DEDICATÓRIA

AGRADECIMENTO

SUMÁRIO

LISTAS DE FIGURAS

LISTAS DE TABELAS

LISTAS DE ABREVIATURAS

AWT - *Abstract Windowing Toolkit*

CDC - *Connected Device Configuration*

CLDC - *Connected Limited Device Configuration*

CSS - *Cascading Style Sheets*

DOM - *Document Object Model*

DTD - *Document Type Definition*

J2ME - *Java 2 Micro Edition*

JCP - *Java Community Process*

JSR - *Java Specification Request*

OASIS - *Organization for the Advancement of Structured Information Standards*

PBP - *Personal Basic Profile*

RMI - *Remote Method Invocation*

RMS - *Record Management System*

RPC - *Remote Procedure Call*

SGML - *Standard Generalized Markup Language*

SOAP - *Simple Object Access Protocol*

SPI - *Service Provider Interface*

UBR - *UDDI Business Registry*

UDDI - *Universal Description Discovery and Integration*

W3C - *World Wide Web Consortium*

WSA - *Web Services API's*

WSDL - *Web Services Descriptor Language*

WS-I - *Web Service Interoperability Organization*

WTK - *Wireless Toolkit*

XLS - *Extensible Stylesheet Language*

XML - *Extensible Markup Language*

XPointer - *XML Pointer Language*

XSLT - *Extensible Stylesheet Language Transformations*

RESUMO

Atualmente com a explosão de vendas na telefonia móvel, o desenvolvimento de sistemas que são suportados por esses dispositivos é cada vez mais incentivado pelas operadoras. O desenvolvimento de sistemas que permite uma independência de plataforma e de fabricante, e com a possibilidade de comunicação entre outras aplicações, motivou o estudo da arquitetura J2ME.

Web Services permite uma comunicação e interação entre aplicações pela internet ou por redes corporativas. Com o uso de padrões independentes de plataforma, possibilita a comunicação entre aplicações desenvolvidas em diferentes linguagens, transformando assim um sistema em um provedor de serviços. Portanto, a capacidade de desenvolvimentos de sistemas, independente de plataforma para dispositivos móveis, e a sua interação com serviços Web, independente de linguagem, é uma grande oportunidade de negócio. Neste trabalho serão explicados os conceitos relacionados a J2ME e Web services. E para demonstrar a integração entre essas duas tecnologias, será desenvolvida uma aplicação exemplo, que tomou como base uma empresa de laticínios, que possibilitará a automatização da sua força de vendas.

ABSTRACT

1 INTRODUÇÃO

O paradigma de *Web Services* é uma tecnologia promissora para desenvolver aplicações abertas, distribuídas e heterogêneas. A proliferação desta nova tecnologia tem coincidido com avanços significantes na capacidade de hardware e software dos dispositivos móveis. Lidar com os grandes benefícios que vêm com a tecnologia de *Web Services* , tal como interoperabilidade, descoberta de serviço dinâmico e usabilidade é um desafio interessante.

Web Services constituem um novo modelo para o uso na *web*, que permite a publicação de funções de negócios e acesso universal. Tanto desenvolvedores como os usuários finais podem desfrutar dos benefícios providos pelo modelo de *Web Services* , não somente no desenvolvimento e na operação de uma aplicação de negócios, mas também na interoperação desta com outras aplicações. Por outro lado, o acesso do usuário final é facilitado através da provisão de uma interface intuitiva baseada no *browser*, sendo ainda possível a integração do *Web Services* com outras aplicações.

1.1 Motivação

Dados os avanços no panorama tecnológico ocorridos nos últimos anos, tanto no que se refere à disponibilização de serviços baseados na Web quanto na disseminação de dispositivos móveis, é razoável supor que o uso de Web Services a partir de móveis logo viesse a ocorrer. logo viesse a ocorrer. Este casamento entre *Web Services* e dispositivos móveis abre a possibilidade de prover serviços a

usuários destes dispositivos, não importando onde o serviço requisitante e o provedor estejam localizados. A descoberta dinâmica e a invocação de serviços que a tecnologia de Web Services possibilita apresenta dificuldades no caso de aplicações móveis, onde o contexto do usuário pode mudar dinamicamente, provendo diferentes serviços ou implementações de serviços apropriadas em diferentes momentos. Ao final do trabalho com a aplicação exemplo, poderemos ver como empregar essas tecnologias e demonstrar como transpor obstáculos que o emprego de dispositivos móveis nos obriga a enfrentar.

1.2 Objetivos

Com o intuito de investigar o emprego de Web Services em dispositivos móveis, estaremos desenvolvendo neste projeto uma aplicação exemplo que consiste em um Web Services que disponibiliza serviços de pedido a uma empresa. A aplicação será disponibilizada em um servidor, no qual será executado um Web Services que contém regras de negócio para pedidos de uma empresa de laticínios. Na outra ponta, estará os dispositivos móveis com a aplicação que irá consumir esses serviços e que terá uma interface simplificada para fazer consultas a pedidos de clientes da empresa.

1.3 Organização do texto

A fazer

2 DISPOSITIVOS MÓVEIS

2.1 J2ME

Efetuar ligações telefônicas e servirem de agendas. Alguns ainda vêem os celulares e dispositivos móveis com apenas estas funcionalidades. Porém, nos últimos anos, diversos serviços foram incorporados a estes aparelhos, permitindo aos seus usuários acompanhar as ações do mercado financeiro, verificar suas caixas de correio eletrônico, trocar informações com seus escritórios quando estão em visita a clientes, divertir-se com jogos ou músicas, entre outras coisas.

Com a grande demanda para o desenvolvimento de novas aplicações e cada vez mais complexas, as empresas que fabricavam os aparelhos deixaram de ser as únicas responsáveis por produzir os aplicativos contidos em seus aparelhos. Estas empresas precisaram encontrar um meio de permitir que outras empresas pudessem desenvolver estas aplicações. Só que esta situação envolvia um problema com relação às informações que seriam distribuídas para estes desenvolvedores, pois não era interessante que eles tivessem conhecimento da arquitetura interna dos aparelhos nem de algumas funcionalidades dos sistemas operacionais dos mesmos, garantindo o sigilo de soluções proprietárias.

Os fabricantes, por conta dessa necessidade, decidiram construir uma camada entre os sistemas operacionais de seus aparelhos e as aplicações. Camada esta, que seria usada por outras empresas para acessar as funcionalidades nativas dos aparelhos.

Java 2 *Micro Edition* (J2ME), foi a primeira iniciativa neste sentido, sendo uma versão da linguagem Java para dispositivos móveis. Assim sendo, só seria necessário que os fabricantes implementassem uma JVM em seus sistemas operacionais, assim como acontece com os navegadores de Internet e os *applets* Java. Com esta solução adotada, permitiu-se em muito menos tempo, o desenvolvimento de um número muito maior de aplicações, devido a grande disponibilidade de programadores que conheciam a linguagem Java (versão Standard) e que poderiam migrar, facilmente, para J2ME. Aplicações portáveis e de menor custo de desenvolvimento foram algumas das conseqüências dessa medida.

A plataforma J2ME é a plataforma JAVA para dispositivos compactos, como celulares, PDA's, controles remotos, e uma outra gama de dispositivos. Assim como a versão *Enterprise* (J2EE), a *Standard* (J2SE) e a *Smart Card* (Java *Card*), a plataforma J2ME é uma coleção de API's do *JAVA* definidas através da JCP (Java *Community Proccess*). A plataforma leva ao consumidor o poder e os benefícios da tecnologia JAVA em seus aparelhos, incluindo uma interface flexível, um modelo robusto de segurança e suporte a aplicações em rede ou offline. Com o J2ME, as aplicações são escritas somente uma vez para um grande número de dispositivos e são baixadas dinamicamente, dentre outros benefícios.

2.1.1 A arquitetura J2ME

A arquitetura J2ME define configurações, que especifica a JVM e algumas api's básicas, perfis, que adiciona api's específicas para uma determinada família de dispositivos e pacotes opcionais como elementos para construir ambientes de

execução completos que preenchem os requisitos para um bom número de dispositivos e mercados-alvo. Cada combinação é otimizada para a memória, capacidade de processamento e de I/O de uma determinada categoria de dispositivos.

O resultado é uma plataforma comum que é compatível com a maioria dos dispositivos móveis do mercado.

Apesar de os dispositivos de comunicação como celulares, *pagers* e outros dispositivos conectados como PDA's e minicomputadores possuírem várias características em comum, eles diferem bastante em relação à forma, às funções e aos componentes internos. Isso exige que eles sejam categorizados e agrupados de maneira que a compatibilidade seja garantida apenas entre os equipamentos semelhantes.

solução adotada para a plataforma Java foi categorizá-la em duas camadas chamadas "configuração" e "perfil".

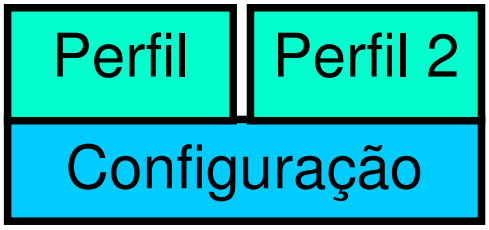

Figura 2.1: Perfil e configurações.

Há ainda a apresentação de uma terceira camada, representando diferentes pacotes opcionais padronizados na plataforma J2ME, o que garante a flexibilidade e a extensibilidade da plataforma.

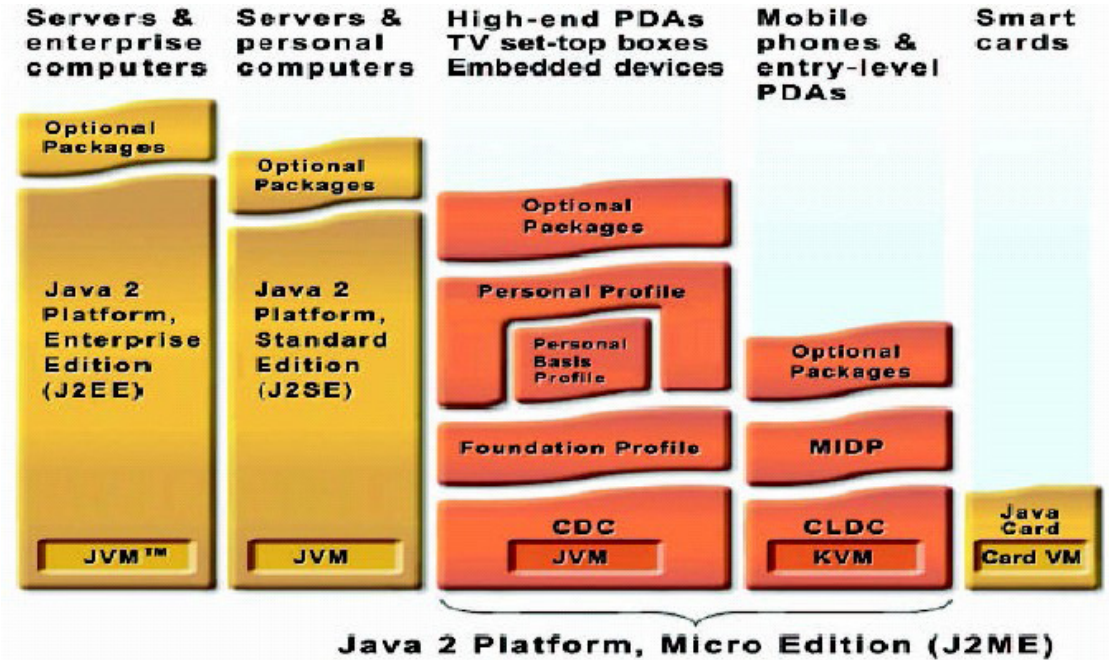

Figura 2.2: Edições da plataforma Java.
Fonte: RIGS, (2003).

2.1.1.1 Configurações

A configuração especifica a máquina virtual JVM e alguns conjuntos de API's básicas que deverão ser utilizados para uma família particular de dispositivos. Atualmente, existem basicamente dois tipos de configurações: *Connected Device Configuration* (CDC – Configuração para Dispositivos Conectados) e *Connected Limited Device Configuration* (CLDC – Configuração para dispositivos conectados limitados).

2.1.1.2 Resumo das configurações existentes

JSR	Nome
30	Connected Limited Device Configuration(CLDC) 1.0
36	Connected Device Configuration
139	Connected Limited Device Configuration(CLDC) 1.1

Tabela 2.1: Configurações.

Fonte: JCP, (2005).

2.1.1.3 *Connected Device Configuration*

Um dispositivo para suporte ao CDC devem ter no mínimo 512 KB de memória para sistema (ROM), 256 KB de memória de acesso randômico para leitura e escrita (RAM), e algum tipo de conexão de rede. O CDC é destinado a dispositivos com recursos mais avançados de processamento e armazenamento, podendo incluir alguns PDA's. De forma geral, o CDC especifica que uma JVM completa deve ser suportada, assim como definida nas especificações da plataforma Java 2.

As configurações e os perfis da J2ME são genericamente descritos baseados na capacidade de memória, sendo especificadas as quantidades mínimas de ROM e RAM, sendo que essas especificações do CDC são desenvolvidas sobre o Java *Community Process* (JCP).

2.1.1.4 *Connected Limited Device Configuration*

A CLDC é a configuração que engloba telefones celulares, *pagers*, PDA's e outros dispositivos semelhantes. A CLDC está voltada a dispositivos menores que os CDC, cuja limitação se refere aos recursos de memória, processamento e, principalmente, da conectividade com redes e Internet.

A CLDC foi criada considerando dispositivos com 128 KB a 512 KB de memória disponível para a plataforma Java. Quem já viu uma máquina virtual Java ocupando dezenas de megabytes nos computadores tipo desktop pode imaginar o desafio para a execução de algo semelhante, restringindo-se enormemente essa quantidade de memória, como é realizado no J2ME.

A conexão limitada refere-se a uma rede lenta e intermitente. Na maioria dos telefones celulares, por exemplo, a velocidade alcançada é baixa se comparada a conexões a redes locais. Lembrando que estamos tratando de dispositivos com tela reduzida, memória limitada e conexão lenta, aplicações desenvolvidas para CLDC devem ser cautelosas no uso da rede.

A CLDC é baseada em uma JVM reduzida, conhecida como KVM, assim designada pelo fato de seu tamanho ser medido em kilobytes em vez de megabytes. Enquanto a CLDC é um documento de especificação, a KVM refere-se a uma parte de um software específico.

Por se tratar de uma máquina virtual de tamanho reduzido, a KVM não pode fazer tudo que a JVM faz no mundo J2SE. Por exemplo:

a) Códigos nativos não podem ser associados em tempo de execução, pois todas as funcionalidades nativas são compiladas junto à própria KVM;
b) A KVM inclui apenas um subconjunto dos verificadores de bytecode padronizados. Isso significa que a tarefa de verificar as classes é dividida entre o dispositivo CLDC e algum mecanismo externo, o que causa sérios problemas de segurança.

Na prática, a questão de verificação passará como uma nova tarefa no processamento de desenvolvimento do aplicativo, sendo uma etapa a ser seguida após a compilação do software.

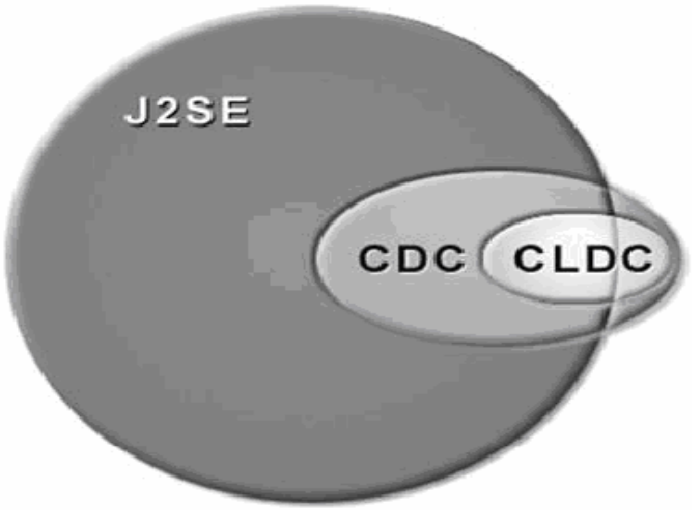

Figura 2.3: Comparativo entre J2SE, CDC e CLDC.
Fonte: RIGS, (2003).

2.1.1.5 Perfis

O perfil está na camada superior à de uma configuração, adicionando API's e especificações necessárias para o desenvolvimento de aplicações para uma determinada família de dispositivos.

2.1.1.6 Perfis atuais

Vários perfis diferentes estão sendo desenvolvidos sobre o JCP, são eles:

JSR	Nome
37	*Mobile Information Device Profile (MIDP) 1.0*
46	*Foundation Profile*
62	*Personal Profile*
75	*PDA Profile 1.0*
118	*Mobile Information Device Profile (MIDP) 2.0*
129	*Personal Basis Profile*

Tabela 2.2: Tabela dos principais perfis.
Fonte: JCP, (2005).

O *Foundation Profile* é uma especificação para dispositivos que podem suportar ambientes J2ME com conexões de rede. Esse perfil por si só não suporta interface com o usuário. Dessa forma, outros perfis podem pertencer a uma camada superior ao *Foundation Profile* para permitir e adicionar suporte à interface com o usuário e outras funcionalidades.

Tanto o *Personal Basis Profile* quanto o *Personal Profile* são exemplos de perfis que ficam sobre o *Foundation Profile*, sendo que a combinação CDC + *Foundation Profile* + *Personal Basis Profile* + *Personal Profile* foi desenvolvida como a nova geração do ambiente de execução de aplicação chamado de Personal Java.

O perfil para PDA's, que está sobre o CLDC, é destinado a dispositivos de computadores de mão, tais como *Palmtops* e pocketPCs, com um mínimo de 512 KB de memória ROM e RAM combinadas e um máximo de 16 MB. Esse perfil está sobre o *Mobile Information Device Profile* ("Perfil para Dispositivos Móveis") e o *Personal Profile*, incluindo um modelo de aplicação baseado em MIDlets, mas usando um subconjunto de API's gráficas AWT ("*Abstract Windowing Toolkit*") da plataforma J2SE. Mesmo com a finalização da especificação do perfil PDAP, nenhum fabricante de hardware anunciou sua implementação, sendo que o mundo J2ME atualmente se resume nos perfis MIDP, para pequenos terminais, e *Personal Profile*, para terminais mais avançados.

2.1.1.7 *Mobile Information Device Profile*

O MIDP, de acordo com a sua especificação, tem as seguintes características:

a) 256 KB de memória não volátil para a implementação do MIDP;
b) 128 KB de memória volátil para a máquina virtual;
c) 32 KB de memória volátil para a pilha de execução;
d) 8 KB de memória não volátil para armazenamento persistente de dados;
e) Visor com tamanho mínimo de 96 X 54 pixels;

f) Algum mecanismo de entrada de dados, seja teclado, seja tela sensível ao toque;
g) Conexão de rede de dois sentidos, possivelmente intermitente e com limitada largura de banda;
h) Habilidade de tocar sons, utilizando hardware ou algoritmos de software;

2.1.1.8 Os principais objetivos e vantagens

Os padrões de configuração CLDC e de perfil MIDP visam a proporcionar benefícios da tecnologia Java a dispositivos sem fio com restrições de recurso e conectividade de internet limitada. Dentre as vantagens oferecidas, destacam-se:

a) Distribuição dinâmica de aplicações e conteúdos;
b) Desenvolvimento de aplicações por terceiros;
c) Independência de padrões e tecnologias de rede;
d) Compatibilidade com outros padrões de aplicações sem fio.

Dispositivos Alvos:

Figura 2.4: Dispositivos que suportam J2ME.

O modelo de interação com o usuário dos dispositivos-alvos do CLDC e MIDP variam significantemente. Considerando como os usuários interagem com os atuais aparelhos celulares, apesar da maioria trabalhar no mesmo padrão, há alguns modelos que diferem bastante entre si, podendo ser resumido em três tipos:

a) manipulação com uma mão: é o mais comum, encontrado em celulares comuns;
b) manipulação com duas mãos: é utilizado em alguns aparelhos celulares com teclado estendido, possuindo normalmente teclas alfanuméricas, sendo divididas ou não em dois segmentos entre o visor;
c) uso de tela sensível ao toque: normalmente utilizados em PDA's e *smartphones* com o auxilio de um pointer.

2.1.1.9 Pacotes opcionais

Com 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 são um conjunto de bibliotecas que podem ser utilizados para estender um perfil, com a vantagem de ter suas funcionalidades independentes de qualquer perfil.

São eles:

JSR	Nome	Descrição
82	*Bluetooth* API	Define um conjunto de bibliotecas que possibilitam a comunicação de entre dispositivos via *bluetooth*.
120	*Wireless Messaging* API	Define um conjunto de bibliotecas que padronizam recursos de comunicação. Utilizam SMS (*Short Message Service*) e CBS (*Cell Broadcast Service*).
135	*Mobile Media API* (MMAPI)	Define um conjunto de biblioteca de multimídia, providenciando acesso e controle a recursos básicos de áudio, multimídia e arquivos.
172	J2ME *Web Services Specification*	Define padrões de acesso para acessar *Web Services*. Definindo uma infra-estrutura para processamento de XML, reuso de conceitos de *Web Services*
177	*Security And Trust Services* for J2ME	Conjunto de bibliotecas que definem serviços de segurança aos dispositivos (Serviços confiáveis).
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.
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.
184	*Mobile 3D Graphics for* J2ME	Define uma biblioteca leve e interativa de gráficos 3D que consume pouco processamento e memória.
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.

Tabela 2.3: Tabela dos pacotes opcionais.

Fonte: JCP, (2005).

3 *WEB SERVICES*

3.1 XML

Extensible Markup Language (XML) é uma linguagem de marcação de dados (*meta-markup language*) que provê um formato para descrever dados estruturados. Isso facilita declarações mais precisas do conteúdo e resultados mais significativos de busca através de múltiplas plataformas. O XML permite as aplicações manipularem e visualizarem dados via internet.

O XML permite a definição de um número infinito de tags. Enquanto no HTML as tags podem ser usadas para definir a formatação de caracteres e parágrafos, o XML provê um sistema para criar tags para dados estruturados.

Um elemento XML pode ter dados declarados como sendo preços de venda, taxas de preço, um título de livro, a quantidade de chuva, ou qualquer outro tipo de elemento de dado. Como as tags XML são adotadas por intranets de organizações, e também via Internet, haverá uma correspondente habilidade em manipular e procurar por dados independentemente das aplicações onde os mesmos são encontrados. Uma vez que o dado foi encontrado, ele pode ser distribuído pela rede e apresentado em um navegador de Internet (*browser*) de várias formas possíveis, ou transferido para outras aplicações para processamento futuro e visualização.

3.1.1 Representação estruturada dos dados

O XML provê uma representação estruturada dos dados que mostrou ser amplamentemente implementável e fácil de ser desenvolvida.

Implementações industriais na linguagem SGML (*Standard Generalized Markup Language*) mostraram a qualidade intrínseca e a força industrial do formato estruturado em árvore dos documentos XML.

O XML é um subconjunto do SGML, o qual é otimizado para distribuição através da web, e é definido pelo *Word Wide Web Consortium* (W3C), assegurando que os dados estruturados serão uniformes e independentes de aplicações e fornecedores.

O XML provê um padrão que pode codificar o conteúdo, as semânticas e as esquematizações para uma grande variedade de aplicações, desde simples até as mais complexas, dentre elas:

a) um simples documento;
b) um registro estruturado tal como uma ordem de compra de produtos;
c) um objeto com métodos e dados como objetos Java ou controles ActiveX;
d) um registro de dados, como por exemplo o resultado de uma consulta a bancos de dados;
e) apresentação gráfica, como interface de aplicações de usuário.
f) Entidades e tipos de esquema padrões;
g) todos os links entre informações e pessoas na web.

Uma característica importante é que, uma vez tendo sido recebido o dado pelo cliente, tal dado pode ser manipulado, editado e visualizado sem a necessidade de contatar novamente o servidor. Dessa forma, os servidores têm menor

sobrecarga, reduzindo a necessidade de computação e reduzindo também a requisição de banda passante para as comunicações entre cliente e servidor.

O XML é considerado de grande importância na Internet e em grandes intranets porque provê a capacidade de interoperação dos computadores, por ter um padrão flexível e aberto e independente de dispositivo. As aplicações podem ser construídas e atualizadas mais rapidamente, Além de permitir múltiplas formas de visualização dos dados estruturados.

3.1.2 Separação entre dados e apresentação

A mais importante característica do XML se resume em separar a interface com o usuário (apresentação) dos dados estruturados. O HTML especifica como o documento deve ser apresentado na tela por um navegador. Já o XML define o conteúdo do documento. Por exemplo, em HTML são utilizadas tags para definir tamanho e cor de fonte, assim como formatação de parágrafo. No XML você utiliza as tags para descrever os dados, como por exemplo, tags de assunto, título, autor, conteúdo, referências, datas, etc...

O XML ainda conta com recursos tais como folhas de estilo definidas com *Extensible Style Language* (XSL) e *Cascading Style Sheets* (CSS) para a apresentação de dados em um navegador. O XML separa os dados da apresentação e do processamento, o que permite visualizar e processar o dado como quiser, utilizando diferentes folhas de estilo e aplicações.

Exemplo de aplicação Web em três níveis, a qual é flexível e permite a troca de dados entre a camada de persistência e os clientes (desktops, web, etc).

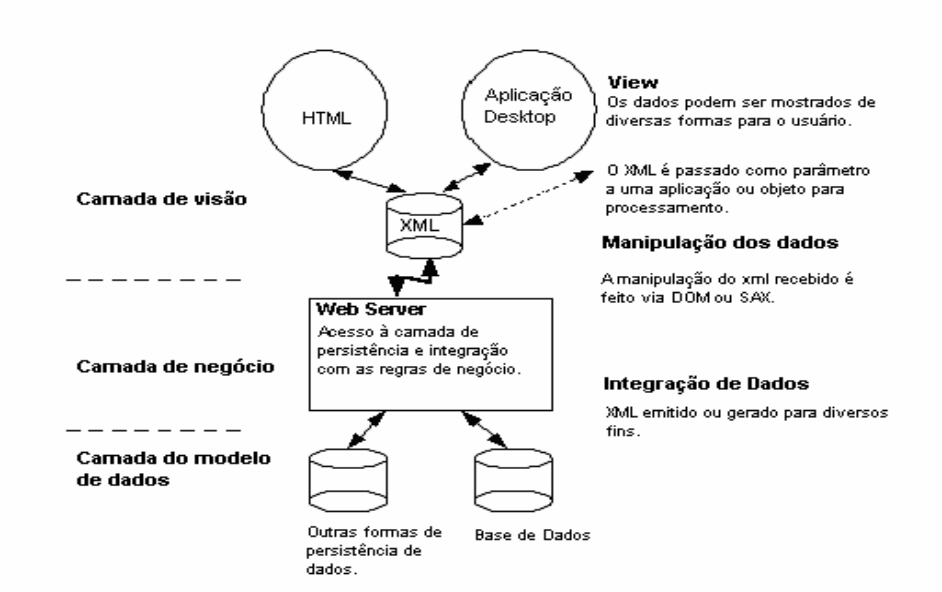

Figura 3.1: Exemplo da aplicação Web em três níveis.
Fonte: DUARTE; JUNIOR, (*Sine die*).

Essa separação dos dados da apresentação permite a integração dos dados de diversas fontes. Informações de consumidores, compras, ordens de compra, resultados de busca, pagamentos, catálogos, etc... podem ser convertidas para XML em uma camada intermediária (*middle-tier*, que pode ser vista como uma espécie de servidor de aplicação), permitindo que os dados sejam trocados online tão facilmente como as páginas HTML mostram dados hoje em dia. Dessa forma, os dados em XML podem ser distribuídos através da rede para os clientes que desejarem.

3.1.3 Estrutura do documento

Um documento XML é uma árvore rotulada onde um nó externo consiste de:

a) dados de caracteres (uma seqüência de texto);
b) instruções de processamento (anotações para os processadores), tipicamente no cabeçalho do documento;
c) um comentário (nunca com semântica acompanhando);
d) uma declaração de entidade (simples macros);
e) nós DTD (*Document Type Declaration*);

Um nó interno é um elemento, o qual é rotulado com:

a) um nome ou;
b) um conjunto de atributos, cada qual consistindo de um nome e um valor. Normalmente, comentários, declarações de entidades e informações DTD não são explicitamente representadas na árvore.

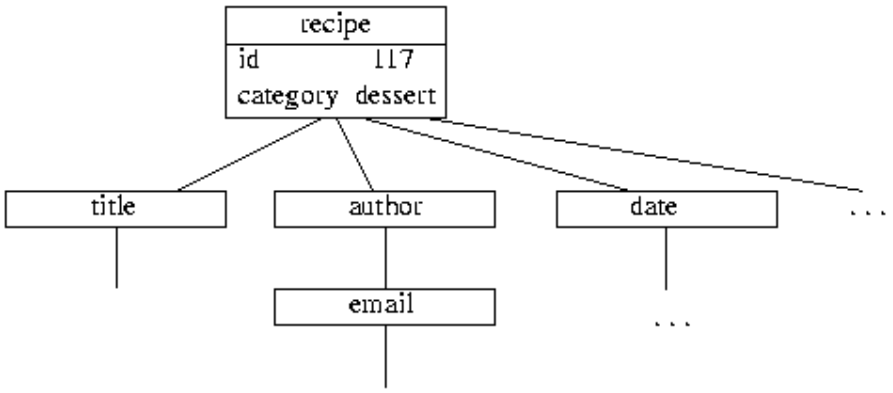

Figura 3.2: Representação XML.
Fonte: DUARTE; JUNIOR, (*Sine die*).

3.1.4 Padrões da estrutura do XML

3.1.4.1 Definição

O XML é baseado em padrões de tecnologia comprovadamente otimizados para a Web.

Os padrões que compõem o XML são definidos pelo W3C (*World Wide Web Consortium*) e são os seguintes:

a) *Extensible Markup Language* (XML): é um padrão mais utilizado pelos desenvolvedores de ferramentas;
b) XML *Namespaces*: é também uma Recomendação, a qual descreve a sintaxe de *namespace*, ou espaço de nomes, e que serve para criar prefixos para os nomes de tags, evitando confusões que possam surgir com nomes iguais para tags que definem dados diferentes;
c) *Document Object Model* (DOM) Level 1 - é uma Recomendação que provê formas de acesso aos dados estruturados utilizando scripts, permitindo aos desenvolvedores interagir e computar tais dados consistentemente;
d) *Extensible Stylesheet Language* (XSL) - O XSL apresenta duas seções: a linguagem de transformação e a formatação de objetos. A linguagem de transformação pode ser usada para transformar documentos XML em algo agradável para ser visto, assim como transformar para documentos HTML, e pode ser usada independentemente da segunda seção (formatação de objetos). O *Cascade Style Sheet* (CSS) pode ser usado para XML simplesmente estruturado mas não pode apresentar informações em uma ordem diferente de como ela foi recebida.

3.1.4.2 Principais benefícios da linguagem XML

O XML tem por objetivo trazer flexibilidade e poder às aplicações Web. Dentre os benefícios para desenvolvedores e usuários temos:

a) Buscas mais eficientes;
b) Desenvolvimento de aplicações Web mais flexíveis. Isso inclui integração de dados de fontes completamente diferentes, de múltiplas aplicações; computação e manipulação local dos dados; múltiplas formas de visualização e atualização granulares do conteúdo;
c) Distribuição dos dados via rede de forma mais comprimida e escalável;
d) Padrões abertos.

3.2. SOAP

O SOAP (*Simple Object Access Protocol*) é um protocolo para a troca de informação em um ambiente descentralizado e distribuído. É um protocolo baseado em XML que consiste em três partes: um envelope, que define um *Framework* para descrever o que contém em uma mensagem e como processá-la, definição de regras para expressar as instâncias dos tipos de dados definidas pela aplicação e uma convenção para representar as chamadas e respostas remotas do procedimento.

SOAP se tornou em um dos mais conhecidos formatos de mensagens e protocolos utilizado por *Web Services* baseado em XML. A sua especificação estabelece um formato padrão de mensagem que consiste em um documento XML capaz de hospedar dados RPC e centrados em documentos. Isto facilita o intercâmbio de dados de modelo síncrono (pedido e resposta) e assíncrono (orientado a processo). Com WSDL estabelecendo um formato de descrição padrão

para aplicações, o uso do formato de mensagem centrados em documento é muito mais comum.

3.2.1 Nós SOAP

Um nó SOAP representa o processamento lógico responsável pela transmissão, recebimento e realização de uma série de tarefas sobre mensagens SOAP. Uma implementação de um nó SOAP é tipicamente específica à plataforma e é normalmente rotulada como um SOAP server ou SOAP listener. Existem também variações especializadas, tais como SOAP routers.

3.2.2 Estrutura das mensagens SOAP

O recipiente da informação de uma mensagem SOAP é chamado de envelope SOAP. A figura abaixo, descreve a estrutura de uma típica mensagem SOAP. O elemento raiz *envelope,* que delimita o documento da mensagem, é composta por uma seção *Body* e uma área *Header* que é opcional.

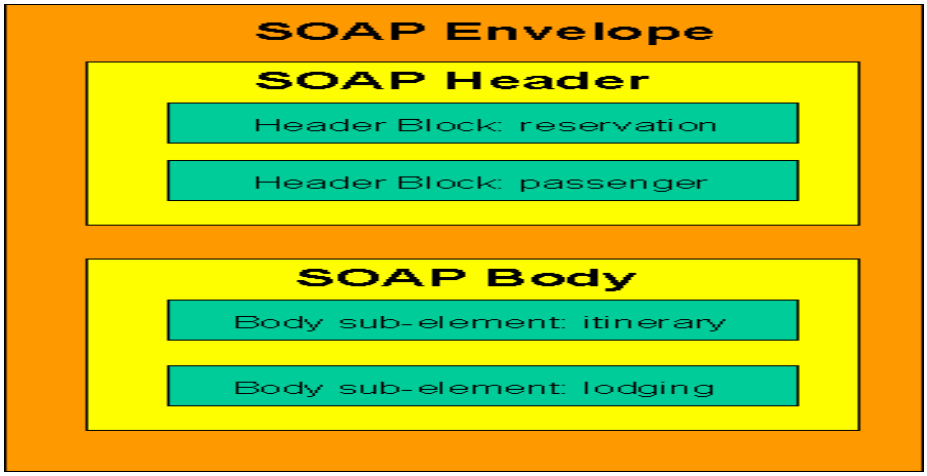

Figura 3.3: Representação da mensagem SOAP.
Fonte: MITRA, (2003).

O cabeçalho SOAP é representado através do uso de um construtor *Header*, o qual pode conter um ou mais blocos ou seções. O uso mais comum de blocos *Header* ocorre em:

a) implementações de extensões SOAP;
b) identificação de alvos SOAP intermediários;
c) fornecimento de meta-informação adicional sobre a mensagem SOAP.

Enquanto uma mensagem SOAP trafega até o seu destino, intermediários podem acrescentar, remover ou processar as informações contidas no *Header*. Mesmo sendo parte opcional de uma mensagem SOAP, o uso da seção *Header* para carregar informações é muito comum quando se trabalha com especificações Web Services de segunda geração.

A única parte de uma mensagem SOAP que é obrigatória é o corpo (*Body*). Esta seção age como um recipiente para os dados que são entregues pela mensagem. Esses dados são freqüentemente chamados de carga útil ou dados de carga útil.

O construtor *Body* pode ser usado também para hospedar informações de exceção dentro de elementos de falha (*Fault*). Seções do tipo *Fault* podem residir junto com informações úteis dentro de uma mensagem, contudo, este tipo de informação é freqüentemente enviado em separado, dentro de mensagens de respostas criadas especificamente para comunicar condições de erro.

O construtor *Fault* consiste em uma série de elementos de sistema utilizados para identificar características de exceção.

3.2.3 Exemplo de um arquivo SOAP

O exemplo abaixo, descreve a reserva de uma passagem de avião.

O cabeçalho, contém as informações do passageiro e a data da reserva da passagem.

O corpo da mensagem, contém as informações de itinerário de partida e retorno: local da partida, local da chegada, data da partida, tempo de partida e o assento de preferência. Ainda no corpo da mensagem, são descritas as informações de preferência de hotel.

```
<?XML version='1.0' ?>
<env:Envelope XMLns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:Header>
  <m:reservation XMLns:m="http://travelcompany.example.org/reservation"
        env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
         env:mustUnderstand="true">
   <m:reference>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d</m:reference>
   <m:dateAndTime>2001-11-29T13:20:00.000-05:00</m:dateAndTime>
  </m:reservation>
  <n:passenger XMLns:n="http://mycompany.example.com/employees"
        env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
         env:mustUnderstand="true">
```

```
  <n:name>Åke Jógvan Øyvind</n:name>
 </n:passenger>
</env:Header>
<env:Body>
 <p:itinerary
   XMLns:p="http://travelcompany.example.org/reservation/travel">
  <p:departure>
    <p:departing>New York</p:departing>
    <p:arriving>Los Angeles</p:arriving>
    <p:departureDate>2001-12-14</p:departureDate>
    <p:departureTime>late afternoon</p:departureTime>
    <p:seatPreference>aisle</p:seatPreference>
  </p:departure>
  <p:return>
    <p:departing>Los Angeles</p:departing>
    <p:arriving>New York</p:arriving>
    <p:departureDate>2001-12-20</p:departureDate>
    <p:departureTime>mid-morning</p:departureTime>
    <p:seatPreference/>
  </p:return>
 </p:itinerary>
 <q:lodging
  XMLns:q="http://travelcompany.example.org/reservation/hotels">
  <q:preference>none</q:preference>
 </q:lodging>
</env:Body>
</env:Envelope>
```

3.3 WSDL

WSDL é um formato de XML para descrever serviços de rede como uma informação orientada a documento ou orientada a procedimento que operam nas mensagens. As operações e as mensagens são abstratamente descritas, e limitam-se então a um protocolo de rede e a um formato concreto da mensagem para definir um nó.. Os nós concretos relacionados são combinados em nós abstratos (serviços).

3.3.1 Exemplo de um WSDL

O exemplo abaixo mostra a definição de um WSDL, que provê um simples serviço de informações de estoque. O serviço suporta uma simples operação, chamada *GetLastTradePrice*, que é controlada usando protocolo SOAP 1.1 com http. A requisição recebe um “tickeSymbol” do tipo String e retorna o preço do tipo float.

```
<?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>
      <element name="TradePrice">
        <complexType>
          <all>
            <element name="price" type="float"/>
          </all>
        </complexType>
      </element>
    </schema>
  </types>

  <message name="GetLastTradePriceInput">
    <part name="body" element="xsd1:TradePriceRequest"/>
  </message>

  <message name="GetLastTradePriceOutput">
    <part name="body" element="xsd1:TradePrice"/>
  </message>

  <portType name="StockQuotePortType">
    <operation name="GetLastTradePrice">
      <input message="tns:GetLastTradePriceInput"/>
      <output message="tns:GetLastTradePriceOutput"/>
    </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>
```

3.4 UDDI

O protocolo *Universal Description Discovery and Integration* (UDDI) é um dos pilares para o sucesso de *Web Services*. UDDI cria uma plataforma padrão de interoperabilidade, que permite as companhias e as aplicações encontrar (rapidamente, facilmente e dinamicamente) e usar os *Web Services* espalhados pela internet.

A arquitetura de UDDI consiste basicamente em três partes:

a) modelo de Informação: um XML Schema que descreve os negócios e serviços web;
b) API UDDI: Uma API baseada em SOAP para publicação e busca de informação UDDI;
c) *UDDI Business Registry*: são sites operadores, que provêem implementações da especificação UDDI e sincronizam todos os dados sobre uma "*scheduled basis*". Exemplos de UBR's providos por grandes empresas são Microsoft, IBM, HP e SAP.

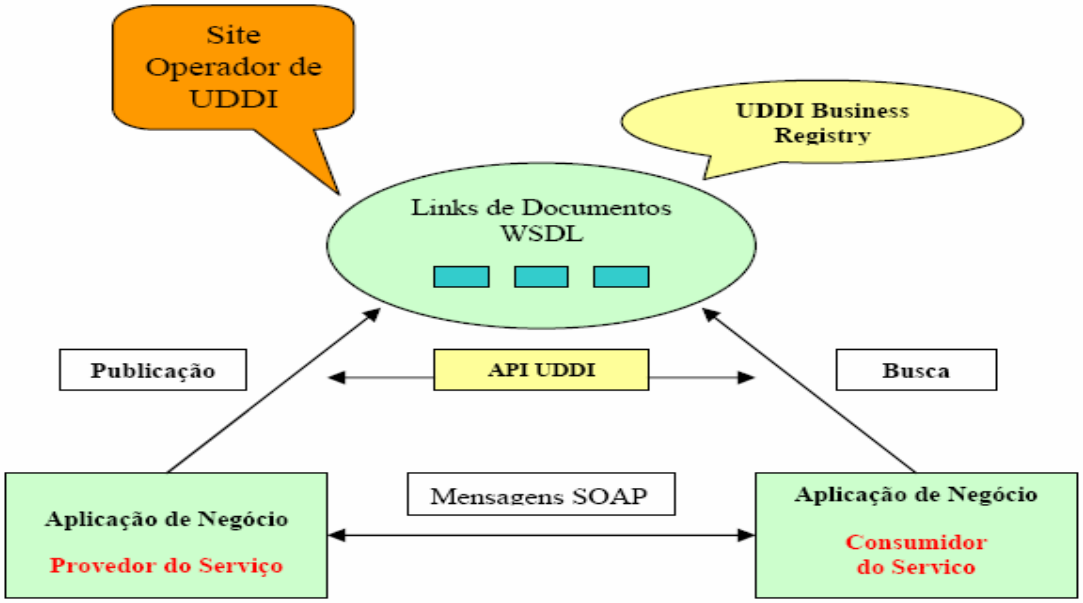

Figura 3.4: Arquitetura técnica de serviços na WEB.

3.4.1 Sites Operadores

A organização que hospeda uma implementação do *UDDI Business Registry* (UBR) são os Sites Operadores. Uma empresa que usa UDDI necessita se registrar em apenas um Site Operador, conhecido como *custodian,* tendo como principio básico, “Registrar uma vez e publicar em qualquer lugar”. A informação contida em um UBR *custodian* é replicada sobre os outros UBR. Esse processo de replicação é a atualização dos registros. Assim, uma empresa registrando os seus dados em um Site Operador, elas aparecerão também em outros Sites Operadores. Essa replicação não ocorre instantaneamente pelo motivo de disponibilidade das informações, mas os Sites Operadores Sincronizam as informações em intervalos de tempos, periodicamente.

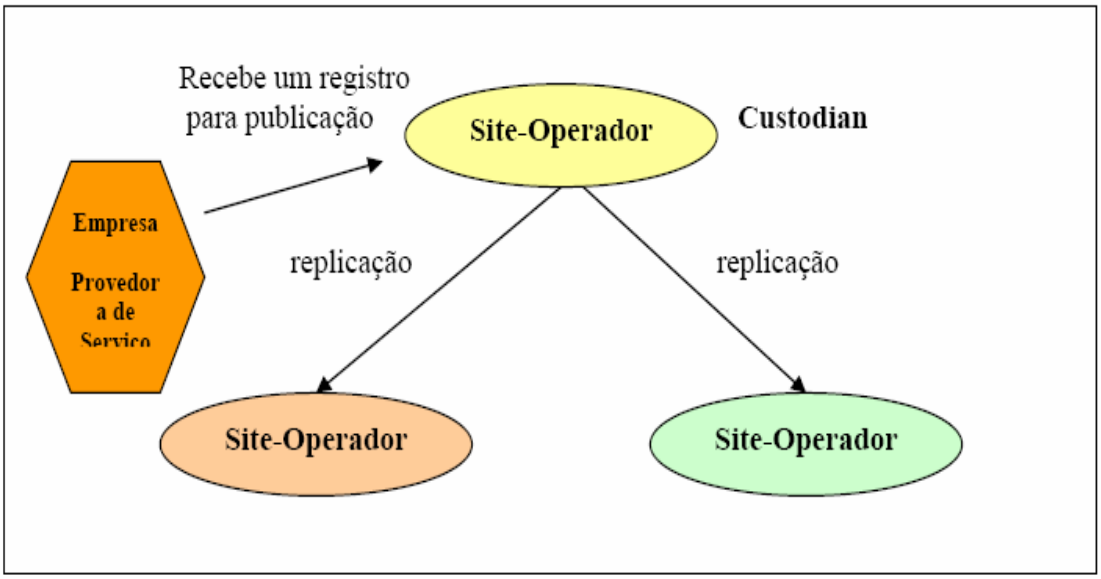

Figura 3.5: Diretório Distribuído de Sites Operadores UDDI.

Mas é possível também fazer registros UDDI privados, podendo assim estabelecer seu próprio sistema de UDDI, registrando todos seus serviços WEB internos. Quando esses serviços não são sincronizados automaticamente com os Sites Operadores UDDI, eles não são considerados parte do conjunto de *UDDI Business Registries.*

3.5 *Wireless Toolkit*

O J2ME *wireless Toolkit* suporta o desenvolvimento de aplicações Java em dispositivos com o perfil MIDP, como telefones celulares, *pagers* e *palmtops*.

O WTK suporta um número de maneiras para desenvolvimento de aplicações MIDP. Pode se realizar o processo de desenvolvimento usando as ferramentas de linha de comando ou usando os ambientes de desenvolvimento que automatizam uma grande parte deste processo. O *KtoolBar*, incluído com o WTK é

um ambiente mínimo de desenvolvimento com uma GUI para compilar, empacotar e executar aplicações MIDP. As únicas ferramentas a mais que são necessários são um editor de código fonte e um debugger. Uma IDE compatível com o WTK fornece maior conveniência para o desenvolvimento pois a própria IDE se encarrega de fazer o processo com o mínimo de conhecimento, sendo que a maioria dos processos se torna transparente para o desenvolvedor.

3.5.1 Suporte a Web Services no J2ME Wireless Toolkit

O J2ME Wireless Toolkit, a partir da versão 2.1, inclui as bibliotecas necessárias para desenvolvimento do J2ME Web Services. A ferramenta inclui um gerador de stubs JAX-RPC, o qual pode ser rodado em linha de comando ou pelo menu do KToolBar, como mostra a figura:

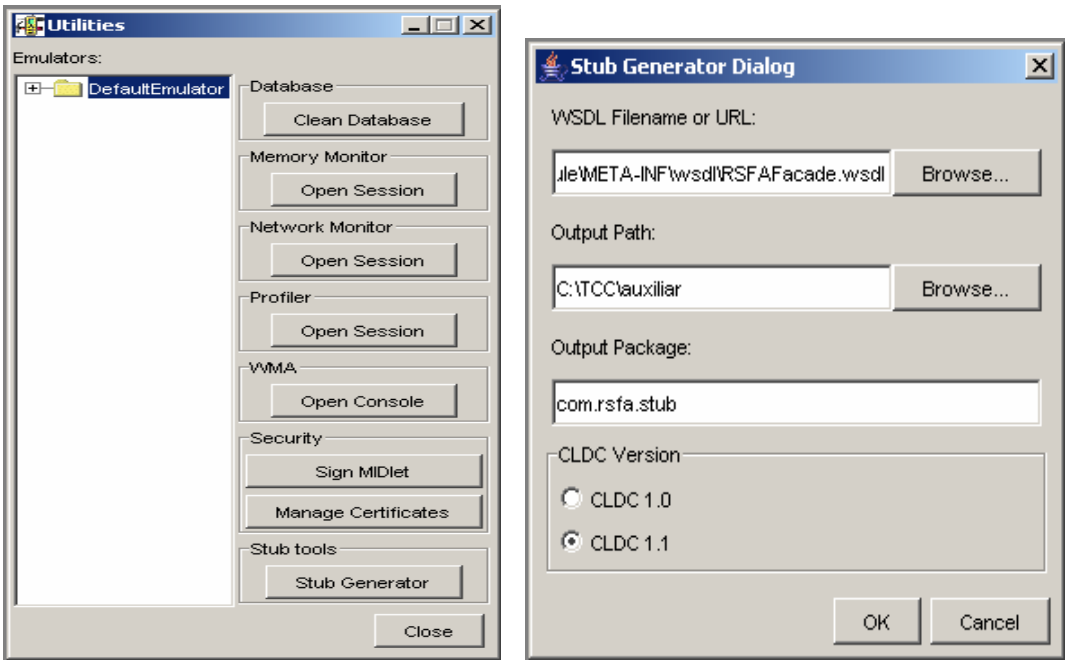

Figura 3.6: J2ME Wireless Toolkit 2.1 Utilities and Stub Generator.

4 JSR 172

Desenvolvido dentro do JCP como JSR 172, a API J2ME de Web Services(WSA) estende o J2ME para suporte a Web Services. Os dois pacotes opcionais da API padronizam duas áreas de funcionalidades cruciais para clientes Web Services: invocação de serviço remoto e parser XML.

WSA foi projetado para trabalhar com os perfis J2ME com base na configuração CDC ou na configuração CLDC 1.0 ou CLDC 1.1. A API de invocação remota é baseada em um subconjunto restrito da API de RPC XML-Based (Jax-RPC 1.1) do J2SE, com algumas classes do RMI incluídas para satisfazer as dependências JAX-RPC. O XML-Parser é uma API baseada em um conjunto restrito da API para XML, versão 2 (SAX 2).

O objetivo do WSA é integrar o suporte a invocação de Web Services e parser XML no ambiente de execução do dispositivo, sendo que assim os desenvolvedores não precisam adicionar tal funcionalidade em cada aplicação. Em particular, o processamento do XML se mostra especialmente caro em dispositivos com recurso limitados como telefones móveis e PDA's.

A especificação e o protocolo que define Web Services são feitos pelo Web Services Interoperability Organization(WS-I), pelo World Wide Web Consortium(W3C) e pela Organization for the Advancement of Strutctured Information Standards (OASIS). Quatro padrões chaves especificam como criar, instalar, encontrar e usar Web Services:

Web Services Standard	Description
Simple Object Access Protocol(SOAP) 1.1	define o transporte e a codificação de dados.
Web Services Definition Language (WSDL) 1.1	Define como os serviços remotos são descritos.
Universal Description,Discovery, & Integration(UDDI) 2.0	define como os serviços remotos são encontrados
Extensible Markupt Language (XML) 1.0 and XML Schema	Define o XML e o XML Schema

Tabela 4.1: Tabela de padrões que especificam Web Services.
Fonte: ORTIZ, (2004).

Essas especificações preliminares tendem a ser muito ampla, e os desenvolvedores de Web Services encontram dificuldades de conseguir uma completa interoperabilidade. Para resolver diferenças na interpretação dos padrões, o WS-I definiu regras de conformidades chamada de WS-I Basic Profile, versão 1.0. a JSR 172 está em conformidade com o Basic Profile.

4.1 Web Services na plataforma J2ME

JSR 172 especifica a tecnologia padrão do lado cliente para permitir que aplicações J2ME consumam serviços remotos em arquiteturas típicas de Web Services, como a figura 4.2 ilustra:

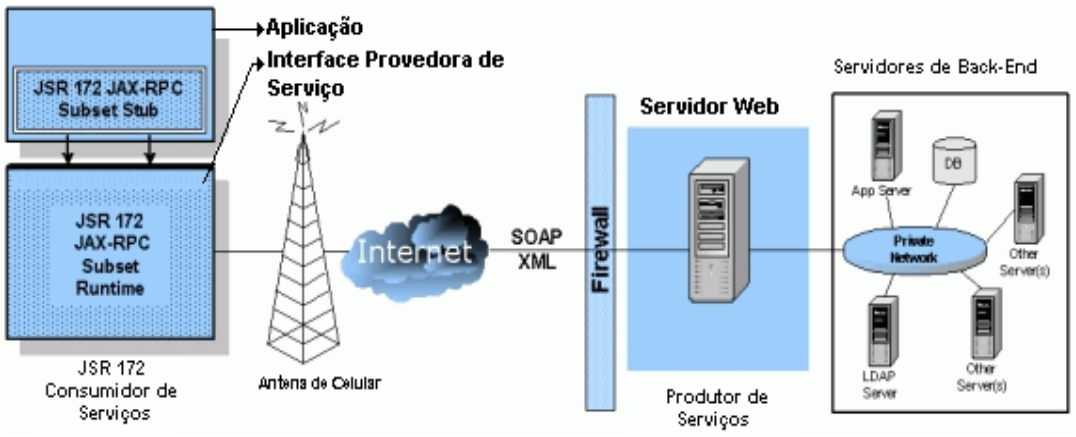

Figura 4.1: J2ME Web Services em uma arquitetura típica de Web Services.
Fonte: ORTIZ, (2004).

Em um nível mais alto, esta arquitetura de Web Services tem três elementos:

a) Uma aplicação cliente que está em um dispositivo Wireless WSA-Enabled. A aplicação inclui um stub JSR 172 que usa a JSR 172 em tempo de execução para comunicação com a rede;
b) A rede Wireless, a internet, a comunicação correspondente e o protocolo de codificação dos dados, incluem protocolos binários, como HTTP e SOAP/XML;
c) O servidor web, atuando como produtor de serviços, tipicamente está atrás de um ou mais firewalls e proxy. O servidor web freqüentemente provém acesso as aplicações de back-end e servidores na rede privada.

4.2 Entendendo a JSR 172

Uma típica aplicação baseada na JSR 172 é organizada conforme a figura:

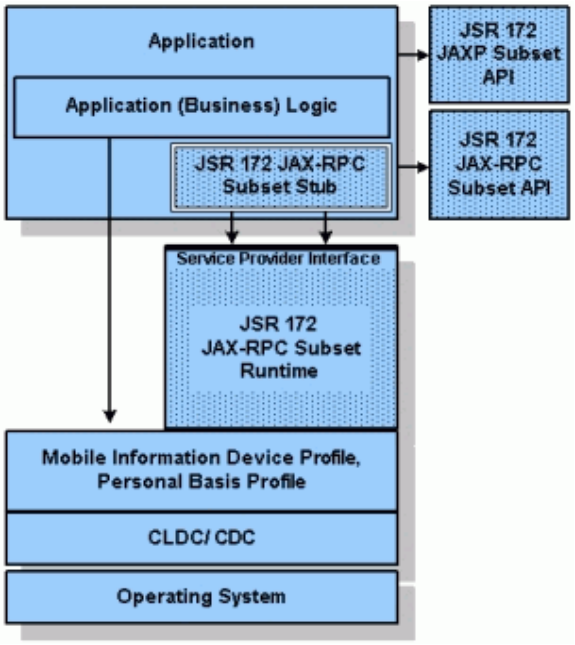

Figura 4.2: Organização típica de uma aplicação baseada na JSR 172.
Fonte: ORTIZ, (2004).

A própria aplicação é um cliente baseado no perfil MIDP ou no PBP, com lógica de negócio específica, interface com usuário, lógica de persistência, ciclo de vida e gerenciamento do estado da aplicação. Para manipular documentos XML, a aplicação pode usar o subconjunto de API's do JAXP. Para consumir Web Services, ela pode usar o subconjunto de API's do JAX-RPC, usando os stubs JSR 172.

Em dispositivos como os telefones celulares, as aplicações e o stub JSR 172 tipicamente ficam na memória do dispositivo, enquanto todos os elementos da JSR 172, junto com o perfil e a configuração subjacentes, são colocados no próprio dispositivo.

4.3 A JSR 172 Runtime e a interface provedora de serviços.

O centro de operações da JSR 172 é o seu suporte de execução (runtime), com a interface que provê os serviços, que permite os stubs executar todas as tarefas associadas com a invocação de um endpoint do serviço de RPC:

a) Define propriedades específicas para uma invocação RPC;
b) Descreve a entrada e o retorno de valores de uma invocação RPC;
c) Codifica valores de entrada;
d) Invoca um endpoint do serviço de RPC;
e) Decodifica e retorna para a aplicação qualquer valor que o endpoint do serviço retornar.

A figura descreve os relacionamentos entre a aplicação cliente, o stub, e o runtime:

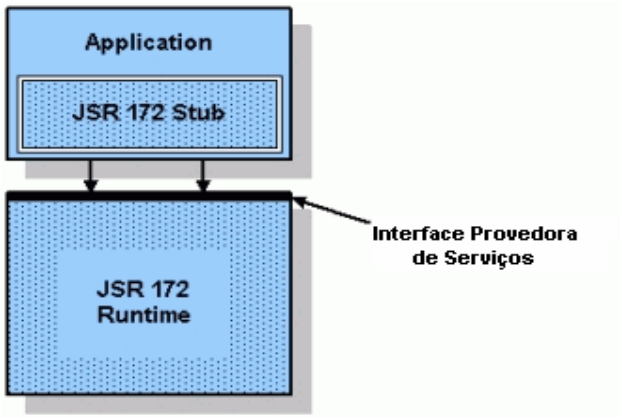

Figura 4.3: A JSR 172 Runtime e SPI (interface provedora de serviços).
Fonte: ORTIZ, (2004).

Enquanto o suporte de execução esconde a complexidade do gerenciamento de conexões e da codificação de dados, o SPI desacopla os stubs

dos detalhes da implementação do runtime, permitindo portabilidade dos stubs com as implementações de diversos fabricantes.

A aplicação cliente não interage com o suporte de execução e com a SPI diretamente; toda a interação é feito através dos stubs. O suporte de execução e a SPI são principalmente de interesse dos fabricantes que pretendem desenvolver ferramentas de automatização para JSR 172, tais como o a geração de código Java a partir de um arquivo WSDL. Essas ferramentas são usadas por desenvolvedores para gerar os stubs.

4.4 A JSR 172 e o subconjunto de API's JAX-RPC

Como foi mencionada, a API de invocação remota da JSR 172 Web Services é baseada em um conjunto restrito do subconjunto da API JAX-RPC 1.1 do J2SE. Abaixo, a lista que descreve esse subconjunto:

a) Conformidade com o WS-I Basic Profile;
b) Suporte a SOAP 1.1;
c) Suporte a todos os transportes, como HTTP 1.1, responsável por enviar mensagens SOAP;
d) Suporte a um conjunto completo de tipos de dados: boolean, byte, short, int, long, float, double, String, arrays de tipos primitivos, e tipos complexos (estruturas que contém tipos primitivos ou tipos complexos);
e) Suporte a representação literal da mensagem SOAP representando uma chamada ou resposta RPC (uma operação WSDL usa o modo de mensagem Literal); não suporta representação de codificada;
f) Não suporta mensagens SOAP com anexos;
g) Não suporta busca de serviços (UDDI).

Abaixo, a descrição dos pacotes do subconjunto JAX-RPC da JSR 172:

Java Package	Description
Javax.microedition.XML.rpc:	- Define o tipo, usado pelos stubs, na qual enumera os tipos simples válidos: boolean, byte, short, int, long, float, double, String. -Define ComplexType, usado pelos stubs, na qual é um tipo especial, que representa a definição WSDL xsd:complextype. - Define Element, usado pelos stubs, na qual representa a definição WSDL xsd:element. - Define Operation, usado pelos stubs, na qual representa um WSDL wsdl:operation para definição de um serviço. - Define FaultDetailHandler, uma interface implementada pelos stubs que manipula erros customizados. - Define FaultDetailException, usado pelos stubs, uma classe de Exception que representa uma exception jogada pela implementação do runtime como resultado de uma JAXRPCException
Javax.XML.namespace	-Define uma classe QName(qualified name) que é usada pelo stub.
Javax.XML.rpc	- Define a interface stub, na qual é a interface base para todos os stubs JAX-RPC.
Java.rmi	um subconjunto do pacote padrão Java.rmi incluído para satisfazer dependências JAX-RPC. Define apenas a interface remota,e a árvore de classes Exception: MarshalException, RemoteException, ServerException.

Tabela 4.2: Tabela de pacotes do subconjunto JAX-RPC da JSR 172.

Fonte: Ortiz (2004).

4.5 Mapeando os tipos de dados WSDL para Java

Os tipos de dados presentes em WSDL são mapeados da forma descrita na tabela 7 para os tipos equivalente em J2ME.

Tipo de Dado WSDL	Tipo de Dado Java
xsd:string	String
xsd:int	int ou Integer
xsd:long	long ou Long
xsd:short	short ou Short
xsd:boolean	boolean ou Boolean
xsd:byte	byte ou Byte
xsd:float	String, float ou Float
xsd:double	String, double ou Double
xsd:Qname	Javax.XML.namespace.Qname
xsd:base64Binary	byte[]
xsd:hexBinary	byte[]
Arrays	Baseado no schema do array XML
xsd:complexType	Sequencia de tipos primitivos e classes

Tabela 4.3: Tabela com os mapeamentos de dados wsd – Java.

CLDC 1.0 não suporta ponto flutuante, portanto, float e double devem ser mapeados para String. Desenvolvedores usam esse mapeamento default quanto incluem as duas configurações: CLDC 1.0 e CLDC 1.1.

4.6 JAXP

A API JAXP de parsers XML é baseada num subconjunto restrito da Simple API para XML, versão 2 (SAX2). Mesmo que a maioria das classes da API padrão do SAX2 não estão nesse subconjunto, a API JAXP contém todas as funcionalidades necessárias.

A lista abaixo descreve o subconjunto JAXP:

a) Baseado no subconjunto restrito do SAX 2.0;
b) Suporte a XML namespaces;
c) Suporte a UTF-8 e UTF-16;
d) suporte a *Document Type Definition* (DTD).

JAXP não possui suporte a *Document Object Model* (DOM), por ser considerado muito pesado, e tampouco para *Extensible Stylesheet Language Transformations* (XSLT);

4.7 Conclusão

A especificação do J2ME Web Services padronizou a programação de interfaces para Web Services e de parsers XML na plataforma J2ME. Com o advento do subconjunto de API JAX-PRC, desenvolvedores podem usar a linguagem de programação Java e as API's familiares JAX-RPC para criar aplicações que utilizam serviços remotos baseados em XML, sem ter que ir a um nível mais baixo de detalhes, manipulando mensagens do HTTP e do SOAP. E com a introdução do subconjunto de API JAXP, o parser XML é agora parte da própria plataforma J2ME.

A API de J2ME Web Services elimina a necessidade de desenvolvedores adicionarem códigos para invocação remota e parsers XML em cada aplicação.

Esta poderosa API dá acesso muito mais fácil das aplicações móveis aos Web Services, mas os desenvolvedores devem ter em mente que os dispositivos provêem um ambiente restrito para as aplicações. Simplesmente portar aplicações Desktop ou aplicações corporativas baseadas em XML não é uma maneira apropriada para desenvolver clientes Web Services na plataforma J2ME. A consideração do poder de processamento do dispositivo, da vida da bateria, da largura de banda e da segurança é essencial.

5 PROJETO

5.1 Arquitetura

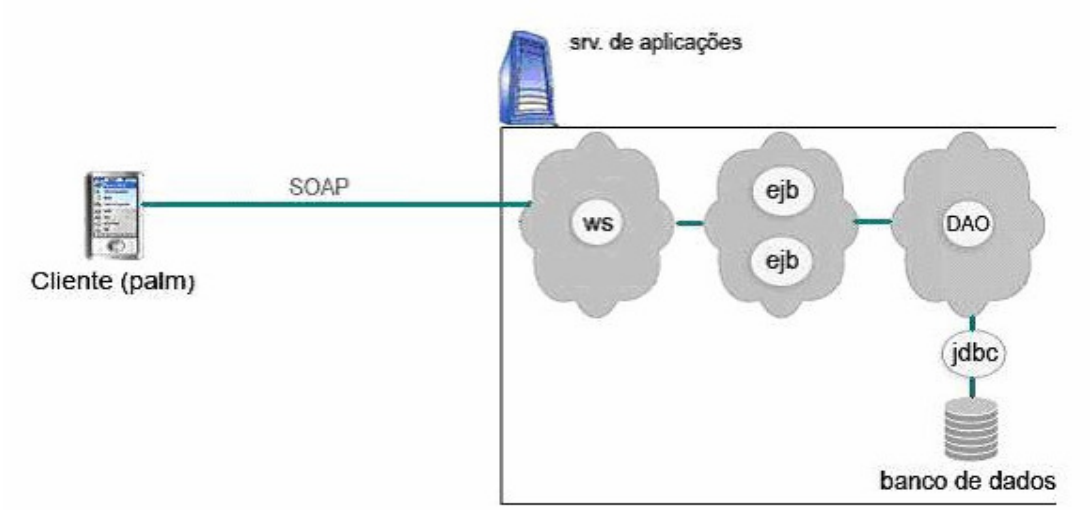

Figura 5.1: Arquitetura geral da aplicação.
Fonte: GUEDES, (2005).

A arquitetura projetada segue o modelo de arquitetura clássica cliente-servidor, onde numa ponta está o cliente, acessando o servidor via algum protocolo, ou até mesmo diretamente via RMI, e no servidor, as classes de negócio, de serviço e de persistência da aplicação. Ela implementa uma série de padrões de projeto (*design patterns*) como o Session Façade, que disponibiliza uma "Fachada" de serviços que podem ser acessados via Web Services; Business Delegate, que abstrai a complexidade de acesso a um EJB por um cliente desktop; Singleton, que possibilita o controle de objetos instanciados em memória, aumentando a performance da aplicação; e Abstract Factory, que é uma fábrica de fábricas de objeto.

O emprego de *design patterns* em projetos pequenos pode ser muito custoso, mas em ambientes corporativos, onde a complexidade das aplicações é grande, é indispensável sua utilização. Seu uso possibilita a reutilização de código, uma programação mais estruturada e uma maior manutenibilidade.

5.2 A aplicação cliente

O sistema cliente é uma aplicação para automação da força de vendas para uma empresa de laticínios. Permite ainda o controle e gerenciamento de títulos, contas a pagar e receber de produtores e fornecedores.

A Aplicação cliente foi baseada em um estudo de caso, proveniente da necessidade real de uma empresa de laticínios automatizar a entrega de leite aos seus clientes e o recolhimento da matéria prima junto aos seus fornecedores.

Foi feito um levantamento de requisitos junto ao dono da empresa onde foi passada toda a lógica do processo da empresa junto aos seus clientes e fornecedores. Com base nesses dados, foi montado o diagrama de classe e o diagrama de caso de uso da parte cliente do sistema que seriam dispositivos móveis.

A empresa beneficia leite e produz iogurte. Ela compra toda a matéria prima para realização desses serviços. A compra é feita direto na casa dos produtores, por um funcionário que vai em suas casas buscando o leite. Após chegar na usina, uma parte do leite, a grande maioria, é pasteurizada e a outra é feita o iogurte. No caso do iogurte, ele é distribuído em várias embalagens: Saco de 1 litro, garrafa de 250, 500 e 700 ml. Após o beneficiamento do leite e a produção de

iogurte, o mesmo funcionário faz a entrega nos clientes, que são mercados, mercearias, padarias e supermercados.

5.2.1 Diagrama de Caso de Uso

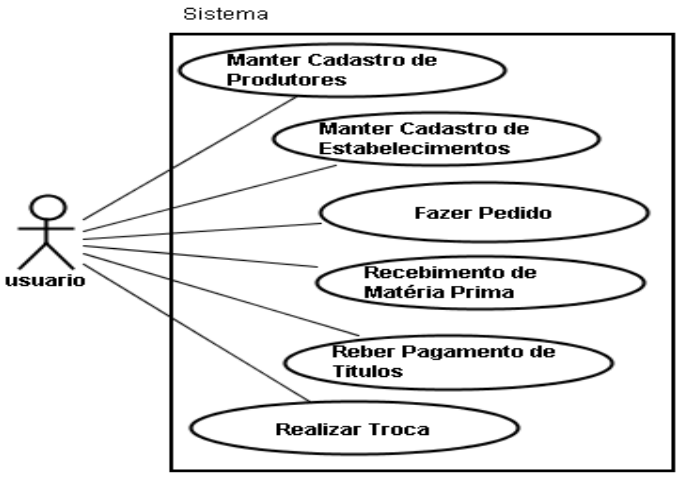

Figura 5.2: Diagrama de caso de uso da aplicação cliente.
Fonte: Primária, (2005).

5.2.2 Caso de Uso: Manter Clientes

O caso de uso Manter Clientes especifica o fluxo para cadastro e alteração de fornecedores e de produtores.

Descrição e Fluxo do caso de uso:

a) Descrição*:* manter um cliente na aplicação - inicialização dos campos (valores *defaults*);
b) Pré-condições: usuário logado;

c) Pós-condições: cliente gravado;
d) Fluxo básico: incluir estabelecimento:

- O ator seleciona a opção *"Cadastro"* no menu "Cliente/Estabelecimento" para incluir um novo Estabelecimento;
- O sistema verifica se existe um cliente selecionado;
- O Sistema carrega o campo código com o código seqüencial;
- O Sistema carrega o campo desconto com o valor 0;
- O ator preenche o campo nome;
- O ator preenche o campo contato;
- O ator preenche o campo CNPJ;
- O ator define um desconto para o cliente, se tiver;
- O ator seleciona a opção "Endereço" no menu para incluir o endereço do cliente;
- O Sistema carrega a tela de Endereço com o campo tipo tendo como valor default, o tipo Entrega;
- O ator preenche o campo rua;
- O ator preenche o campo complemento;
- O ator preenche o campo bairro;
- O ator preenche o campo cidade;
- o ator preenche o campo CEP;
- o ator define o tipo de Endereço;
- O ator seleciona a opção "Telefone" no menu para incluir o telefone do cliente;
- Sistema carrega o campo tipo com o valor default "Comercial";
- O ator preenche o campo número;
- O ator preenche define o tipo do telefone;
- O ator preenche o campo descrição;
- O ator preenche o campo DDD;
- O ator seleciona a opção "Cliente" no menu para voltar a tela de cliente;
- O sistema carrega a tela de cadastro de cliente;
- O ator seleciona a opção "Salvar" no menu para salvar o cadastro;
- O sistema faz a validação de dados;

- O sistema salva o cadastro;
- O sistema carrega a tela de menu inicial da aplicação.

5.2.3 Fluxo alternativo 1: incluir produtor

a) Após o passo 5 do fluxo básico:
b) O ator preenche o campo CPF;
c) Continua no passo 8 do fluxo básico.

5.2.4 Fluxo alternativo 2: alterar cliente

a) O ator seleciona um cliente;
b) O sistema carrega os dados do cliente selecionado;
c) O ator pode alterar qualquer dado do cliente;
d) O sistema faz a verificação das regras para cada dado alterado, conforme fluxo básico de “inclusão de cliente”;
e) O ator seleciona a opção de salvar o cliente;
f) Volta ao passo 28 do fluxo básico.

5.2.5 Caso de Uso: Registrar Pedido

O caso de uso Registrar Pedido especifica o fluxo de registro de pedidos feito nos estabelecimentos.

Descrição e fluxo do caso de uso:

a) Descrição: registrar um pedido para o cliente selecionado;
b) Pré-condições: ter um cliente do tipo Estabelecimento selecionado;
c) Pós-condições: pedido gravado;
d) Fluxo básico:

- O ator seleciona a opção “Novo” no menu “Pedido”.
- O sistema carrega o campo valor total com valor default “0.00”
- O sistema carrega o campo “cliente” com o nome do cliente selecionado.
- O Sistema carrega o campo “Data do pedido” com a data atual do sistema.
- O ator preenche a data de entrega do pedido.
- O ator seleciona a opção “Itens” no menu.
- O sistema carrega os itens do pedido.
- O ator seleciona a opção “Adicionar item” no menu.
- O sistema carrega a busca de produtos.
- (*1)O ator preenche o tipo do produto.
- (*1) O ator preenche o sabor do produto.
- O ator seleciona a opção “Procurar” no menu.
- O sistema carrega todos os produtos do sistema.
- O ator seleciona um produto da lista.
- O sistema carrega o item com os dados do produto.
- O ator preenche a quantidade do produto.
- O ator salva o item.
- O sistema calcula o valor do item e soma ao valor total do pedido.
- O sistema carrega o pedido.
- O ator seleciona a opção “Salvar” do menu.
- O sistema Valida os dados.
- O sistema Salva o pedido.
- O sistema carrega o menu inicial do sistema.

Restrições/Ressalvas: (*1) Campo não obrigatório para o preenchimento.

5.2.6 Fluxo alternativo 1 – alterar pedido

a) O ator seleciona um pedido;
b) O sistema carrega os dados do pedido selecionado;
c) O ator pode alterar qualquer dado do pedido;
d) O sistema faz a verificação das regras para cada dado alterado, conforme fluxo básico de “Novo Pedido”;
e) O ator seleciona a opção de salvar o pedido;
f) Volta ao passo 21 do fluxo básico.

5.2.7 Caso de Uso: Receber Matéria Prima

O caso de uso Receber Matéria Prima especifica o fluxo de registro de recebimento de matéria prima feito junto aos produtores.

Descrição e fluxo do caso de uso:

a) Descrição: recebe a matéria prima junto ao produtor;
b) Pré-condições: ter um cliente do tipo produtor selecionado;
c) Pós-condições: registro de matéria prima salvo;
d) Fluxo básico:

- O ator seleciona a opção “Matéria Prima” no menu Recebimento;
- O sistema carrega o campo código com o código seqüencial;
- O sistema carrega o campo cliente com o nome do cliente selecionado;
- O sistema carrega o campo Data com a data atual do sistema;
- O ator seleciona a opção “Produto” no menu;
- O sistema carrega todos os produtos do sistema;
- O ator seleciona um produto;
- O sistema carrega os dados do registro de matéria prima com o nome do produto no campo produto;
- O ator preenche a quantidade do produto;

- O ator seleciona a opção “Salvar”;
- O sistema salva o registro de matéria prima;
- O sistema carrega o menu inicial do sistema;

5.2.8 Caso de Uso: Receber Pagamento de Títulos

O caso de uso Pagamento de Títulos especifica o fluxo de registro de pagamentos feito pelos estabelecimentos.

Descrição e fluxo do caso de uso:

a) Descrição: registra o recebimento efetuado para pagamento dos títulos em aberto de clientes do tipo Estabelecimento;
b) Pré-condições: ter um cliente do tipo Estabelecimento selecionado;
c) Pós-condições: Salva a alteração no título;
d) Fluxo básico:

- O ator seleciona a opção “Títulos” no menu Recebimento;
- O sistema carrega o campo cliente com o nome do cliente selecionado;
- O sistema carrega o campo Saldo Devedor com o valor total da dívida do cliente;
- O ator informa o valor pago pelo cliente;
- O ator seleciona a opção “Salvar” no menu;
- O sistema diminui o valor informado no passo 4 do valor do passo 3;
- O sistema salva o título;
- O sistema carrega o menu inicial do sistema.

5.2.9 Caso de Uso: Realizar Troca.

O caso de uso Realizar Troca especifica o fluxo para registro de troca de produtos feito nos estabelecimentos.

Descrição e fluxo do caso de uso:

a) Descrição: registra uma troca feita em um cliente do tipo Estabelecimento;
b) Pré-condições: ter um cliente do tipo Estabelecimento selecionado;
c) Pós-condições: salva o registro de troca;
d) Fluxo básico:

- O ator seleciona a opção “Trocas” no menu Pedidos;
- O sistema carrega a data da troca com a data atual do sistema;
- O sistema carrega o nome do cliente com o cliente selecionado;
- O ator seleciona a opção “Buscar Produto” no menu;
- O sistema carrega todos os produtos do sistema;
- O ator seleciona um produto;
- O sistema carrega o registro de trocas com o nome do produto selecionado;
- O ator informa a quantidade do produto que foi feita a troca;
- O sistema carrega o valor total com a conta: valor do produto x quantidade;
- O ator seleciona a opção “Salvar” do menu;
- O sistema salva o registro de troca;
- O sistema carrega o menu inicial do sistema.

5.2.10 Diagrama de Classe do Sistema

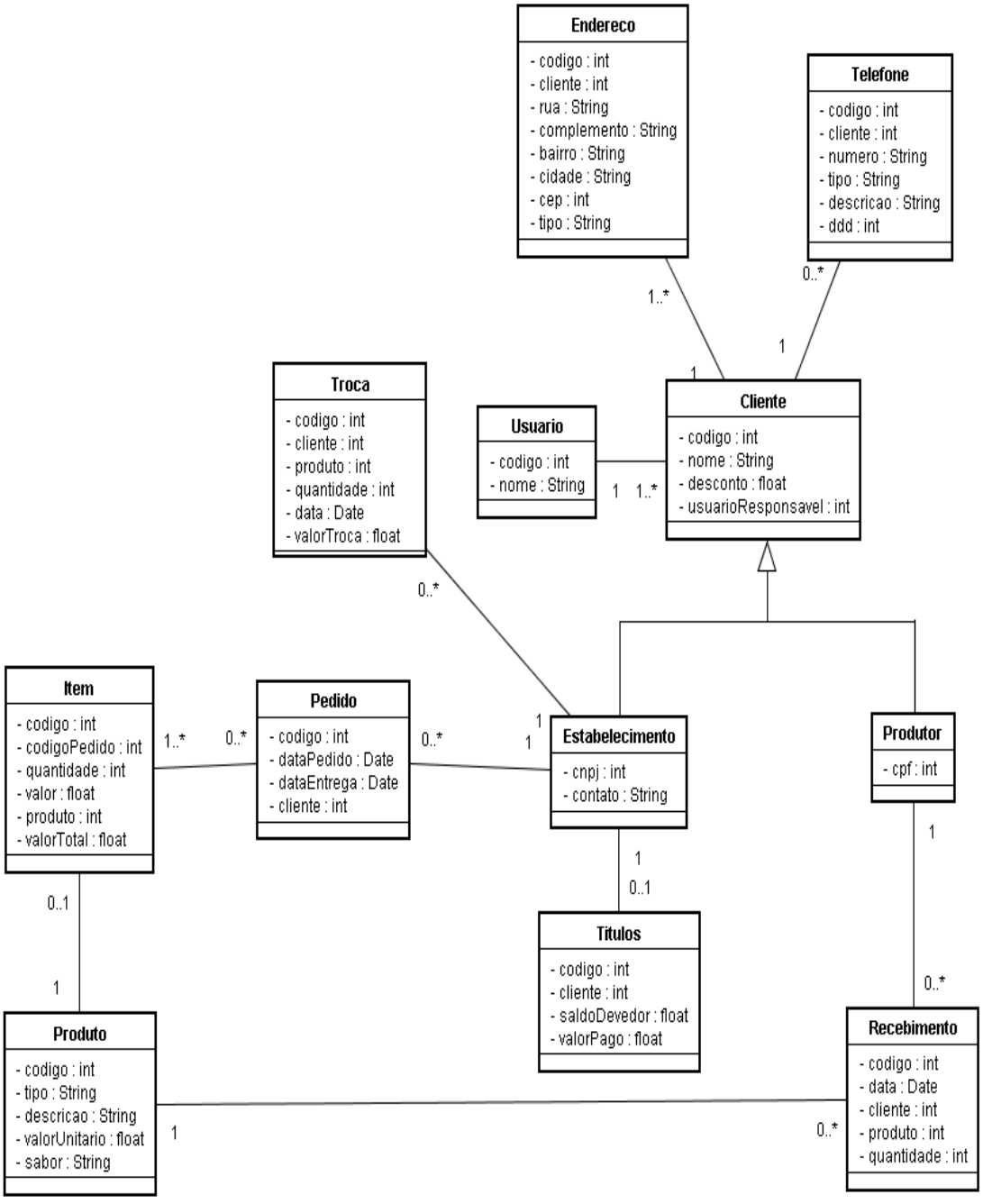

Figura 5.3: Diagrama de classe da aplicação Cliente.

Fonte: Primária, (2005).

As classes da aplicação são as seguintes:

Endereço: um endereço está relaciona a um cliente apenas. Endereço contém os campos código, código do cliente a quem pertence o endereço, nome da rua, complemento, bairro, cidade, cep e tipo de endereço;

Telefone: um telefone está relaciona a um cliente apenas. Telefone contém os campos código, código do cliente a quem pertence o telefone, número do telefone, tipo, descrição e DDD;

Cliente: um cliente possui um ou mais endereços. Um cliente possui um ou mais telefones. Cliente contém os campos código, nome do cliente, desconto que o cliente possui na compra de produtos e o usuário responsável por aquele cliente, ou seja, o usuário que fará a venda àquele cliente;

Estabelecimento: estabelecimento herda da classe cliente. Estabelecimento possui os campos CNPJ e contato, que é pessoa responsável pela compra de produtos no cliente. Um Estabelecimento pode ter nenhum ou um título relacionado. Um estabelecimento pode ter nenhuma ou várias trocas relacionadas. Um estabelecimento pode ter nenhum ou vários pedidos relacionados;

Produtor: produtor é herdado da classe cliente. Produtor possui o campo CPF. Um produtor pode ter nenhum ou vários recebimentos associados a ele;

Títulos: títulos é a classe que mantém a informação da dívida de um estabelecimento. Títulos contém os campos código, código do cliente a quem pertence o título, saldo devedor do cliente, e o valor que foi pago no momento em que está lançando um pagamento.

Recebimento: recebimento é a classe que registra a informação da quantidade de matéria prima que foi recebida pelos produtores. Recebimento

contém os campos código, data do recebimento, código do cliente que forneceu a matéria prima, código da matéria prima, e quantidade que foi vendida;

Troca: troca é a classe que registra a informação de troca de produto em um estabelecimento. Essa informação serve para contabilizar toda a entrega de produtos, já que as trocas não entram como venda, e não entrando como venda, não entram no caixa. Troca contém os campos código, código do cliente que fez a troca, código do produto que foi trocado, quantidade que foi trocada, data da troca e o campo valor da troca, que é um campo informativo;

Pedido: pedido é a classe responsável pelo registro da venda de produtos a um estabelecimento. Pedido está relacionado a um cliente apenas. Pedido pode ter um ou mais itens. Pedido contém código, data do pedido, data de entrega do pedido, e código do cliente a quem o pedido está sendo feito;

Item: item é a classe que registra as informações de um produto selecionado pelo usuário em um pedido de venda. Item pode pertencer a nenhum ou a vários pedidos. Item está relacionado com apenas um produto. Item contém os campos código, código do pedido a qual o item está relacionado, quantidade desse produto que foi vendida, o campo valor, o código do produto e o valor total do item;

Produto: um produto está relacionado a nenhum ou mais itens. Produto está relacionado a nenhum ou mais recebimentos. Produto contém os campos código, tipo do produto, descrição, valor unitário para venda e sabor do produto;

Usuário: usuário é o responsável pelo uso do sistema. Um usuário é responsável por um ou vários clientes. Usuário contém os campos código e nome.

5.2.11. Implementação do Sistema

O sistema foi dividido em grupos para uma melhor usabilidade e agilidade por parte do usuário. São eles:

a) cliente: trata toda a parte de clientes e fornecedores;
b) pedido: trata toda a parte de pedido e troca de produto junto aos clientes;
c) produto: trata a parte de produtos vendidos e comprados pela empresa;
d) recebimento: trata a parte de recebimento de matéria prima junto aos fornecedores e a parte financeira, tanto do pagamento de clientes quanto ao pagamento a fornecedores.

Essa foi a divisão lógica do sistema, Uma divisão que possibilita no futuro, subdividir facilmente o sistema em duas partes: atendimento ao cliente e atendimento junto aos fornecedores.

A divisão que modela o sistema foi feita da seguinte maneira:

a) classe de negócio;

b) classe de interface;

c) classe de gerenciamento.

Toda a classe de negócio estende a classe pai *Entity*. É nessa classe que está implementada toda a regra de persistência de dados e filtros de registros na base de dados. Nela está implementada também a serialização e transformação dos objetos em XML para comunicação com o servidor.

A base de dados é constituída de uma tabela, que seria um arquivo e dos registros, que são array de bytes gravado nesse arquivo. A persistência dos dados foi feita com a própria API do J2ME, chamada de RMS (*Record Management*

System). É ela que faz todo o gerenciamento da persistência e classes para implementação de filtros para buscas de registros.

A classe de negócio implementa dois métodos abstratos da classe mãe. Os dois métodos trabalham em conjunto*: getTypes* e *getFields*.

O *getTypes* retorna um array de bytes que identifica o tipo de dado de cada campo que é retornado no *getFields*. Assim, o primeiro registro do *array* do *getTypes* representa o tipo de dado do primeiro registro do *getFields*. O RMS salva e recupera o *array* de bytes com base nessa ordem.

A classe de negócio também pode implementar o método *checkConstraint*, que faz a validação dos dados de entrada da classe de interface, caso haja alguma a ser feita.

As classes de interfaces estenderam as principais classes do pacote lcdui do J2ME, bem como seus componentes: Form e List.

O sistema foi dividido nos seguintes pacotes:

a) com.rsfa.business: pacote onde contém todas as classes de negócio;

b) com.rsfa.framework: pacote onde contém classes da estrutura do sistema. São elas:

- Entity: classe pai de todas as classes de negócio;
- Filter: classe usada para montar filtros;
- EntityFilter: classe usada para executar os filtros;
- FieldMap: classe que identifica os tipos de dados que podem ser salvos no rms.

c) com.rsfa.gui.customer: pacote onde contém as classes de interface gráfica do cadastro de telefones, endereços e dados pessoais de clientes e produtores e também listas de registros;

d) com.rsfa.gui.list.manager: pacote onde contém a classe gerenciadora de listas do sistema;

e) com.rsfa.gui.main: pacote onde contém a classe de login e da árvore de menu do sistema;

f) com.rsfa.gui.order: pacote onde contém as classes de pedido e trocas do sistema;

g) com.rsfa.gui.product: pacote onde contém as classes de produtos do sistema;

h) com.rsfa.gui.recebimento: pacote onde contém as classes de cadastro de Recebimento de matéria prima junto aos fornecedores e de pagamentos de títulos;

i) com.rsfa.principal: pacote onde contém a classe que estende MIDlet;

j) com.rsfa.services.stub: pacote onde contém a classe que estende MIDlet;

l) com.rsfa.util: pacote onde contém classes auxiliares do sistema como classes de Formatação de datas e classes de Formatação de números.

Todos os pacotes contêm sua classe de gerenciamento. Todas as requisições feitas pelas classes de interface são feitas à sua classe gerenciadora. Por exemplo, se eu quisesse mostrar a lista de produtos a partir do meu pedido, a classe pedido pediria a classe *OrderManager* para chamar a tela de produtos. Por sua vez a classe *OrderManager* pediria a classe *ProductManager* para mostrar a tela de produtos. Dessa forma, a manutenção de solicitações entre classes se concentra em apenas uma classe, facilitando a manutenção e controle do sistema.

O Sistema possui cadastros de clientes e de fornecedores. A cada cliente cadastrado, é gerado automaticamente um título referente a esse cliente. O sistema também permite alteração de clientes e fornecedores bem como listar todos os clientes e fornecedores cadastrados.

O sistema permite também fazer pedidos de vendas. Para fazer um pedido, é obrigatório ter um cliente selecionado. Ao abrir a tela de pedidos, é carregado automaticamente o título referente a esse cliente para memória. O pedido deve conter no mínimo 1 item. Não há uma regra para quantidade máxima de itens que pode conter um pedido. Ao ser salvo o pedido, o valor do título é atualizado,

somando o valor do pedido a ele. O sistema ainda permite listar todos os pedidos realizados, bem como alterá-lo, caso não tenha sido sincronizado com a base central.

O sistema ainda permite registrar as trocas. Caso o vendedor vá fazer uma venda, e no estabelecimento tenha produtos vencidos, é feita uma troca. Esse registro é feito apenas para controle, não contando como uma venda.

O sistema permite fazer o registro de entrega de matéria prima pelos fornecedores, bem como listá-las.

Permite também fazer consulta de títulos dos clientes, e dar baixa nos títulos, ao receber o pagamento dos clientes. Nesse caso, o título será apenas zerado.

5.2.12 Telas da aplicação Cliente

Segue abaixo um passo a passo de cadastro de cliente e sincronização com o servidor:

a)

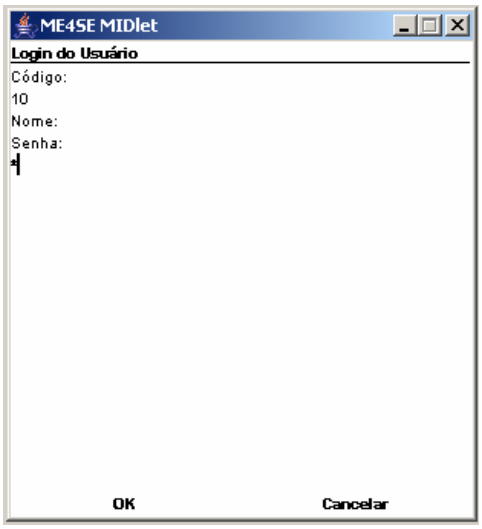

Figura 5.4: Tela de Login.

Fonte: Primária, (2005).

b)

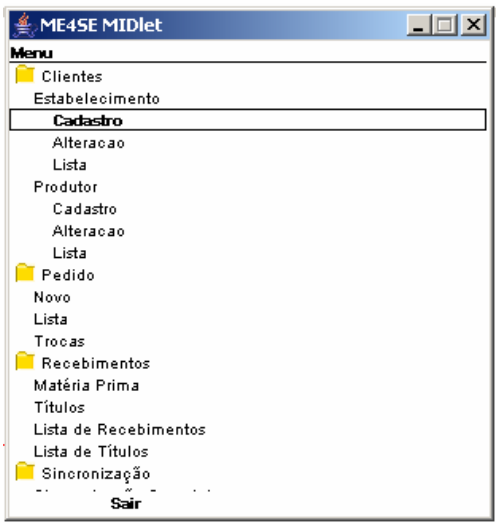

Figura 5.5: Menu principal.

Fonte: Primária, (2005).

a) o usuário entra com a senha. Se for a primeira vez que está acessando o sistema, o usuário terá que fornecer o seu código de acesso e digitar uma senha, e seleciona a opção *OK*;

b) o sistema abrirá a tela inicial. Selecione a opção *cadastro* no sub-menu estabelecimentos no menu principal;

c)

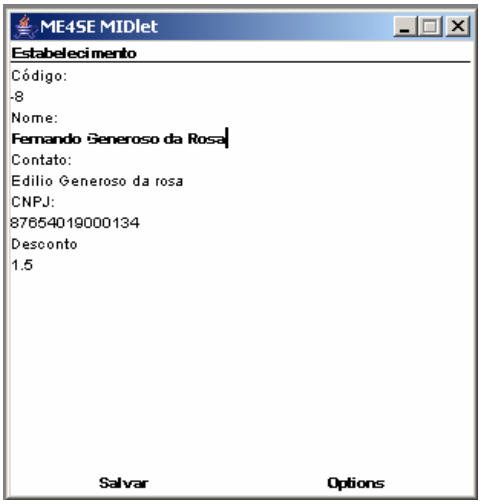

Figura 5.6: Cadastro de Cliente.

Fonte: Primária, (2005).

d)

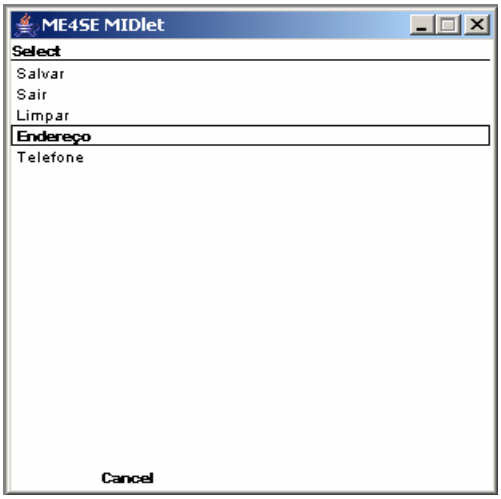

Figura 5.7: Menu de opções.

Fonte: Primária, (2005).

c) o usuário preenche o nome do estabelecimento, o cnpj do estabelecimento, o contato direto do estabelecimento e o desconto, se houver, que esse estabelecimento têm na venda dos produtos;

d) o usuário seleciona a opção *Options/Endereço*. O sistema irá carregar a tela para preenchimento do endereço do cliente;

e)

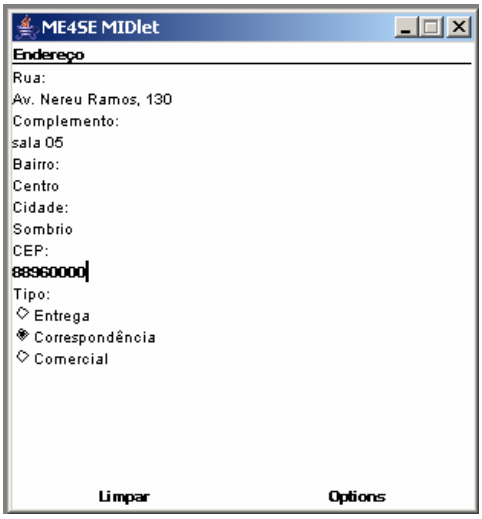

Figura 5.8: Cadastro de Endereço.
Fonte: Primária, (2005).

f)

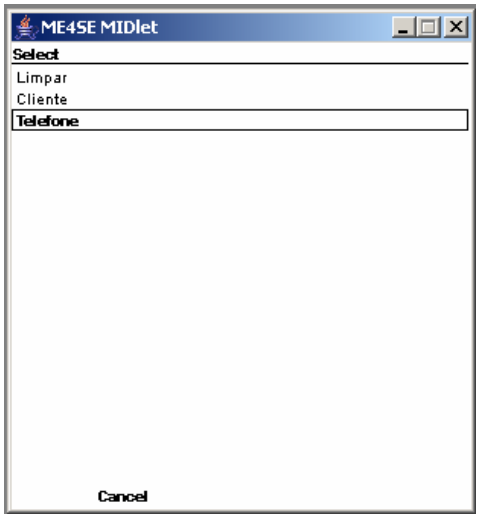

Figura 5.9: Menu de Opções.
Fonte: Primária, (2005).

e) o usuário preenche o nome da rua onde o estabelecimento funciona, complemento, bairro, cidade, cep e o tipo de endereço, se é para entrega, correspondência ou comercial;

f) o usuário seleciona q opção *Options/Telefone*. O sistema irá carregar a tela para preenchimento do telefone do cliente;

g)

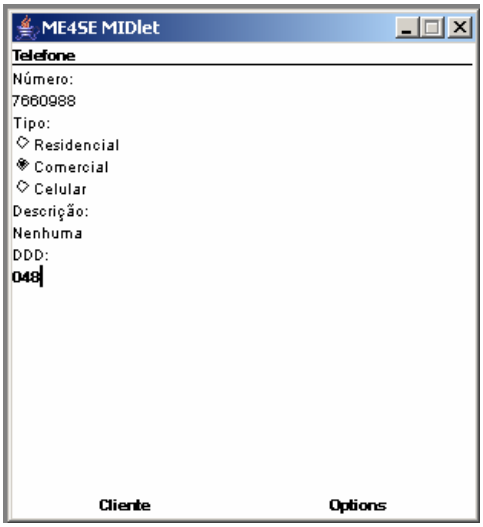

Figura 5.10: Cadastro de Telefone.
Fonte: Primária, (2005).

h)

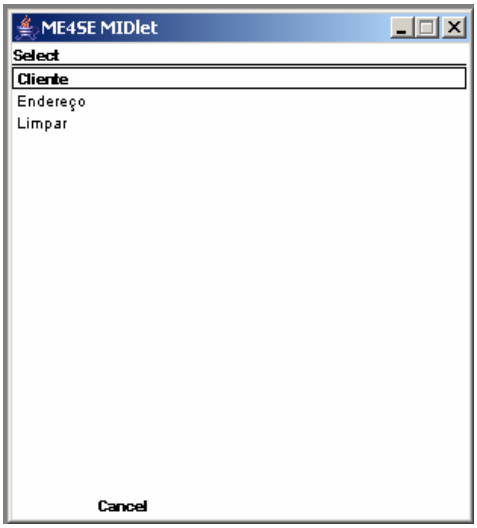

Figura 5.11: Menu de opções.
Fonte: Primária, (2005).

g) o usuário preenche o número do telefone, o tipo, se é telefone residencial, comercial ou celular, alguma informação adicional, no campo descrição e o DDD do estabelecimento;

h) o usuário seleciona a opção *cliente*. O sistema carrega novamente a tela de cadastro do cliente para finalizar o cadastro;

i)

Figura 5.12: Cadastro de cliente.
Primária, (2005).

j)

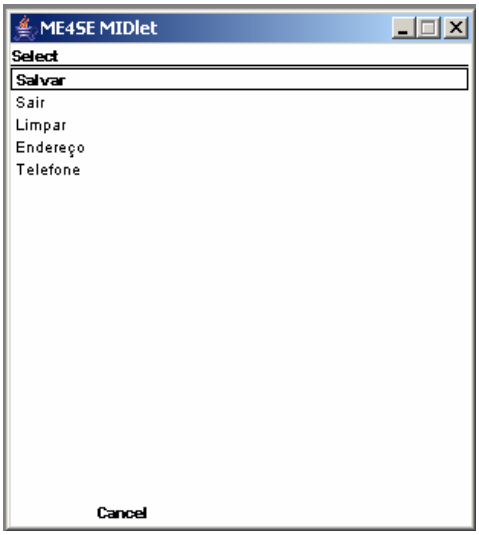

Figura 5.13: Salvando Cadastro.
Fonte: Primária, (2005).

i/j) o usuário seleciona a opção *salvar*. O sistema carrega novamente o menu principal;

l) m)

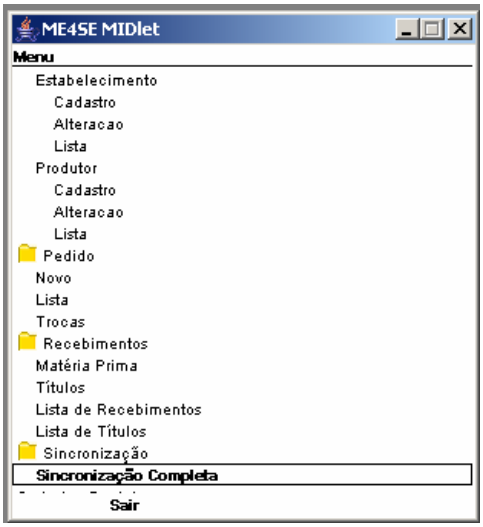

Figura 5.14: Menu Principal.
Fonte: Primária (2005).

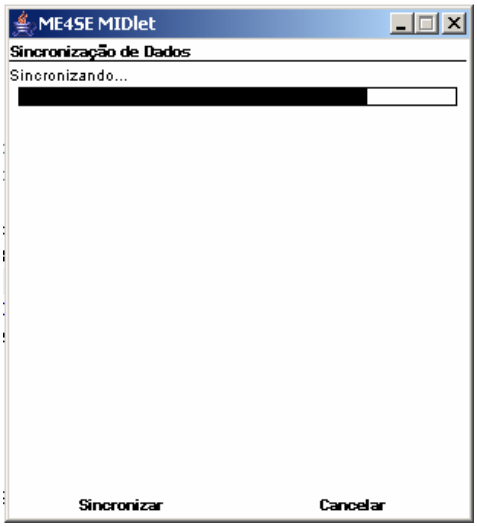

Figura 5.15: Sincronização dos dados.
Fonte: Primária (2005).

l) para sincronização dos dados, o usuário seleciona a opção *Sincronização completa*. O sistema carrega a tela de sincronização;

m) o usuário seleciona a opção *Sincronizar*, e os novos dados serão sincronizados com a base;

5.3 Análise

O desenvolvimento do sistema, além de automatizar o processo da empresa, permitirá que as informações de venda, de recolhimento de leite junto aos produtores, cheguem de uma maneira mais rápida e eficiente. Poderá ser obtido a informação da quantidade de leite que foi recolhida junto ao produtor antes do

funcionário chegar efetivamente na empresa para o beneficiamento da matéria prima.

As informações dos pagamentos dos clientes já irão vir com os totais calculados, permitindo que a empresa otimize custos operacionais, tempo e minimizar erros humanos.

5.4 A aplicação servidora

A Aplicação servidora é constituída de um Façade que é um EJB disponibilizado como *Web Services*. Foi feito um façade porque ele concentra todos os serviços que serão disponibilizados e que se encontram no servidor.

Contém também um *Enterprise Bean* que faz o gerenciamento das requisições dos serviços. Esse Bean recebe a requisição do Façade, processa a String recebida como parâmetro, transformando em XML, e repassa para o Bean que executará a regra de negócio correspondente.

No servidor existem dois módulos: Um módulo gerencia cadastros de usuários e clientes. O outro gerencia a parte de pedidos.

O servidor foi dividido nos seguintes pacotes:

a) tcc.server.ejb.account: neste pacote está o Enterprise Bean que gerencia os cadastros dos clientes e usuários do sistema;
b) tcc.server.ejb.manager: neste pacote está o Enterprise Bean que gerencia as requisições do façade e repassa para os respectivos Beans responsáveis pelo serviço;

c) tcc.server.ejb.order: neste pacote está o Enterprise Bean que gerencia a parte de serviços relacionados a pedidos, do sistema;

d) tcc.server.services.facade: neste pacote está o Enterprise Bean que é disponibilizado como Web Services. O Bean disponibiliza os seguintes serviços para acesso:

- createOrder: cria o pedido no servidor;
- getCustomers: retorna os clientes relacionados ao usuário que está sincronizando;
- getOrders: retorna os pedidos relacionados ao usuário que está sincronizando;
- getProducts: retorna todos os produtos cadastrado na base;
- registerAgent: cadastra um novo usuário na base;
- registerChange: cadastra um novo registro de troca na base;
- registerCustomer: cadastra um novo cliente na base.
- registerRecebimento: cadastra uma nova entrega de matéria prima;
- registerTitle: cadastra um novo título na base.

e) tcc.server.util;

f) tcc.server.util.dao;

g) tcc.server.util.parserXML.

5.4.1 Classes da Aplicação Servidora

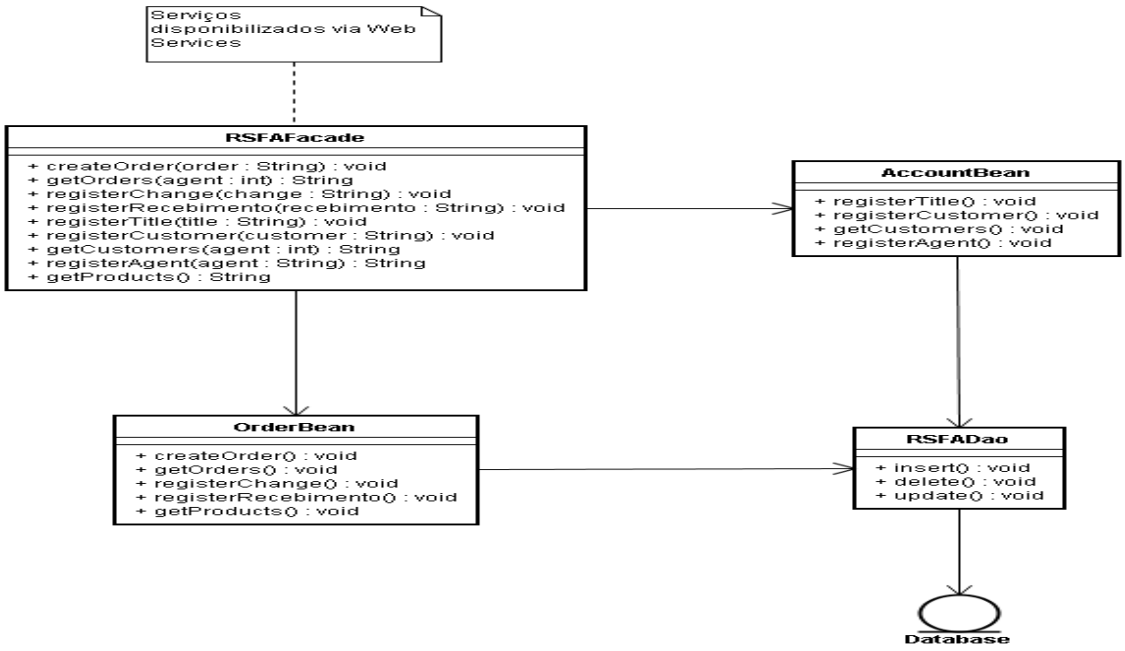

Figura 5.16: Diagrama de classe da aplicação Servidora.
Fonte: Primária, (2005).

As classes das aplicações servidoras são as seguintes:

RSFAFacade – Classe que contém todos os serviços disponíveis no servidor. Esse método serve como Fachada para acesso aos serviços. Todos os métodos recebem e/ou retornam uma String como parâmetro, contendo o XML com os dados a serem persistidos ou recuperados, no servidor.

AccountBean – Classe que contém os métodos referente à clientes. Todas as operações com clientes estão nesse método.

OrderBean – Classe que contém os métodos referente a Pedido. Todas as operações com pedidos, produto, troca, estão nesse método.

RSFADao – Classe que manipula a persistência dos dados. Os métodos insert, delete e update estão implementados nessa classe.

5.4.2 Projeto do banco de dados

A figura 5.17 apresenta o banco de dados utilizado pela aplicação servidora.

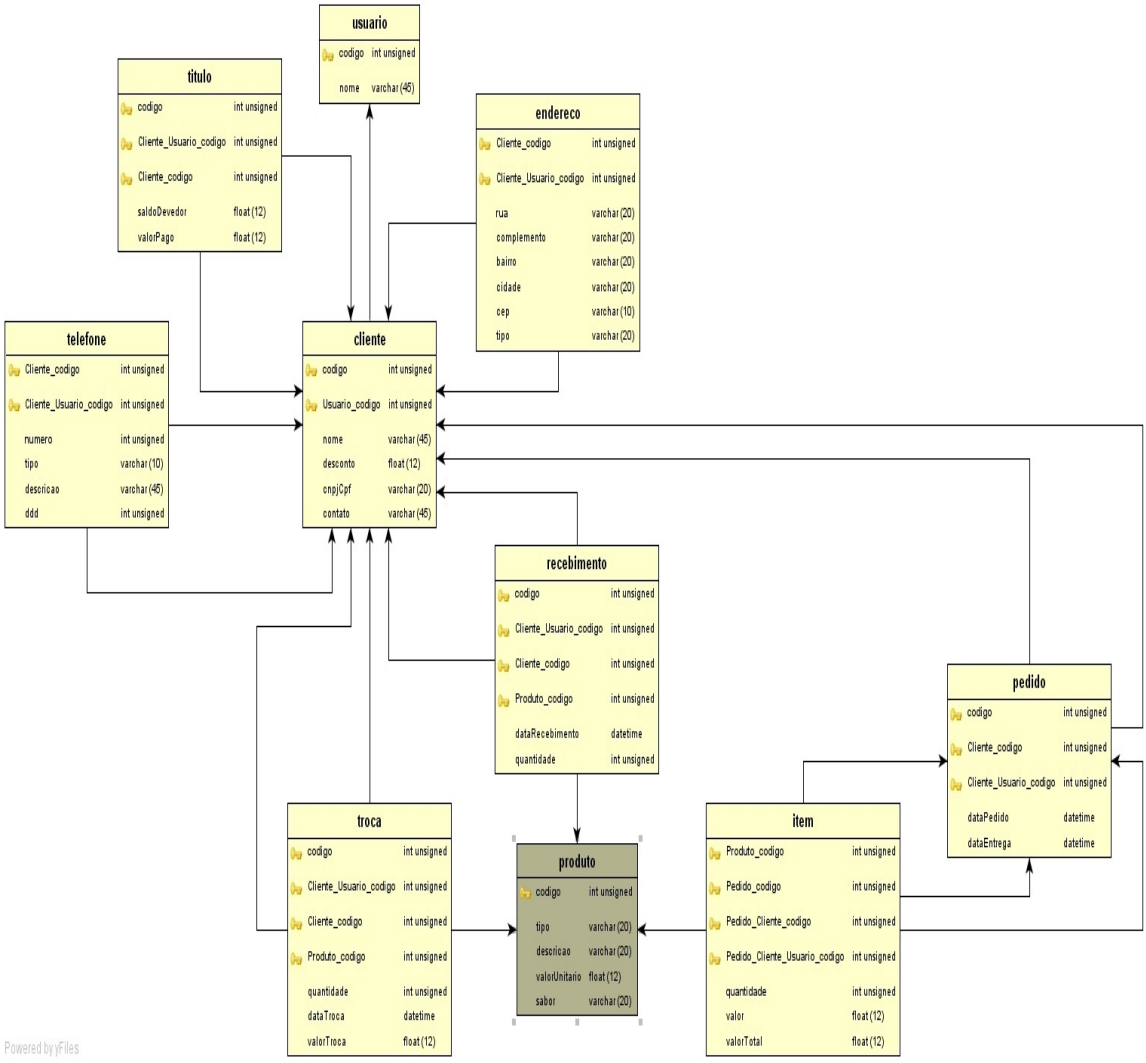

Figura 5.17: Diagrama do banco de Dados.

Fonte: Primária (2005).

Tabela: Produto - Contém todos os produtos vendidos e comprados pela empresa. Esses dados são cadastrados no servidor e enviados para os dispositivos móveis. Não há possibilidade de cadastro de produtos nos dispositivos.

Tabela Usuário - Contém todos os usuários que utilizam o Sistema. Na primeira sincronização do dispositivo, o usuário informa seu código e sua senha e o servidor retornará o nome de identificação ao dispositivo.

Tabela: Cliente - Contém o registro de todos os clientes, estabelecimentos e produtores, da empresa. Os clientes podem ser registrados no servidor e no dispositivo móvel, em uma eventual visita do usuário ao cliente.

Tabela: Pedido - Tabela que contém os registros de todas as vendas efetuadas pela empresa.

Tabela: Troca - Tabela que contém os registros das trocas de produtos efetuadas nos clientes.

Tabela: Item - Tabela que contém os registros dos itens dos pedidos feito pelos clientes.

Tabela: Recebimento - Tabela que contém os registros dos recebimentos de matéria prima feito no produtor.

Tabela: Telefone - Tabela que contém os registros dos telefones dos clientes.

Tabela: Titulo - Tabela que contém os registros dos títulos dos estabelecimentos. O título contém a informação do valor total da dívida do cliente relacionado.

Tabela: Endereço - Tabela que contém o registro de endereço dos clientes.

5.5 Ferramentas utilizadas

As ferramentas utilizadas para o projeto foram:

Eclipse IDE 3.1: IDE utilizada para desenvolvimento da parte cliente. Foi utilizado também o plugin Ant que vem com a ferramenta para automatizar o processo de deploy.

IBM *WebSphere Everyplace Micro Environment* 5.7.1: Foi utilizada a ferramenta JarToPrc.exe para gerar o arquivo executável para instalação da aplicação no dispositivo.

J2ME Wireless toolkit 2.2: Contém as API's J2ME para desenvolvimento de aplicações em dispositivos móveis. Foi usado o perfil MIDP 2.0 e a configuração CLDC 1.1.

KtoolBar: ferramenta para geração automática dos stubs para conexão com o Web Services.

ME4SE: Pacote de API's para emular uma aplicação desenvolvida em J2ME, no desktop.

Netbeans 4.1: IDE utilizada para desenvolvimento da parte servidora do projeto. Foi utilizado também o servidor de aplicação *Sun One Application Server*, que vém junto com a IDE.

6 CONSIDERAÇÕES FINAIS

Hoje Web Services é um padrão, principalmente no mundo corporativo e em soluções Business-to-Business (B2B), onde a heterogeneidade de sistemas e de tecnologias é fato, e que exigem maior interoperabilidade entre as aplicações, portabilidade entre os diversos sistemas e interação com tecnologias existentes no mercado. E disponibilizar os sistemas como um serviço, faz parte do amadurecimento da aplicação, dentro do seu ciclo de vida, principalmente de sistemas distribuídos, seja na intranet da empresa, ou na web, onde seu principal objetivo até pouco tempo, era apenas como provedor de informação, mas descobriu-se o grande potencial como provedor de serviços.

Os dispositivos móveis, em especial com suporte a J2ME, estão mostrando um grande potencial como clientes e consumidores desses serviços disponibilizados via web e são sem dúvida uma grande oportunidade de negócio. Hoje são 708 milhões de celulares rodando Java no mundo, em 635 modelos de celulares diferentes, 32 fabricantes de celulares com Java em todo o mundo e 140 operadoras de telefonia móvel usando Java [JAVAONE 2005]. Isso sem falar nos diversos modelos de Palms e HandHelds disponíveis no mercado que também suportam Java. Com esses números, se tem a noção do grande mercado que se forma em torno desses dispositivos, que permitem uma grande mobilidade do usuário, podendo acessar serviços web de qualquer parte do globo, a qualquer hora. O grande desafio se dá em desenvolver aplicações que consumam esses serviços, em dispositivos com capacidade ainda limitada de memória e de processamento. Um outro desafio não menos importante é a curva de aprendizagem da tecnologia

de Web Services, tanto para aplicações servidoras como para aplicações clientes como os dispositivos móveis, que é muito grande. Mas o mercado lança a cada dia novas ferramentas que abstraem muito da complexidade do desenvolvimento dessas tecnologias, tornando essa tarefa cada dia mais fácil, podendo o desenvolvedor ficar focado a maior parte do tempo no negócio.

A maior dificuldade no desenvolvimento da aplicação, foi pensar na arquitetura como um todo, mesmo porque o desenvolvimento ficou muito facilitado com as ferramentas de desenvolvimento que foram usadas. Os objetivos do trabalho, que consistiam inicialmente em apresentar o desenvolvimento de Web Services em dispositivos móveis, foram plenamente alcançados. Mas mais que isso, foi possível utilizar práticas conceituais de orientação objeto, práticas de *Design Patterns*, práticas de análise, construção de diagramas, administração de base de dados, ou seja, foi possível desenvolver uma aplicação corporativa completa, passando por todas as etapas do desenvolvimento.

REFERÊNCIAS BIBLIOGRÁFICAS

CHRISTENSEN, Eric *et* al. **Web Services Description Language (WSDL)**. Disponível em: <http://www.w3.org/TR/wsdl>. Acesso em 28 jun. 2005.

Disponível em: <http://www.inf.br/~bosco/ensino/ine5656/uddi>. Acesso em: 14 ago. 2005.

MATTOS, Érico Tavares de Mattos. **Programação Java Wireless**. *Sine loco*: Digerati Books, 2005.

MITRA, Nilo. **SOAP Version 1.2 Part 0**: Primer. Disponível em: <http://www.w3.org/TR/2003/REC-soap12-part0-20030624>. Acesso em: 15 ago. 2005.

MUCHOW, John W.,Core. **J2ME Tecnologia & MIDP**. *Sine loco*: ,Makron Books,2004.

Sun Java Wireless Toolkit. Disponível em <http://Java.sun.com/products/sjwtoolkit>. Acesso em 22 set. 2005.

XML. Disponível em <http://www.gta.ufrj.br/grad/00_1/miguel/index.html>. Acesso em 05 jun. 2005.

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

TOPLEY, Kim. J2ME is a Nutshell, O'Reilly & Associates. 478p

APÊNDICE

Script da base de dados:

```
CREATE TABLE Produto (
  codigo INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
  tipo VARCHAR(20) NULL,
  descricao VARCHAR(20) NULL,
  valorUnitario FLOAT NULL,
  sabor VARCHAR(20) NULL,
  PRIMARY KEY(codigo)
);

CREATE TABLE Usuario (
  codigo INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
  nome VARCHAR(45) NULL,
  PRIMARY KEY(codigo)
);

CREATE TABLE Cliente (
  codigo INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
  Usuario_codigo INTEGER UNSIGNED NOT NULL,
  nome VARCHAR(45) NULL,
  desconto FLOAT NULL,
  cnpjCpf VARCHAR(20) NULL,
  contato VARCHAR(45) NULL,
  PRIMARY KEY(codigo, Usuario_codigo),
  INDEX Cliente_FKIndex1(Usuario_codigo),
  FOREIGN KEY(Usuario_codigo)
    REFERENCES Usuario(codigo)
      ON DELETE NO ACTION
      ON UPDATE NO ACTION
);

CREATE TABLE Pedido (
  codigo INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
  Cliente_codigo INTEGER UNSIGNED NOT NULL,
  Cliente_Usuario_codigo INTEGER UNSIGNED NOT NULL,
  dataPedido DATETIME NULL,
  dataEntrega DATETIME NULL,
  PRIMARY KEY(codigo, Cliente_codigo, Cliente_Usuario_codigo),
  INDEX Pedido_FKIndex1(Cliente_codigo, Cliente_Usuario_codigo),
  FOREIGN KEY(Cliente_codigo, Cliente_Usuario_codigo)
    REFERENCES Cliente(codigo, Usuario_codigo)
      ON DELETE NO ACTION
      ON UPDATE NO ACTION
);

CREATE TABLE Troca (
  codigo INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
```

```
  Cliente_Usuario_codigo INTEGER UNSIGNED NOT NULL,
  Cliente_codigo INTEGER UNSIGNED NOT NULL,
  Produto_codigo INTEGER UNSIGNED NOT NULL,
  quantidade INTEGER UNSIGNED NULL,
  dataTroca DATETIME NULL,
  valorTroca FLOAT NULL,
  PRIMARY KEY(codigo, Cliente_Usuario_codigo, Cliente_codigo, Produto_codigo),
  INDEX Troca_FKIndex1(Cliente_codigo, Cliente_Usuario_codigo),
  INDEX Troca_FKIndex2(Produto_codigo),
  FOREIGN KEY(Cliente_codigo, Cliente_Usuario_codigo)
    REFERENCES Cliente(codigo, Usuario_codigo)
      ON DELETE NO ACTION
      ON UPDATE NO ACTION,
  FOREIGN KEY(Produto_codigo)
    REFERENCES Produto(codigo)
      ON DELETE NO ACTION
      ON UPDATE NO ACTION
);

CREATE TABLE Item (
  Produto_codigo INTEGER UNSIGNED NOT NULL,
  Pedido_codigo INTEGER UNSIGNED NOT NULL,
  Pedido_Cliente_codigo INTEGER UNSIGNED NOT NULL,
  Pedido_Cliente_Usuario_codigo INTEGER UNSIGNED NOT NULL,
  quantidade INTEGER UNSIGNED NOT NULL,
  valor FLOAT NULL,
  valorTotal FLOAT NULL,
  PRIMARY KEY(Produto_codigo, Pedido_codigo, Pedido_Cliente_codigo,
Pedido_Cliente_Usuario_codigo),
  INDEX Item_FKIndex1(Produto_codigo),
  INDEX Item_FKIndex2(Pedido_codigo, Pedido_Cliente_codigo, Pedido_Cliente_Usuario_codigo),
  FOREIGN KEY(Produto_codigo)
    REFERENCES Produto(codigo)
      ON DELETE NO ACTION
      ON UPDATE NO ACTION,
  FOREIGN KEY(Pedido_codigo, Pedido_Cliente_codigo, Pedido_Cliente_Usuario_codigo)
    REFERENCES Pedido(codigo, Cliente_codigo, Cliente_Usuario_codigo)
      ON DELETE NO ACTION
      ON UPDATE NO ACTION
);

CREATE TABLE Recebimento (
  codigo INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
  Cliente_Usuario_codigo INTEGER UNSIGNED NOT NULL,
  Cliente_codigo INTEGER UNSIGNED NOT NULL,
  Produto_codigo INTEGER UNSIGNED NOT NULL,
  dataRecebimento DATETIME NULL,
  quantidade INTEGER UNSIGNED NULL,
  PRIMARY KEY(codigo, Cliente_Usuario_codigo, Cliente_codigo, Produto_codigo),
  INDEX Recebimento_FKIndex1(Cliente_codigo, Cliente_Usuario_codigo),
  INDEX Recebimento_FKIndex2(Produto_codigo),
  FOREIGN KEY(Cliente_codigo, Cliente_Usuario_codigo)
    REFERENCES Cliente(codigo, Usuario_codigo)
      ON DELETE NO ACTION
      ON UPDATE NO ACTION,
  FOREIGN KEY(Produto_codigo)
    REFERENCES Produto(codigo)
      ON DELETE NO ACTION
      ON UPDATE NO ACTION
);
```

```
CREATE TABLE Telefone (
  Cliente_codigo INTEGER UNSIGNED NOT NULL,
  Cliente_Usuario_codigo INTEGER UNSIGNED NOT NULL,
  numero INTEGER UNSIGNED NOT NULL,
  tipo VARCHAR(10) NULL,
  descricao VARCHAR(45) NULL,
  ddd INTEGER UNSIGNED NULL,
  PRIMARY KEY(Cliente_codigo, Cliente_Usuario_codigo),
  INDEX Telefone_FKIndex1(Cliente_codigo, Cliente_Usuario_codigo),
  FOREIGN KEY(Cliente_codigo, Cliente_Usuario_codigo)
    REFERENCES Cliente(codigo, Usuario_codigo)
      ON DELETE NO ACTION
      ON UPDATE NO ACTION
);

CREATE TABLE Titulo (
  codigo INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
  Cliente_Usuario_codigo INTEGER UNSIGNED NOT NULL,
  Cliente_codigo INTEGER UNSIGNED NOT NULL,
  saldoDevedor FLOAT NULL,
  valorPago FLOAT NULL,
  PRIMARY KEY(codigo, Cliente_Usuario_codigo, Cliente_codigo),
  INDEX Titulo_FKIndex1(Cliente_codigo, Cliente_Usuario_codigo),
  FOREIGN KEY(Cliente_codigo, Cliente_Usuario_codigo)
    REFERENCES Cliente(codigo, Usuario_codigo)
      ON DELETE NO ACTION
      ON UPDATE NO ACTION
);

CREATE TABLE Endereco (
  Cliente_codigo INTEGER UNSIGNED NOT NULL,
  Cliente_Usuario_codigo INTEGER UNSIGNED NOT NULL,
  rua VARCHAR(20) NOT NULL,
  complemento VARCHAR(20) NULL,
  bairro VARCHAR(20) NULL,
  cidade VARCHAR(20) NULL,
  cep VARCHAR(10) NULL,
  tipo VARCHAR(20) NULL,
  PRIMARY KEY(Cliente_codigo, Cliente_Usuario_codigo),
  INDEX Endereco_FKIndex1(Cliente_codigo, Cliente_Usuario_codigo),
  FOREIGN KEY(Cliente_codigo, Cliente_Usuario_codigo)
    REFERENCES Cliente(codigo, Usuario_codigo)
      ON DELETE NO ACTION
      ON UPDATE NO ACTION
);
```