessas de web services é o seu potencial para alcançar a interoperabilidade através de

sistemas heterogêneos, plataformas, aplicações e linguagens de programação. As organizações passaram a dar mais atenção à necessidade de interoperabilidade, principalmente nas áreas de Enterprise Application Integration (EAI) e Business-to-Business Integration (B2Bi).

Um dos maiores desafios representados na área de EAI é integração entre aplicações que rodam em diferentes plataformas e servidores web. Estes sistemas precisam se comunicar e trocar informações, inclusive com outros dispositivos como impressoras e fax.

B2Bi representa integrações de negócios entre diferentes sistemas, ou seja, se um negócio for à realização de compra de estoque, os sistemas precisariam interagir e trocar informações. Geralmente esta troca de informação ocorre utilizando diferentes tecnologias. Muitas organizações gostariam de estender isto ao alcance de seus usuários, mas para que isto ocorra, os web services e os navegadores web precisam tornar-se disponíveis para a maioria das plataformas, tornando a interoperabilidade uma realidade.

A seguir, serão detalhadas todas as tecnologias que utilizadas na arquitetura de web services, que agora, segundo Hendricks[HENDRICKS, 2002], pode ser definida como:

- uma tecnologia que se comunica via protocolos abertos (HTTP, SMTP, etc);
- processa mensagens XML utilizando SOAP;
- descreve suas mensagens utilizando XML Schema;
- permite a descrição de um serviço utilizando WSDL;
- pode ser descoberto utilizando UDDI

3.1 XML

O XML - Extensible Markup Language - é uma linguagem de marcação de dados, que provê um formato para descrever dados estruturados, facilitando a declaração mais precisa de conteúdos e resultados mais significativos de busca através de múltiplas plataformas.

O XML foi inicialmente projetado como um formato de dados para facilitar o recebimento, o processamento e a geração de informações na Web de uma maneira simplificada, assim como o HTML é utilizado para descrever a forma como dados devem ser exibidos para o usuário. Enquanto no HTML utiliza-se tags (markup tags) pré-definidas para visualizar uma página, no XML, estas as tags são definidas conforme a sua necessidade, da maneira mais coesa para representar um dado.

3.2 HTTP

O HTTP (HyperText Transfer Protocol) é o protocolo de nível de aplicação responsável pelo funcionamento da World Wide Web. Torna possível a transmissão dos documentos hipermídias, como textos, gráficos, som, imagem e outros recursos disponíveis na Web para posterior visualização na máquina cliente.

Apesar da aparente complexidade da Web, o protocolo HTTP é relativamente simples. As especificações do protocolo HTTP são úteis para todos que trabalham ativamente na criação de sites Web, manutenção de servidores Web, ou com desenvolvimento de programas-clientes e servidores que venham a interagir com a Web.

O protocolo HTTP é baseado no modelo pedido/resposta (request/response), onde um cliente estabelece uma comunicação com o servidor e envia um pedido. O servidor analisa o pedido, e devolve uma resposta. A conexão deve ser estabelecida antes de cada pedido de cliente e encerrada após a resposta.

Um pedido é uma mensagem enviada de um cliente ao servidor com o método que será aplicado ao recurso, o identificador do recurso, e a versão do protocolo em uso.

3.3 SOAP

Segundo [**Erro! A origem da referência não foi encontrada.**], SOAP é um protocolo superficial que exporta informação de forma centralizada em um ambiente distribuído. Ele é um protocolo baseado em XML que consiste de três partes: um envelope que define um framework para descrever o que é uma mensagem e como processá-la; um conjunto de regras de codificação para expressar instâncias de tipos definidos pela aplicação; e uma convenção para representar chamadas remotas de procedimento e respostas.

O SOAP não está vinculado a uma plataforma de hardware, a um sistema operacional, linguagem de programação ou hardware de rede. Diferentemente de outros sistemas de computação distribuídos, o SOAP é construído no topo de padrões abertos, como HTTP e XML. A utilização de padrões amplamente aceitos e conhecidos facilita o aprendizado, por parte dos desenvolvedores, da nova tecnologia, e também da infra-estrutura existente para suportá-la.

O protocolo SOAP é considerado superficial, pois contém menos recursos do que outros protocolos de computação distribuídos, o que torna o protocolo menos complexo. Além disso, o SOAP não define um mecanismo para a descrição e localização de objetos em sistemas distribuídos. Esses mecanismos são definidos pelas especificações WSDL e UDDI, que serão discutidas posteriormente.

O SOAP proporciona várias vantagens em relação a outros sistemas distribuídos, entre as quais podemos citar:

- O SOAP pode atravessar firewalls com facilidade;
- Os dados do SOAP são estruturados usando XML;
- O SOAP pode ser usado em combinação com vários protocolos de transporte, como HTTP, SMTP e JMS.
- O SOAP mapeia satisfatoriamente para o padrão de solicitação/resposta (request/response) HTTP e do HTTP Extension Framework.
- O SOAP é razoavelmente superficial como um protocolo, pois contém menos recursos do que outros protocolos de computação distribuídos.
- Existe suporte para SOAP por parte de vários fornecedores

Porém, devido à falta de recursos do protocolo SOAP, ele pode apresentar algumas desvantagens:

- Falta de interoperabilidade entre kits de ferramentas do SOAP;
- Mecanismos de segurança são imaturos;
- Não existe garantia quanto à entrega da mensagem;

O SOAP utiliza XML para codificar todas as mensagens. Devido a sua simplicidade, as mensagens SOAP não podem conter um DTD. Ela deve incluir os *namespaces* característicos de XML em todos os elementos e atributos definidos pelo SOAP. A vantagem de utilizar *namespaces* no SOAP é que os elementos definidos não entrarão em conflito com os elementos padrão do SOAP. Sendo assim, todos os elementos e atributos padrão do SOAP recebem um prefixo com o identificador do *namespace SOAP-ENV*, que está associado ao *namespace* http://schemas.xmlsoap.org/soap/envelope.

3.4 WSDL

Segundo [CHRISTENSEN, 2001], WSDL é um formato XML para descrever serviços de rede como um conjunto de operações nas mensagens contidas em documentos ou procedimentos orientados a informação.

Um documento WSDL representa a interface externa de um serviço Web. Entretanto, além de descrever a interface apresentada, um documento WSDL também contém a localização do serviço. O registro do serviço está disponível em um local previsível e bem conhecido na rede.

Para permitir maior flexibilidade, a WSDL apresenta a definição de um serviço Web em duas partes: a primeira é responsável por representar uma definição abstrata independente do protocolo de transporte de alto-nível de um serviço Web, enquanto a segunda representa uma descrição específica do protocolo utilizado na camada de transporte de rede.

3.5 UDDI

Segundo [BELLWOOD, 2002], Universal Description, Discovery and Integration, ou UDDI, é o nome do grupo de registros baseados na web que expõem informações sobre negócios ou outras entidades e suas interfaces ou APIs. Estes registros funcionam sobre múltiplos sites operadores, que podem ser utilizados por qualquer um que gostaria de disponibilizar alguma informação sobre seus negócios ou entidades.

O conceito relacionado com o UDDI provém do fato que a tecnologia dos serviços Web lida com a comunicação de aplicações, sem que seja necessária uma grande intervenção humana. Todos devem ter condições de usar qualquer serviço, independente de plataforma e linguagem de programação. E o principal conceito que dá embasamento a esse ponto é o do SOA (Service Oriented Architecture), que se define em três papéis:

- requisitante do serviço;
- provedor do serviço;
- agente do serviço (broker);

4. J2ME WEB SERVICES API

A natureza de um Web Service de oferecer funcionalidades distribuídas sobre a Internet independente de sistema operacional e linguagem de programação já era muito explorada através de aplicações convencionais, utilizando-se J2SE e J2EE. Entretanto, com o surgimento de J2ME, criou-se uma nova oportunidade de mercado, onde os serviços que antes eram acessados através de computadores desktop, agora podem ser acessados de qualquer lugar.

Através do uso de tecnologias como SOAP, WSDL, XML e RPC, os desenvolvedores de dispositivos móveis podem criar aplicações clientes que são distribuídas, portáveis e que podem ser utilizadas em telefones celulares, PDAs e carregadas pelos seus clientes para qualquer lugar.

A especificação J2ME Web Services[ELLIS, 2004] (JSR-172 ou WSA) foi especialmente projetada para facilitar o desenvolvimento de pequenas, rápidas e robustas aplicações do mercado wireless. Esta especificação fornece dois pacotes opcionais baseados em XML que podem ser utilizados em dispositivos com pouca memória, como telefones celulares, handhelds, PDAs e pagers.

4.1 Especificação JSR-172

O grande interesse e atividade da comunidade Java no uso de padrões de web services para construção de serviços, fez surgir a necessidade da utilização destes padrões no desenvolvimento de clientes J2ME no acesso a serviços corporativos. Ao comparar a arquitetura de Web Services de J2ME observa-se que ela segue as mesmas especificações, arquiteturas e modelos de invocação padrão de Web Services, WS-I Basic Profile 1.0. Ou seja, ela especifica:

- SOAP 1.1: que define como um dado é transportado e codificado.
- WSDL 1.1 : que define como descrever serviços remotos.
- XML 1.0, e
- XML Schema.

Deve-se observar que a JSR-172 não suporta UDDI 2.0, que define como descobrir um serviço web. Esta também não permite que uma aplicação funcione como produtora de serviço (Endpoint), mas somente como consumidora de serviços.

Segundo [ELLIS, 2004], a infra-estrutura desta especificação baseia-se em dois Pacotes Opcionais[COURTNEY, 2002] que oferecem as seguintes propostas:

- processamento básico de XML;
- reuso de conceitos de web service para utilização de serviços corporativos através de cliente J2ME;
- oferecer bibliotecas e convenções de código para programar estes dispositivos;
- aderir a padrões de web services e convenções que estão sendo consolidados pela comunidade de desenvolvedores Java;
- permitir a interoperabilidade de clientes J2ME com web services;

- permitir um modelo de comunicação do cliente J2ME com um web service consistente com outras tecnologias clientes Java, como o J2SE.

E para isso foram oferecidas as seguintes tecnologias:

- biblioteca de classes para manipulação de dados XML (parsing);

- biblioteca de classes para permitir a comunicação entre mecanismos RPC baseados em XML.

A manipulação dos documentos XML é oferecida através de um subconjunto da biblioteca JAXP 1.2 (JSR-063)[MORDANI, 2002] , e a comunicação através de um subconjunto da biblioteca JAX-RPC 1.1, que serão descritos no próximo tópico.

5. CONCLUSÕES

A arquitetura de Web Services está se difundindo cada vez mais. A possibilidade de criar serviços que podem ser distribuídos pela Internet sem a necessidade de conhecer a sua implementação é o principal aspecto positivo desta arquitetura. Os consumidores destes serviços necessitam conhecer somente a interface do serviço. Assim, os web services podem ser descritos como componentes, que são acessados através de protocolos Web amplamente difundidos, como o HTTP, e que utilizam formatos universais para a representação de dados (XML). A arquitetura de web services não define um conjunto de tecnologias, mas padrões para transporte, descrição e descoberta de serviços, que são implementados utilizando SOAP, WSDL e UDDI, respectivamente.

A arquitetura J2ME fez o Java retornar aos seus princípios, ou seja, ser utilizado em dispositivos portáteis. Esta característica fez o Java crescer muito no mercado abrindo novas portas para esta tecnologia.

Neste trabalho foi demonstrado como é possível desenvolver clientes de web services para dispositivos móveis com suporte para J2ME. Isto foi conseguido através do desenvolvimento de uma aplicação cliente J2ME que acessa serviços oferecidos por um Web Service. Esta aplicação foi implementada utilizando a configuração CLDC de J2ME, e o perfil MIDP 2.0, que atendem aplicações dos dispositivos com poucos recursos de processamento, memória, e conexão intermitente de rede, e abrangem os dispositivos mais populares hoje em dia, como celulares e PDAs.

Para que esta aplicação se comunicasse com um web service, foi necessário utilizar o pacote opcional de J2ME para web services, o WSA. Este pacote oferece suporte a arquitetura básica de web services (WS-I Basic Profile 1.0). Então, para a construção do web service foi necessário restringir-se a esta especificação, ou seja, chegamos à conclusão que uma aplicação J2ME não é compatível com todos os web services, o que não pode ser considerado uma falha desta especificação, mas sim uma condição existente graças à limitação de recursos de hardware dos dispositivos.

Este pacote opcional deve ser instalado no kit de ferramentas de desenvolvimento de J2ME (WTK) e oferece, além da biblioteca de classes, diversas ferramentas para o desenvolvimento de clientes de web services, como um gerador de classes Stub - baseados em arquivos WSDL - que são utilizadas para representar o serviço do web service na plataforma cliente.

A grande dificuldade para o desenvolvimento deste trabalho foi a especificação do serviço, pois a partir desta especificação foi possível criar o arquivo WSDL que serviu de base para a comunicação entre o cliente e o servidor do web service.

O desenvolvimento de uma aplicação J2ME possui características diferentes de uma aplicação desktop convencional, pois é necessário cuidar de aspectos de desempenho e de interfaceamento com o usuário que não são necessários em uma aplicação convencional que não possui tais requisitos.

Portanto, com este trabalho, conseguiu-se comprovar a possibilidade de desenvolver aplicações que interagem com web services através da comunicação por redes sem fio, e com capacidade de hardware limitada, utilizando as tecnologias descritas neste trabalho.

Ainda deve-se observar que existem muitas áreas a serem exploradas no âmbito destas tecnologias. A necessidade de construir aplicações seguras é um desafio para todas empresas, clientes e desenvolvedores, e o estudo de novas técnicas de segurança para web services e J2ME são temas para trabalhos futuros, além do estudo de Web services com outras configurações J2ME. Outra possibilidade de trabalho futuro consiste no desenvolvimento de um suporte para disponibilização de web services, e não somente seus clientes, em dispositivos móveis.

6. AGRADECIMENTOS

Agradeço primeiramente aos meus pais, avós e irmãos, que são as pessoas mais importantes na minha vida e que sempre me apoiaram durante toda esta etapa.

Agradeço aos meus familiares, amigos, todos que me ajudaram durante este trabalho.

7. REFERÊNCIAS

[1] AHMAD, Saqib. Security and Trust Services API (SATSA) for Java 2 Plataform, Micro Edition. Version 1.0, Specification – 28/07/2004. <http://jcp.org/en/jsr/detail?id=177>. Acessado em 19/05/2005.

[2] ALMEIDA, Leandro Batista de. Introdução à J2ME e programação MIDP. Brasil, Revista Mundo JAVA - Número 5. 2004.

[3] BELLWOOD, Tom. UDDI Version 2.04 API Specification – 19/07/2002 <http://uddi.org/pubs/ProgrammersAPI_v2.htm>. Acessado em 28/11/2004.

[4] BOOTH, David. HAAS, Hugo. MCCABE, Francis. NEWCOMER, Eric. CHAMPION, Michael. FERRIS, Chris. ORCHARD, David. Web Service Architecture – W3C Working Group Note 11/02/2004.

<http://www.w3.org/TR/ws-arch>. Acessado em 14/03/2005.

[5] BOX, Don. EHNEBUSKE, David. KAKIVAYA, Gopal. LAYMAN, Andrew. MENDELSOHN, Noah. NIELSEN, Henrik Frystyk. THATTE Satish. WINER, Dave. Simple Object Access Protocol (SOAP) 1.1. – 08/05/2000 <http://www.w3.org/TR/SOAP>. Acessado em 28/11/2004.

[6] CHAPPELL, David. JEWELL, Tyler. Java Web Services – Using Java in Service-Oriented Architectures. O'Reilly - Edição 1 - 03/2002.

[7] CHRISTENSEN, Erik. CURBERA, Francisco. MEREDITH, Greg. WEERAWARANA, Sanjiva. Web Services Description Language (WSDL) 1.1 . – 15/03/2001 <http://www.w3.org/TR/wsdl.html>. Acessado em 28/11/2004.

[8] CONSORTIUM, World Wide Web – Document Object Model (DOM) Level 1 Specification – 01/10/1998. <http://www.w3.org/TR/REC-DOM-Level-1/> Acessado em 12/06/2005.

[9] CONSORTIUM, World Wide Web – XML Path Language (Xpath) Version 1.0 – 16/11/1999. <http://www.w3.org/TR/xpath/> Acessado em 12/06/2005.

[10] CONSORTIUM, World Wide Web – XML Schema Part 0: Primer Second Edition – 28/10/2004. <http://www.w3.org/TR/xmlschema-0/> Acessado em 12/06/2005.

[11] CONSORTIUM, World Wide Web – XSL Transformations (XSLT). – 16/11/1999. <http://www.w3.org/TR/xslt> Acessado em 12/06/2005.

[12] COURTNEY, Jon. JSR 68: J2ME Platform Specification – 09/07/2002. <http://jcp.org/en/jsr/detail?id=68 >. Acessado em 14/05/2005.

[13] EASTLAKE, Donald E., NILES, Kitty. Secure XML: The New Syntax for Signatures and Encryption – Addison Wesley – 19 Julho 2002.

[14] ELLIS, Jon. YOUNG, Mark. JSR 172: J2ME Web Services Specification – 02/03/2004. < http://jcp.org/en/jsr/detail?id=172 >. Acessado em 14/05/2005.

[15] FIELDING, R.; GETTYS, J.; MOGUL, J.; FRYSTYK, H.; MASINTER, L.; LEACH, P.; BERNERS-LEE, T.; Hypertext Transfer Protocol -- HTTP/1.1 – Junho 1999 <http://www.ietf.org/rfc/rfc2616>. Acessado em 28/11/2004.

[16] FOUNDATION, The Apache Software. Apache Ant. <http://ant.apache.org > Acessado em 20/05/2005.

[17] FOUNDATION, The Apache Software. Web Services – Axis. <http://ws.apache.org/axis/overview.html> Acessado em 20/05/2005.

[18] HENDRICKS, Mack. GALBRAITH, Ben. IRANI, Romin. MILBERNY, James. MODY, Tarak. TOST, Andre. BASHA, Jeelani S. . CABLE, Scott. Profissional Java Web Services – Alta Books – 2002.

[19] INC, Sun Microsystems. SATSA Developer's Guide. SATSA Reference Implementation – 12/2004. <http://java.sun.com/j2me/docs/satsa-dg/>. Acessado em 19/05/2005.

[20] KNUDSEN, Jonathan. Wireless Developing with J2ME, Second Edition – Apress – 2003.

[21] MICROSOFT. Criando Web Services Seguros – 20/05/2004. <http://www.microsoft.com/brasil/security/guidance/topics/devsec/secmod85.mspx> Acessado em 15/05/2005

[22] MORDANI, Rajiv. JSR 63: Java API for XML Processing 1.1 – 10/09/2002. <http://jcp.org/en/jsr/detail?id=63 >. Acessado em 14/05/2005.

[23] MUCHOW, John W. Core J2ME – Tecnologia & MIDP – Pearson Makron Books – 2004.

[24] NETWORK WORKING GROUP. Hypertext Transfer Protocol – HTTP/1.1 – 06/1999. <http://www.w3.org/Protocols/rfc2616/rfc2616.html>. Acessado em 15/05/2005.

[25] ORTIZ, C. Henrique. Introduction to J2ME Web Services – 04/2004. <http://developers.sun.com/techtopics/mobility/apis/articles/wsa/>. Acessado em 17/05/2005.

[26] QUIN, Lean. Extensible Markup Language (XML) – 07/04/2005 <http://www.w3.org/XML/>. Acessado em 15/05/2005.

[27] RIGS, Roger. TAIVALSAARI, Antero. PEURSEN, Jim Van. HUOPANIEMI, Jyri. PATEL, Mark. UOTILA, Aleksi. EDITOR, Jim Holliday. Programming Wireless Devices with the Java 2 Plataform, Micro Edition, Second Edition – Addison Wesley – 2003.

[28] RODRIGUES, Nando. Desenvolvedoras Investem em integração de soluções. Revista Computerworld - Edição 393 – 25/09/2003 < http://computerworld.uol.com.br/AdPortalv5/adCmsDocumentShow.aspx?DocumentID=76278>. Acessado em 02/11/2004.

[29] SKONNARD, Aaron. The Birth of Web Services. – 10/2002. <http://msdn.microsoft.com/webservices/understanding/webservicebasics/default.aspx?pull=/msdnmag/issues/02/10/xmlfiles/default.aspx>. Acessado em 24/11/2004