UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Karin Maria Söhnlein

UMA ARQUITETURA BASEADA EM AMBIENTES PARA SISTEMAS DE COMÉRCIO ELETRÔNICO

Dissertação submetida à Universidade Federal de Santa Catarina
como parte dos requisitos para a obtenção do grau de
Mestre em Ciência da Computação

Prof. Dr. Rosvelter João Coelho da Costa
Orientador

Florianópolis, 27 de novembro de 2002

UMA ARQUITETURA BASEADA EM AMBIENTES PARA SISTEMAS DE COMÉRCIO ELETRÔNICO

Karin Maria Söhnlein

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação Área de Concentração Sistemas de Computação, e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Santa Catarina.

_____

Prof. Fernando A. O. Gauthier, Dr.
Coordenador do Curso de
Pós-Graduação em Ciência da Computação

Banca Examinadora

_____

Prof. Rosvelter João Coelho da Costa, Dr.
Presidente da Banca (Orientador)

_____

Prof. João Bosco Mangueira Sobral, Dr.
Membro da Banca

_____

Prof. Roberto Willrich, Dr.
Membro da Banca

Agradecimentos

Agradeço ao Curso de Pós-Graduação em Ciência da Computação e à Universidade Federal de Santa Catarina pela infra-estrutura e organização que viabilizaram o desenvolvimento deste trabalho.

Agradeço aos professores João Bosco Mangueira Sobral e Roberto Willrich por participarem no julgamento deste trabalho.

Agradeço especialmente ao meu orientador Prof. Rosvelter João Coelho da Costa, pelos valiosos ensinamentos e pela dedicação, paciência e compreensão que demonstrou ao decorrer deste tempo de pesquisa.

Agradeço à minha família, por todo o apoio oferecido ao decorrer deste trabalho.

Este trabalho é dedicado à minha mãe, com certeza a grande incentivadora deste trabalho, que sempre acreditou no meu sucesso.

Resumo

Este trabalho explora uma abordagem de ambientes para o desenvolvimento de sistemas de vendas *online* utilizando a tecnologia Servlets/JSP. A primeira parte deste trabalho é dedicada a um estudo de caso compreendendo a análise de um sistema de vendas *online* para produtos relacionados à construção civil denominado Sistema de Vendas *Online* Casarão. A segunda, dedicou-se ao desenvolvimento de um projeto de software contemplando as necessidades do sistema proposto na primeira etapa. Esse estudo tem como objetivo principal a identificação de conceitos arquiteturais para o projeto de sistemas de vendas *online*. Especificamente, buscou-se, de um lado, esclarecer melhor a problemática envolvida e, de outro, evidenciar uma noção de *ambientes* como um possível candidato a conceito arquitetural para sistemas de vendas *online* onde os ambientes são caracterizados conforme ambientes de uma loja real.

Palavras-chave: comércio eletrônico, arquitetura de software, Servlets/JSP.

Abstract

This work explores a treatment of ambients for the development of online sale systems using the Servlets/JSP technology. The first part of this work is dedicated to a case study that encompasses the analysis of an online sale system related to building supplies called Casarão Online Sale System. The second part is dedicated to the development of a software project that attends the funcionalities described in the first part. The goal of this study is to identify an architectural concept for online sale systems. On the one hand this work tries to shed light upon the related problems. On the other hand it tries to highlight a notion of ambients as a possible candidate to an architectural concept for online sale systems, where the ambients are characterized as the ambients of a real life store.

Keywords: e-commerce, software architecture, Servlets/JSP.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Acrônimos

ADL	*Architecture Definition Language*
API	*Application Program Interface*
ASP	*Active Server Pages*
CGI	*Common Gateway Interface*
CORBA	*Common Object Request Broker Architecture*
DNS	*Domain Name Service*
FTP	*File Transfer Protocol*
HTML	*HyperText Markup Language*
HTTP	*HyperText Transfer Protocol*
IP	*Internet Protocol*
ISO	*International Standard Organization*
JDBC	*Java Database Connectivity*
JSP	*JavaServer Pages*
JVM	*Java Virtual Machine*
LAN	*Local Area Network*
MAN	*Metropolitan Area Network*
MVC	*Model/View/Controller*
OSI	*Open Systems Interconnection*
PHP	*Hypertext Preprocessor*
RMI	*Remote Method Invocation*
SGML	*Standard Generalized Markup Language*
SNMP	*Simple Network Management Protocol*
TCP	*Transmission Control Protocol*
UDP	*User Datagram Protocol*
UML	*Unified Modeling Language*
URL	*Uniform Resource Locator*
WAN	*Wide Area Network*
WML	*Wireless Markup Language*
WWW	*World Wide Web*
XML	*Extensible Markup Language*

Capítulo 1

Introdução

O avanço tecnológico na área de redes de computadores nos últimos anos, vem permitindo que a Internet ocupe cada vez mais espaço como meio de comunicação entre pessoas nas mais diversas finalidades. Mais do que um simples meio de comunicação, a introdução desta tecnologia na área comercial tem exercido influências decisivas na forma pela qual um negócio é conduzido. A Internet possibilitou, por exemplo, que as empresas pudessem anunciar seus produtos globalmente e que as tarefas a serem executadas dentro de uma mesma organização pudessem ser atendidas de forma colaborativa por funcionários fisicamente dispersos. Esta nova realidade possibilitou o surgimento de empresas virtuais e incentivou o desenvolvimento de tecnologias para viabilizar o comércio eletrônico, ou seja, a realização de um negócio utilizando a Internet como meio de comunicação. Como exemplo, pode-se citar a empresa norte-americana de revenda de livros Amazon [http://www.amazon.com].

As tecnologias de comércio eletrônico têm como objetivo atender a três categorias diferentes de negócio: comércio orientado ao consumidor (*bussiness-to-consumer* ou B2C), comércio entre empresas (*bussiness-to-bussiness* ou B2B) e comércio intra-organizacional [ZWA98]. Dentro da primeira categoria, o setor de vendas de produtos tem explorado com bastante sucesso a utilização desta tecnologia. A própria natureza do negócio de vendas, que consiste basicamente na interação entre dois atores - vendedor e cliente, somado ao fato da Internet ser uma tecnologia multimídia, atraiu atenção especial para o desenvolvimento de sistemas de vendas de produtos pela Internet.

Os sistemas de vendas *online*, como vêm freqüentemente sendo chamados, envolvem sempre algum tipo de transação financeira e têm como principal objetivo informatizar o processo de vendas, desde a apresentação do produto até a conclusão de sua aquisição, utilizando a Internet como meio de comunicação para troca de informações entre vendedor e cliente. Uma característica típica de sistemas de vendas *online* é oferecer funcionalidades ao usuário que se aproximem o mais fielmente possível ao comportamento de um cliente dentro de uma loja no mundo real.

Dentro deste contexto, o estudo apresentado neste trabalho teve como objetivo principal a identificação de conceitos arquiteturais para o projeto de sistemas de vendas *online*. O projeto apresentado neste trabalho tem na noção de *ambientes* o seu suporte intuitivo principal, onde um ambiente é definido o mais próximo possível de um ambiente real de uma loja. Especificamente, buscou-se evidenciar a noção destes *ambientes* como um possível candidato a conceito arquitetural para este tipo de sistema.

Para disponibilizar e manipular as várias informações necessárias a um processo de vendas *online*, a abrangência de um sistema deste tipo não se restringe apenas ao ambiente de interação entre cliente e setor de vendas, mas envolve também a troca de informações com outros usuários do sistema como setor de estoque, empresa de cartão de crédito, setor de expedição, etc. Sendo assim, o grau de complexidade de um sistema deste tipo pode ser bastante alto, dependendo da abrangência do sistema e/ou dos tipos de informações trocadas com os usuários.

Projetos consagrados de software fundamentam-se em conceitos arquiteturais que, por sua vez, determinam toda uma tecnologia. Por exemplo, monitores [HOA74] nos sistemas operacionais e a álgebra relacional nos bancos de dados [OZS99]. Para o caso dos sistemas de comércio eletrônico, ainda não há conceitos arquiteturais bem definidos. O projeto de tais sistemas busca atender apenas às necessidades funcionais impostas pelo modelo usuário e são fortemente influenciados pela tecnologia de software disponível no servidor Web para o desenvolvimento de aplicações Web de uma forma geral. As aplicações existentes hoje utilizam as mais diferentes tecnologias, como por exemplo, a

tecnologia de agentes muito utilizada para simular mercados virtuais [DAS02, PET02] ou ainda a tecnologia de realidade virtual [WES98] utilizada por algumas lojas na construção de seu sítio.

O estudo apresentado nesse manuscrito de dissertação de mestrado foi desenvolvido em duas etapas. A primeira, consagrou-se à análise das principais funcionalidades de sistemas de vendas *online* apresentando um estudo de caso que incorpora as principais características deste tipo de sistema. A segunda, dedicou-se ao desenvolvimento de um projeto de software contemplando as funcionalidades que foram apresentadas na primeira etapa. O projeto proposto para o estudo de caso propõe para o servidor uma arquitetura baseada em ambientes definidos o mais próximo possível de um ambiente real de uma loja.

A arquitetura proposta neste trabalho segue o modelo cliente/servidor em três camadas. Para a camada de apresentação é utilizada a tecnologia Servlets/JSP [SUN02]. A camada de lógica da aplicação é composta por um conjunto de ambientes onde cada ambiente disponibiliza um conjunto de recursos a serem utilizados pelos servlets e páginas JSP. Neste contexto, os servlets assumem o papel de representantes do usuário, que visitam estes ambientes e executam ações utilizando os recursos neles disponíveis. As páginas JSP são encarregadas de apresentar a visão que o usuário teria ao término da execução de cada ação. Os recursos disponíveis nos ambientes muitas vezes necessitam manipular informações persistentes. A camada de acesso à base de dados é composta por um conjunto de classes que oferece os serviços relacionados à persistência de informações para estes recursos. Esta camada utiliza a tecnologia JDBC.

Este trabalho está organizado da seguinte forma:

- No Capítulo 2, apresenta-se a tecnologia Servlets.
- No Capítulo 3, apresenta-se a tecnologia JSP.

- No Capítulo 4, apresenta-se a análise de um sistema de vendas pela Internet para uma loja de produtos relacionados à construção civil. Este sistema foi concebido para este trabalho visando fornecer um modelo que incorpore as principais características dos sistemas de vendas *online*.

- No Capítulo 5, apresenta-se um projeto de software para o sistema do capítulo 4 que propõe a utilização de uma arquitetura baseada em ambientes.

- O Capítulo 6, apresenta a conclusão deste trabalho e algumas propostas de continuação.

Este trabalho possui dois anexos. O ANEXO 1 apresenta os principais aspectos da rede Internet e da Web e o ANEXO 2 apresenta os códigos fonte na linguagem Java de algumas classes utilizadas neste trabalho.

Embora a aplicação estudada por esta dissertação foi desenvolvida utilizando a plataforma Java, apenas duas vertentes dessa tecnologia, Servlets, no Cap. 2, e JavaServer Pages, no Cap. 3, será detalhado neste manuscrito. Grande parte do detalhamento omitido, em particular, a sintaxe e a semântica da linguagem Java, poderá ser obtido, por exemplo, no tutorial sobre Java disponibilizado no sítio da *Sun Microsystems* [JAV02].

Capítulo 2

Servelets

2.1 Introdução

Este capítulo é dedicado à apresentação da tecnologia Servlets. Esta tecnologia fornece um conjunto de facilidades para o desenvolvimento de servidores Web, em particular, a parte relacionada à geração de conteúdo dinâmico em resposta a uma solicitação HTTP do cliente.

Este capítulo apresenta inicialmente uma introdução à plataforma Java. A seguir são apresentas as principais características da tecnologia Servlets e é feita uma comparação desta tecnologia com a tecnologia CGI (AXEXO 1 Seção 3.4). Por último, é apresentada a conclusão do capítulo.

2.2 A plataforma Java

Java é uma linguagem criada pela *Sun Microsystems* originada de um projeto para programar dispositivos eletrônicos inteligentes em 1991. Atualmente é utilizada, entre outras, para criar páginas Web com conteúdo dinâmico e interativo, desenvolver sistemas empresariais de larga escala, desenvolver aplicações para dispositivos eletrônicos como telefones celulares e ainda para estender as funcionalidades de um servidor Web [DEI01].

A linguagem Java é orientada a objetos e é independente de plataforma. Ao compilar um programa Java, o compilador Java gera um arquivo com extensão `.class` que contém instruções chamadas *bytecodes*. Estas instruções são compostas por um byte de operação e bytes opcionais de operandos. Em tempo de execução, uma entidade de

software chamada Máquina Virtual Java (JVM) interpreta as instruções contidas neste arquivo e cria o código executável compatível com a máquina real. Desta forma, um arquivo `.class` pode ser executado em qualquer plataforma Java [JES00].

As aplicações desenvolvidas para a Internet utilizam recursos que estão fisicamente distribuídos pela rede e que podem eventualmente ser compartilhados com outras aplicações. Estes recursos incluem, entre outros, dados, processos, dispositivos de I/O, objetos e arquivos. Uma aplicação distribuída desenvolvida sob o paradigma de orientação a objetos pode ser vista como um conjunto de objetos distribuídos pela rede que cooperam para a realização de uma determinada tarefa. Estas aplicações utilizam os serviços de software de rede e os elementos Web descritos na seção anterior.

Visando facilitar o desenvolvimento de tais aplicações, a plataforma Java fornece uma arcabouço de técnicas e mecanismos para que muitos detalhes da tecnologia associada ao funcionamento da Internet fiquem transparentes ao programador. Estes recursos incluem, outre outros, as tecnologias *Servlets*, *JavaServer Pages* (JSP), *Remote Method Invocation* (RMI), *Sockets* e *Common Object Request Broker Architecture* (CORBA) [JAV02].

2.2.1 O servidor Tomcat v.4

Tomcat v.4 é uma aplicação servidor desenvolvido sob a licença do grupo *Apache Software Foundation* [http://jakarta.apache.org]. Este software é um *container* que implementa as especificações da empresa *Sun Microsystems* [http://www.sun.com] referentes às tecnologias Java Servlet v. 2.3 [SER01] e JavaServer Pages v. 1.2. [JSP01]. Esta aplicação pode ser utilizada como um servidor Web independente ou pode funcionar como um *container* que estende as funcionalidades de outro servidor Web. O servidor Tomcat v.4 está disponível para as plataformas WINDOWS e UNIX no endereço: http://jakarta.apache.org/tomcat/index.html.

2.3 Arquitetura dos Servlets

O conjunto de servlets que compõe uma aplicação é gerenciado por um componente denominado *Servlet Container*, que estende a funcionalidade de um servidor Web. Ele pode ser instalado como parte do servidor ou pode ser um componente externo acoplado a um servidor. O serviço de troca de mensagens entre cliente e servlet é feito através da interface do *Servlet Container*. Esta interface aceita o protocolo HTTP e permite também que sejam impostas restrições de segurança.

A Fig. 2.1 ilustra um processo de solicitação/resposta entre um cliente e uma aplicação servidor que utiliza a arquitetura de servlets. Os seguintes eventos estão enumerados na figura [SER01]:

(1) O cliente faz uma solicitação HTTP ao servidor Web que é redirecionada ao *Servlet Container.*
(2) O *Servlet Container* escolhe o servlet a ser acionado consultando um arquivo de configuração.
(3) O Servlet 2 é selecionado e recebe a solicitação. Para respondê-la ele pode eventualmente consultar um banco de dados e fazer os processamentos necessários.
(4) O Servlet 2 responde à solicitação.
(5) O *Servlet Container* envia a resposta ao cliente.

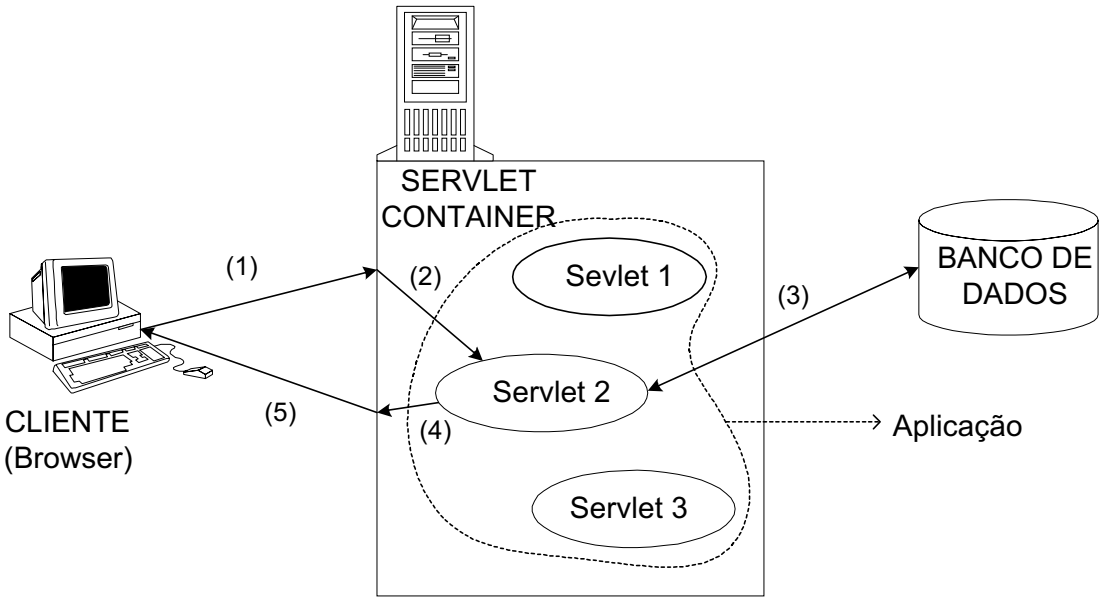

Fig. 2.1 - Comunicação entre cliente e servidor baseada na arquitetura de servlets.

2.4 Contexto de uma aplicação

Uma aplicação é composta por um conjunto de servlets que precisam eventualmente compartilhar informações sobre o contexto da aplicação ao qual pertencem. Estas informações são disponibilizadas no objeto `ServletContext`, disponibilizado pelo *Servlet Container*, onde qualquer servlet deste conjunto pode consultar, adicionar ou remover informações relevantes a outros servlets que participam da mesma aplicação.

2.5 Sessão e Gerenciamento de Sessão

Ao utilizar uma aplicação via Web, um cliente segue um protocolo: primeiro acessa a aplicação, depois troca informações com ela e depois sai da aplicação. Do ponto de vista da aplicação, o conjunto de interações entre cliente e aplicação seguindo este protocolo define uma sessão. Para cada sessão, o *Servlet Container* disponibiliza um objeto do tipo

HttpSession, que é unicamente identificado através de um SessionID. O objeto HttpSession permite que qualquer servlet que seja executado durante uma sessão possa nele consultar, adicionar ou remover informações relevantes a outros servlets que sejam executados na mesma sessão. Para atingir este objetivo, o SessionId é armazenado como um *cookie* no cliente [DEI01]. A cada solicitação, este *cookie* é recuperado e as informações sobre a sessão são disponibilizadas. Caso o *browser* cliente não aceite *cookies*, o sistema oferece uma opção chamada URL Rewriting, onde uma sessão é identificada através da composição da URL com uma SessionID. Neste caso, a SessionID é passada como parâmetro da URL em todas as solicitações. Este parâmetro tem o nome jssesionId.

A vida de um objeto HttpSession no sistema é definida através de um tempo máximo de persistência (*default* 30 minutos) ou pode ser interrompida através do método invalidate(). É importante observar aqui que as solicitações feitas à aplicação utilizam o protocolo HTTP que é *stateless*, ou seja, não tem como identificar o estado do cliente a cada interação. Sendo assim, o gerenciamento de sessão deve ficar a cargo da aplicação. Uma maneira de fazer este gerenciamento é guardar o estado do cliente no próprio objeto HttpSession. Outra forma seria armazenar o estado do cliente em um banco de dados associado à sua SessionId [DAV00].

2.6 Solicitações

Ao fazer uma solicitação, um cliente envia dados a um servlet que, por sua vez, são armazenados em um objeto do tipo HttpServletRequest, criado a cada solicitação. O processamento de uma solicitação pode envolver um conjunto de servlets que eventualmente tenham que compartilhar dados. Todos os servlets envolvidos no processamento de uma única solicitação podem consultar, adicionar ou modificar dados do objeto HttpServletRequest que sejam relevantes a outros servlets executando a mesma solicitação. Através de um objeto do tipo RequestDispatcher, um servlet

pode redirecionar uma solicitação recebida para outro servlet. O fluxo de execução do servlet é interrompido assim que a solicitação é redirecionada.

2.7 Concorrência e sincronização de objetos

O *Servlet Container* cria uma única instância de cada servlet, exceto no caso do servlet pertencer a um ambiente distribuído. Para cada instância, pode-se ter *threads* concorrentes acessando o mesmo serviço. Para garantir que um único *thread* por vez acesse o serviço de um servlet, este servlet deve realizar a interface `SingleThreadModel`. Neste caso, o *Servlet Container* pode escolher entre sincronizar os acessos a esta única instância ou então criar um *pool* de instâncias e despachar cada solicitação para uma instância diferente [SER01]. Entretanto o acesso às classes disponíveis para vários servlets, pertencentes a um escopo externo ao servlet, devem ser sincronizadas, se necessário, utilizando os recursos disponibilizados na API `java.net`.

2.8 Estrutura de um servlet

Toda classe que tem a funcionalidade de um servlet realiza a interface `Servlet`. Os métodos definidos nesta interface são invocados pelo *Servlet Container* e compreendem os seguintes métodos [DEI01]:

a) `ServletConfig getServletConfig()`

   Retorna um objeto que tem as informações básicas sobre as configurações do servlet, parâmetros de inicialização e informações sobre o ambiente onde o servlet atua, por exemplo, se o servlet pertence a um ambiente distribuído ou não. Este objeto é fornecido pelo servidor Web.

b) `void init(ServletConfig configuração)`

É invocado no momento da instanciação de uma classe servlet. Para isso, ele utiliza as informações contidas no objeto *configuração*. Neste método devem ser declaradas todas a tarefas a serem realizadas uma vez só durante a vida do servlet. Muitas aplicações abrem as conexões com um banco de dados neste método. Neste caso, o servlet vai se comunicar com o banco de dados sempre através da mesma conexão.

c) `void service(ServletRequest request, ServletResponse response)`

É invocado ao receber uma solicitação do cliente. Este método é implementada toda a computação que descreve a funcionalidade do servlet. O *Servlet Container* permite acessos concorrentes a este método caso a interface `SingleThreadModel` não seja realizada pelo servlet.

d) `void destroy()`

É invocado antes de terminar um servlet. Este método geralmente libera os recursos alocados no método `init()`, por exemplo, o encerramento da conexão com um banco de dados. O *Servlet Container* só destrói as instâncias dos servlets que não estão sendo utilizadas por nenhum *thread*.

Para que uma classe tenha a funcionalidade de um servlet ela deve estender a classe `HttpServlet` do pacote `javax.servlet.http` ou a classe `GenericServlet` do pacote `javax.servlet`. Ambas realizam a interface `Servlet`.

2.9 Classes utilizadas para solicitações e respostas

Ao ser submetido, um formulário HTML envia uma solicitação a um recurso definido no parâmetro `ACTION`. Os tipos de solicitações mais comuns são `GET` e `POST`. Para solicitar o serviço de um servlet, o parâmetro `ACTION` deve ter o endereço URL deste

servlet. Ao receber uma solicitação um servlet executa as computações necessárias e gera como resposta uma página HTML.

A classe `HttpServlet` é muito utilizada pelas aplicações Web. Ela disponibiliza dois métodos principais para responder uma solicitação HTTP, que são invocados automaticamente pelos seguintes métodos:

a) `doGet(HttpServletRequest request, HttpServletResponse response)`
   Responde à solicitações `GET`.

b) `doPost(HttpServletRequest request, HttpServletResponse response)`
   Responde à solicitações `POST`.

O primeiro parâmetro destes dois métodos contém as informações sobre o serviço solicitado e o segundo contém a resposta ao serviço.

2.10 Servlets x CGI

Em termos de funcionalidade a tecnologia Servlets pode ser vista como uma tecnologia intermediária entre o modelo CGI (ANEXO 1 Seção 3.4) e extensões proprietárias para servidores Web tais como o *Nestscape Server API* (NSAPI) ou *Apache Modules*. Entretanto, a tecnologia Servlets apresenta as seguintes vantagens [SER01, LUB98]:

a) Os servlets são extensões diretas de um servidor Web na forma de objetos que são carregados pelo servidor à medida que são solicitados. Cada servlet presente no sistema trata um determinado tipo de solicitação. Quando ocorre uma solicitação, é criado um *thread* associado ao servlet responsável pelo seu tratamento. No caso do modelo CGI, um processo inteiro é criado a cada solicitação. Uma vez que os

threads criados são computacionalmente mais leves que processos, esta diferença faz com que o desempenho dos servlets seja maior que a dos *scripts* CGI em termos de tempo de resposta e utilização de recursos.

b) Os servlets são objetos que continuam ativos mesmo após o envio da resposta ao cliente. Desta forma, ele pode armazenar informações persistentes entre solicitações tal como a conexão a uma base de dados.

c) A tecnologia Servlets herda vantagens da plataforma Java tais como independência de plataforma e acesso aos recursos disponibilizados pelas várias APIs de Java.

2.11 Exemplo

Para entender melhor a mecânica associada aos servlets, apresenta-se um exemplo onde um usuário preenche o formulário apresentado na Fig. 4 do ANEXO 1. Quando o formulário é submetido, os dados são enviados a um servlet chamado `RecebeDados`. Este servlet cria um objeto com estes dados chamado `cartao` (classe `CartaoCliente`) e envia este objeto para ser aprovado pelo objeto `credito` (classe `DepCredito`). O objeto `credito` preenche o atributo `estado` do objeto `cartao` com *true* ou *false*, indicando a sua aprovação ou não. Conforme o caso, o servlet `RecebeDados` constrói dinamicamente uma página personalizada e envia ao cliente. A Fig. 2.2 ilustra este processo. Os códigos das classes `RecebeDados.java`, `CartaoCliente.java` e `DepCredito.java` encontram-se no ANEXO 2.

A única alteração a ser feita no código da página HTML apresentado no exemplo do ANEXO 1 Seção 3.4 é a invocação do recurso no parâmetro da instrução de formulário. O recurso solicitado não será mais um programa CGI, mas sim o servlet `RecebeDados`, ou seja:

```
... <FORM  METHOD="POST" ACTION="../servlet/RecebeDados"> ...
```

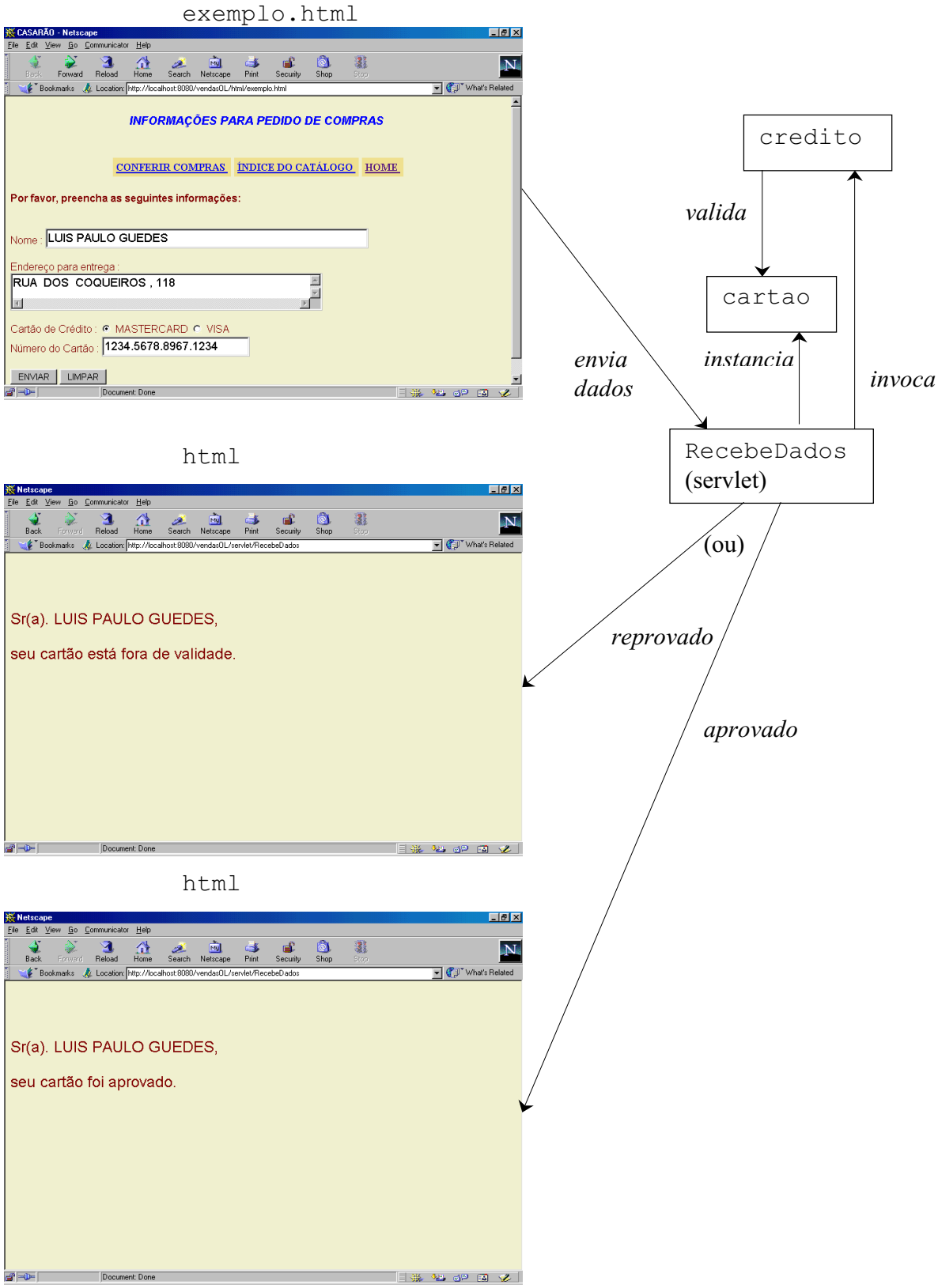

Fig. 2.2 - Estrutura de solicitação/resposta utilizando a tecnologia Servlets

2.12 Conclusão

A tecnologia Servlets permite gerar conteúdo dinâmico como resposta a uma solicitação de um cliente. O conjunto de servlets de uma aplicação é administrado por um *container* que provê algumas funcionalidades aos servlets. Entre elas podemos destacar o gerenciamento de sessão e o gerenciamento de solicitações que simplificam o desenvolvimento de aplicações Web. A tecnologia Servlets pode ser enquadrada, em termos de funcionalidade, como uma tecnologia intermediária entre as tecnologias CGI e as extensões proprietárias de servidores Web [SER01].

Entretanto, como o conteúdo dinâmico da página de resposta é gerado através de um servlet, qualquer atualização, ainda que seja na parte estática da página, requer um especialista em programação para executar a modificação, recompilar a classe e disponibilizá-la na aplicação [HUN00].

A tecnologia JSP, apresentada no próximo capítulo, fornece uma solução a este problema.

Capítulo 3

JSP – JavaServer Pages

3.1 Introdução

A tecnologia *JavaServer Pages* (JSP) permite, a exemplo dos servlets, a geração de conteúdo dinâmico da resposta, mas através de uma especificação textual, permitindo assim que projetistas de Web tenham maior autonomia na criação e modificação de páginas Web. Diferentemente da tecnologia Servlets, entretanto, JSP separa o conteúdo estático da página do conteúdo dinâmico. O conteúdo estático é um molde que pode estar em qualquer formato, por exemplo, HTML (*HyperText Markup Language*), XML (*Extensible Markup Language*) ou WML (*Wireless Markup Language*). A lógica de geração do conteúdo dinâmico fica a cargo de comandos especiais da página JSP. Estes comandos podem conter pedaços de código de programação e/ou acessos a um recurso externo, por exemplo, um componente *JavaBean* [DEI01]. Este pode ser criado e utilizado por uma página JSP através de uma sintaxe especial. Quando alguma modificação no molde de uma página JSP é feita, esta é automaticamente recompilada e disponibilizada no servidor.

A tecnologia JSP é uma extensão da API de Servlets. Sendo assim, existem muitas características comuns entre JSP e Servlets. Assim como os Servlets, as páginas JSP são componentes associados a um *container*. Cada página JSP passa por duas etapas: tradução e solicitação [JSP01]. O *JSP Container* é responsável pela execução da etapa de tradução da página JSP e pelo gerenciamento das solicitações das páginas pelos clientes. O modelo de componentes do JSP é baseado na arquitetura de componentes *JavaBeans* [BEA02]. Um componente *bean* é uma classe Java que segue determinados padrões. Por exemplo, todos os atributos, também chamados de propriedades, devem ser declarados como `private` e só podem ser manipulados externamente através de métodos modificadores e seletores.

A etapa de tradução de uma página JSP ocorre somente quando é feita a primeira requisição desta página. Neste momento é criada e compilada uma classe que realiza a interface `Servlet`. A classe criada tem três métodos principais que são invocados pelo *JSP Container*: `jspInit()`, `_jspService()` e `jspDestroy()`. O método *`jspInit()`* é invocado quando a instância associada à página é criada e o método *`jspDestroy()`* quando a instância é destruída.

A etapa de solicitação ocorre sempre que há uma requisição da página por um cliente. Para cada solicitação, o *JSP Container* invoca o método `_jspService()` sobre a instância associada à página em questão. Este método é responsável por executar as ações necessárias para responder à solicitação [JSP01].

3.2 Arquiteturas baseadas na tecnologia JSP

A tecnologia JSP permite que a geração do conteúdo dinâmico seja toda feita dentro da própria página através da execução de pedaços de código de programação. Entretanto, esta visão vai de encontro aos objetivos de simplificar a criação de páginas Web e de ter uma divisão clara entre conteúdo estático e dinâmico [HUN00]. Para usufruir dos benefícios desta tecnologia dois modelos de arquitetura são muito utilizados: o Modelo I e o Modelo II [SES99, LAV02].

No Modelo I, todas as requisições de serviços da aplicação são recebidas e respondidas por uma página JSP. Ao receber uma solicitação, a página JSP faz o processamento necessário à solicitação e invoca um componente *JavaBean*. Este é responsável por recuperar/inserir os dados necessários em uma base de dados. De posse dos dados necessários, a página compõe a resposta e a envia ao cliente. O problema deste modelo é que, dependendo da complexidade do processamento, teremos ainda uma página cheia de código de programação. A Fig. 3.1 ilustra o Modelo I.

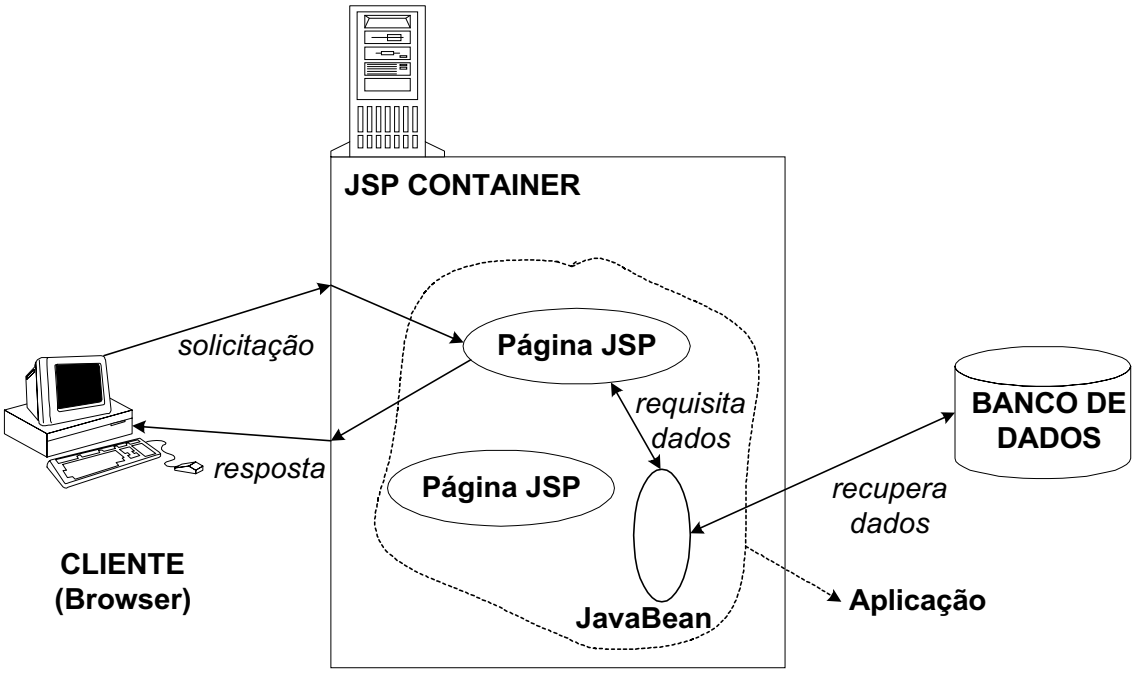

Fig. 3.1 - Modelo I

Para aplicações mais complexas, o Modelo II propõe uma arquitetura baseada no padrão MVC (*Model/View/Controller*) [GAM94]. As requisições são recebidas por um Servlet ou página JSP. Este Servlet ou página JSP assume o papel de *Controller* e é responsável por criar os *JavaBeans* a serem utilizados na resposta, fazer o processamento necessário das informações e invocar a página JSP responsável pela composição e envio da resposta ao cliente. Os *JavaBeans* atualizados pelo *Controller* representam o *Model* do padrão MVC e a página JSP invocada pelo *Controller* assume o papel de *View*. Ela é responsável unicamente pela composição e apresentação da resposta ao cliente. Para isso, ela utiliza os *beans* previamente manipulados pelo *Controller* para associar as informações necessárias ao seu conteúdo estático. Este modelo deixa bastante clara a separação entre conteúdo estático e dinâmico. Enquanto a lógica de geração do conteúdo dinâmico fica a cargo do *Controller*, tipicamente desenvolvido por um especialista em programação, as páginas de apresentação ficam a cargo do *View*, podendo ser facilmente desenvolvidas por *page designers*. A Fig. 3.2 ilustra a arquitetura do Modelo II.

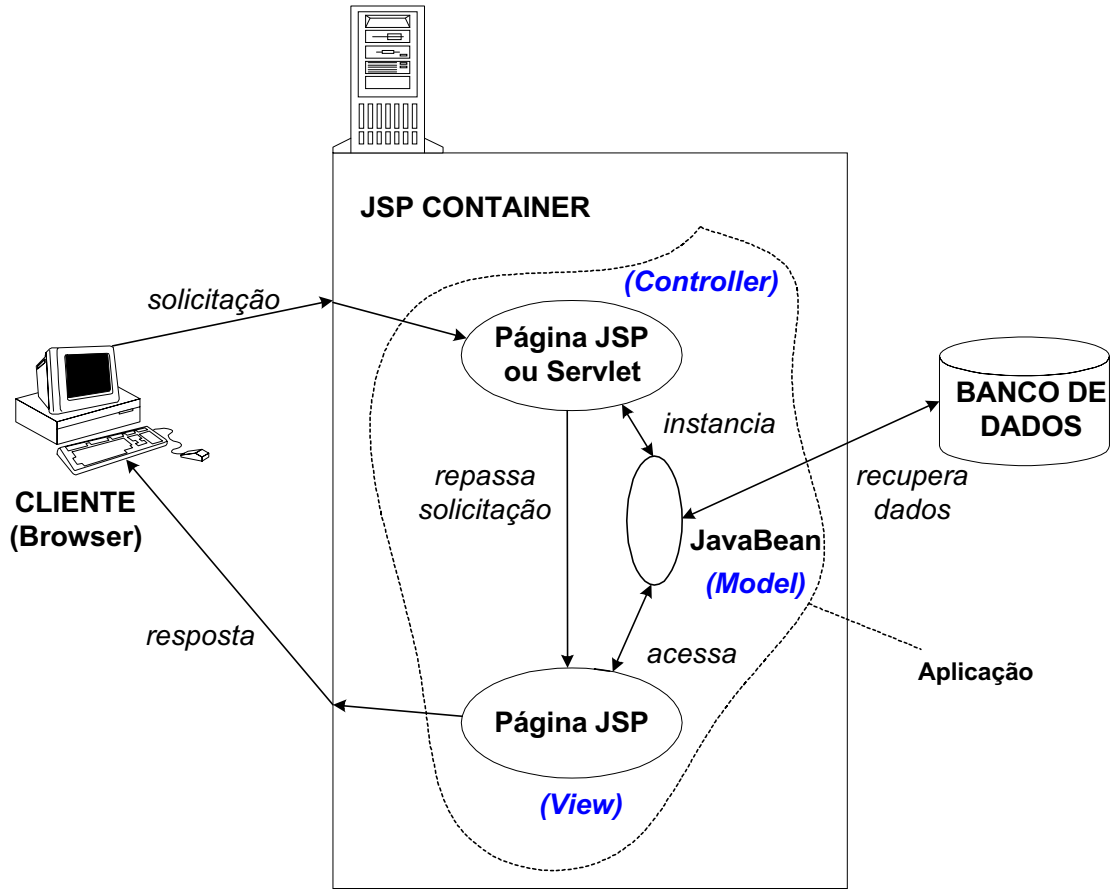

Fig. 3.2 - Modelo II

3.3 Objetos e sincronização de objetos em JSP

A tecnologia JSP permite que objetos sejam instanciados e manipulados por uma página. A visibilidade destes objetos pode ser explicitamente definida no momento de instanciação, através do atributo `scope`. Este atributo pode assumir os seguintes valores [SES01]:

a) `page`

O objeto é acessível apenas pela página em que foi criado. A referência a este objeto é armazenada no objeto `pageContext` e é eliminada quando a página responde à solicitação do cliente ou quando a solicitação é redirecionada.

b) `request`

O objeto é acessível a todas as páginas que respondem à mesma solicitação. A referência a este objeto é armazenada no objeto `request` e é eliminada quando a solicitação é respondida.

c) `session`

O objeto é acessível a todas as páginas que respondem alguma solicitação associada à mesma seção. A referência a este objeto é armazenada no objeto `session` e é eliminada quando o sessão é terminada (*timeout*) ou invalidada.

d) `application`

O objeto é acessível a todas as páginas da aplicação. A referência a este objeto é armazenada no objeto `application` e é eliminada quando o ambiente elimina o objeto `ServletContext`.

Alguns objetos, chamados implícitos, são automaticamente criados pelo *container* e ficam disponíveis para serem acessados por qualquer página JSP. A Tabela 3.1 ilustra quais são estes objetos, sua visibilidade e sua funcionalidade [JSP01].

Objeto	**Visibilidade**	**Descrição**
`pageContext`	página	contém informações de implantação que dizem respeito ao contexto da página (Ex: includes, imports, etc..)
`out`		objeto `JSPWriter` que escreve no stream de saída
`config`		equivalente ao `ServletConfig` (servlets)
`exception`		objeto *Throwable* gerado por um erro e não capturado
`page`		representa a própria página, equivalente ao operador *this*
`response`	solicitação	equivalente ao `HttpServletResponse` (servlets)
`request`	solicitação	equivalente ao `HttpServletRequest` (servlets)
`session`	sessão	equivalente ao `HttpSession` (servlets)
`application`	aplicação	equivalente ao `ServletContext` (servlets)

Tabela 3.1 – Objetos implícitos do JSP.

Como vimos, para cada solicitação o container JSP invoca o método `_jspService()` da página requisitada. Se várias solicitações são enviadas simultaneamente para a mesma página, o *container* despacha cada solicitação em um *thread* diferente. Fica então a cargo da aplicação administrar a sincronização dos acessos aos objetos compartilhados.

A tecnologia JSP oferece um mecanismo de sincronização de páginas equivalente ao mecanismo de sincronização de servlets, acessível através da declaração do diretivo: `<%@ page isThreadSafe="true" %>` no início da página JSP. Neste caso, a classe associada à página JSP na etapa de tradução vai realizar a interface `SingleThreadModel`. O *container* cria várias instâncias desta classe e associa a cada instância uma solicitação. No caso em que muitas solicitações são recebidas ao mesmo tempo, como o número de instâncias é limitado, uma solicitação pode eventualmente ter que aguardar muito tempo para ser atendida. Neste caso, é recomendável que ao invés de sincronizar o acesso à página como um todo através do diretivo acima, seja feita apenas a sincronização dos objetos eventualmente compartilhados por várias páginas através do modificador `syncronized` da API `java.net` [SES00].

3.4 Gerenciamento de sessão

Para cada sessão aberta por um cliente é criado um objeto do tipo `session`. Este objeto tem uma identificação única que é armazenada no *browser* do cliente como um *cookie*. Neste objeto, é possível armazenar informações que podem ser acessadas por qualquer outra página que esteja associada à mesma sessão enquanto a sessão estiver ativa. O objeto `session` é destruído quando a sessão é invalidada ou fechada.

3.5 Sintaxe e semântica de páginas JSP

Uma página JSP é composta de elementos e de um molde. Um elemento é um comando definido entre marcas especiais que é entendido e processado pelo *container*,

enquanto o molde é composto pelo resto da página. Por exemplo, o conjunto de comandos HTML de uma página representa um molde.

Os elementos de uma página JSP são classificados em três tipos: diretivos, *scripts* e ações [SES01].

a) Diretivos

Os diretivos são elementos usados para que a página comunique alguma informação ao JSP Container que será utilizada na fase de tradução. Os tipos de diretivos mais usados são o `page, include e taglib`. Através dos atributos do diretivo `page` pode-se importar classes que serão utilizadas pela página JSP, definir o tamanho de *buffer* da página, definir se os acessos à página devem ser sincronizados, definir a linguagem utilizada pelo *script* e definir se a página pode ter acesso aos objetos da sessão. Utilizando o diretivo `include` é possível incluir um outro arquivo ou página no lugar da página onde se encontra a marca. O diretivo `taglib` permite que sejam criadas marcas personalizadas pelo usuário para serem utilizadas em uma aplicação. Por exemplo, o elemento

```
<%@ page import="java.util.*"  isThreadSafe="true" %>
```

informa ao container que a página vai utilizar recursos do pacote `java.util` e que a classe servlet correspondente à página em questão deve realizar a interface `SingleThreadModel`.

b) *Scripts*

Os elementos *script* são utilizados para criar objetos e fazer as computações que alteram o conteúdo dinâmico da página. Eles são classificados em:

- Declarações

  Permitem declarar variáveis ou métodos que serão utilizados por outro script da página. Por exemplo:

```
<%! String tipo_cartao = "VISA"; %>
```

- Expressões

  São valoradas e seu resultado é inserido na página como um string. Por exemplo:

  ```
  <%= cartao.getNome() %>
  ```

- Comentários

  São utilizados para documentar uma página e não são incluídos na página de resposta ao cliente. Por exemplo:

  ```
  <%-- isto é um comentário -->
  ```

- *Scriplets*

  São pedaços de código que em geral contêm a lógica de geração do conteúdo dinâmico em resposta a uma solicitação. Os *scriplets* também são utilizados para fazer ações sobre objetos ou para executar ações condicionais. Por exemplo, o seguinte *script* constrói uma lista em HTML contendo os elementos do objeto `NOMES`:

  ```
  <%
   out.println("<UL>");
   for (int i=0;  i < NOMES.length();  i++)
        out.println ("<LI>" + NOMES[i]);
       out.println("</UL>");
  %>
  ```

c) Ações

As ações são elementos através dos quais podemos fazer a comunicação entre uma página JSP e recursos externos. As ações podem ser pré-definidas no sistema ou podem ser definidas pelo usuário através do diretivo `taglib`. Entre as ações pré-definidas, as mais utilizadas são as que permitem instanciar e/ou manipular um objeto não implícito, geralmente um componente *JavaBean*, e redirecionar uma solicitação para um servlet ou outra página JSP.

Para acessar um objeto que não é implícito, o JSP oferece a marca `<jsp:useBean>`. Através dos atributos desta marca são informados a classe do objeto, o escopo do objeto e a referência a este objeto. Por exemplo, o elemento:

```
<jsp:useBean id="cartao" class="CartaoCliente" scope="request" />
```

informa ao *container* que o objeto da classe `CartaoCliente` cuja referência é `cartao` vai ser utilizado pela página em questão. O *container* então procura um objeto com esta referência no escopo da solicitação. Caso o objeto não exista, um novo objeto da classe `CartaoCliente` é instanciado e associado à referência `cartao` no escopo `request`. Os valores dos atributos de um objeto são manipulados através dos elementos `<jsp:getProperty>` e `<jsp:setProperty>`.

Uma página JSP pode redirecionar uma solicitação para outra página JSP ou para um Servlet através da marca `<jsp: forward>`. Por exemplo, o elemento:

```
<jsp:forward page="Responde.jsp" >
```

redireciona a solicitação que está sendo processada pela página em questão para a página `responde.jsp`. Ao redirecionar uma solicitação, a execução da página em questão é interrompida. Neste caso, os *beans* eventualmente necessários podem ser passados como parâmetro através de marcas especiais ou acoplados ao objeto `request` referente à solicitação.

3.6 Exemplo

Para ilustrar alguns elementos descritos, é apresentado a seguir uma versão JSP do exemplo apresentado no final do capítulo anterior. A Fig. 3.3 apresenta a estrutura equivalente à Fig. 2.2 do Cap. 2, agora utilizando páginas JSP segundo a arquitetura do Modelo II.

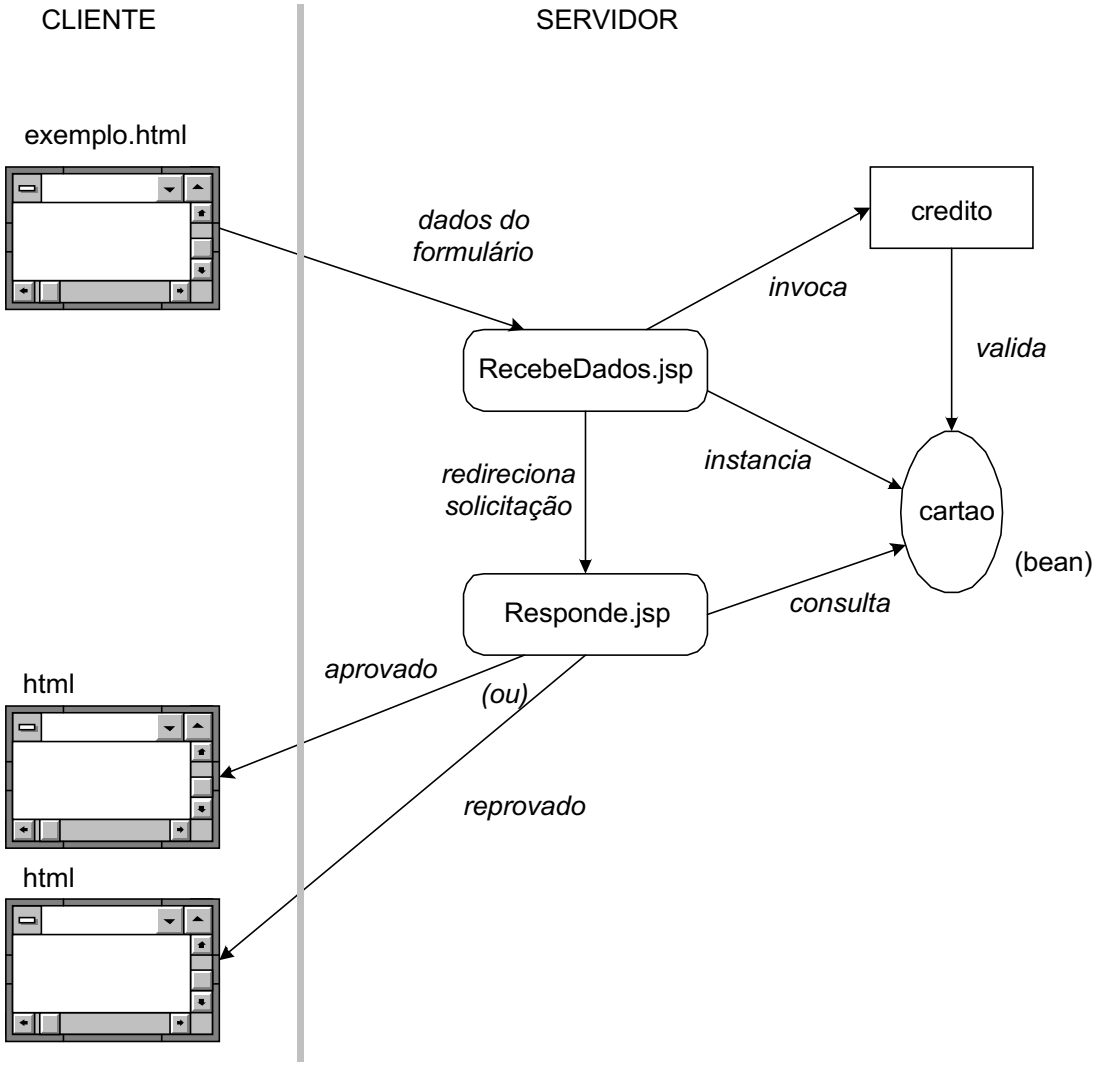

Fig. 3.3 - Estrutura de solicitação/resposta utilizando tecnologia JSP.

As páginas HTML apresentadas ao cliente são idênticas às da Fig. 2.2. A página `RecebeDados.jsp` recebe os dados do formulário `exemplo.html`. Com estes dados ela cria o *bean* `cartao` no escopo da solicitação. Este *bean* é alterado pelo objeto `credito` como no exemplo anterior. A página `RecebeDados.jsp` redireciona a solicitação para a página `Responde.jsp`. Esta página envia uma resposta ao cliente consultando o *bean* `cartao` que foi anexado ao objeto `request`.

A única alteração a ser feita no código da página `exemplo.html` (cf. ANEXO 1 Fig. 3) é novamente a invocação do recurso no parâmetro `ACTION` da instrução de formulário, que agora deve conter a URL da página JSP que recebe os dados do formulário:

```
...
<FORM  METHOD="POST" ACTION="../jsp/RecebeDados.jsp">
...
```

O código fonte das páginas `RecebeDados.jsp` e `Responde.jsp` encontram-se no ANEXO 2. Não há alteração dos códigos das classes `DepCredito.java` e `CartaoCliente.java`.

3.7 Conclusão

A tecnologia JSP estende a tecnologia Servlet visando simplificar a construção de páginas Web permitindo uma separação clara entre o conteúdo estático e dinâmico da página. Devido ao mecanismo utilizado pelo *JSP Container*, o conteúdo estático de uma página pode ser modificado e disponibilizado no servidor sem necessidade de pré-compilação. Desta forma, projetistas de páginas Web adquirem maior independência no trabalho de construção e manutenção destas páginas.

Capítulo 4

O Sistema de Vendas *Online* Casarão

4.1 Introdução

Muitas empresas hoje adotam os sistemas de vendas *Online* como um canal de vendas alternativo para seus produtos. Estas empresas pertencem a diferentes ramos de negócio, como por exemplo, vendas de livros, CDs, materias de construção, aparelhos eletrônicos, produtos alimentícios, etc. Apesar da grande variedade de produtos comercializados, estes sistemas possuem, entre outras, as seguintes características comuns:

a) Os produtos a serem vendidos são apresentados geralmente através de um catálogo. Este catálogo é muitas vezes sub-dividido em categorias de produtos para facilitar a navegação do cliente.
b) Os produtos selecionados pelo cliente durante uma sessão são armazenados em uma estrutura de dados que pode ser consultada e modificada pelo cliente durante a compra.
c) O pagamento pode ser efetuado de várias formas, sendo que o pagamento feito através de cartão de crédito é bastante utilizado devido à facilidade e segurança oferecida pelas tecnologias disponíbilizadas pelas empresas operadoras de cartões de crédito.
d) Um cliente pode optar por fazer parte do cadastro de clientes da empresa para receber informações sobre novos produtos, liquidações, eventuais descontos, etc.

Visando fazer um levantamento mais detalhado das principais características dos sistemas de vendas *Online* este trabalho adotou, a título de estudo de caso, um sistema de

vendas pela Internet para produtos relacionados à construção civil de uma loja fictícea chamada Casarão. Este capítulo apresenta a análise deste sistema, denominado *Sistema de Vendas Online Casarão*, que compreende um conjunto de funcionalidades ligadas à divulgação de produtos e procedimentos de compra conforme as características observadas acima.

O objetivo da análise apresentada neste capítulo é o de fornecer um modelo de concepção para o projeto que será apresentado no próximo capítulo. É importante observar aqui que, dentro desse, as necessidades de segurança de troca de informações inerentes a este tipo de sistema não são consideradas. Além disso, as funcionalidades escolhidas tem como objetivo abranger apenas um conjunto básico de funcionalidades suficientemente expressivas para uma avaliação do projeto que será apresentado. Assume-se também alguns elementos implícitos ao sistema tais como um navegador HTML (por exemplo, o *Netscape Communicator*) e um servidor Web (por exemplo, o servidor Tomcat versão 4.03).

4.2 Aspectos gerais

O sistema Casarão cumpre basicamente dois objetivos:

1) Disponibilizar via Internet um sistema de vendas de produtos da loja, onde qualquer cliente esteja habilitado a:
   - Conhecer os produtos e preços da loja através de um catálogo.
   - Preencher um pedido de compras e efetuar o pagamento via Internet.
   - Acompanhar o estado do seu pedido a qualquer momento do processo.
   - Cadastrar-se como cliente da loja.
   - Poder entrar em contato com a empresa via correio eletrônico ou telefone.

2) Armazenar e processar todas as informações necessárias para atingir o objetivo anterior de forma integrada e consistente entre os seguintes departamentos:

Financeiro, Atendimento ao Cliente, Estoque e Expedição. Deve também estar integrado com o sistema da Cia. de Carões de Crédito.

Para atender os objetivos acima, o sistema disponibiliza um endereço Internet onde o cliente possa consultar um catálogo que contenha descrição e preço de cada produto. O cliente então seleciona os itens que deseja comprar e respectivas quantidades e informa os dados necessários para que o pagamento seja feito através de cartão de crédito. O pagamento é então analisado pela Cia. de Cartões de Crédito. Se o pagamento for recusado, o sistema ignora o pedido. Caso contrário o sistema aprova o pedido, emite informações de lançamento contábil para o Financeiro e disponibiliza o pedido para ser processado pelo Estoque, colocando o pedido em estado de "Aprovado".

No Estoque, os usuários têm acesso aos pedidos que estão em estado de "Aprovado" ou estão à espera de reposição de estoque. Se não existe material disponível para entrega, o sistema coloca o pedido em estado de "Espera". Caso contrário o Estoque separa o material e o sistema coloca o pedido em estado de "Atendido", ou seja, pronto para ser despachado. Caso esta separação de material tenha feito com que a quantidade em estoque deste item fique abaixo de um limite mínimo, o sistema emite uma ordem de compra para este item.

Na Expedição, os usuários têm acesso apenas aos pedidos que foram totalmente atendidos pelo Estoque. Para cada pedido, quando o transporte estiver disponível, o usuário solicita ao sistema a emissão da nota fiscal de saída. O sistema então emite a nota fiscal de saída, envia uma cópia desta ao Financeiro e modifica o pedido para o estado de "Despachado".

O cliente poderá também consultar o estado do seu pedido no sítio da empresa através da Internet a qualquer momento e, eventualmente, entrar em contato com o Atendimento ao Cliente via correio eletrônico ou por telefone para esclarecimento de dúvidas.

As próximas seções serão dedicadas à análise do sistema que compreende:

- Análise dos requisitos do sistema
- Identificação dos casos de uso
- Análise do comportamento do sistema

4.3 Requisitos do sistema

A seguir são identificados os usuários que utilizarão o sistema. Para cada usuário é fornecida uma breve descrição de suas principais atividades no sistema.

1) `Cliente`

   Qualquer pessoa que acesse a página de vendas da loja através da Internet.

   - Entra na loja virtual e escolhe os produtos que quer comprar.
   - Emite um pedido de compras.
   - Consulta o estado de seu(s) pedido(s).
   - Cadastra-se como cliente.
   - Pode fazer algum tipo de solicitação para o Atendimento ao Cliente através de correio eletrônico.

2) `Atendimento ao Cliente`

   Entidade da empresa que atende a dúvidas e registra reclamações do cliente.

   - Consulta pedidos no sistema para tirar dúvidas do cliente.
   - Atende e registra reclamações do cliente.
   - Comunica-se com cliente via correio eletrônico ou telefone.

3) `Estoque`

   Entidade da empresa que é responsável pelas entradas/saídas de material de estoque.

   - Fornece informações de estoque para atualização de catálogo.
   - Consulta pedidos para fazer a separação de material de estoque.
   - Emite ordem de compra para reposição de materiais.

4) Financeiro

Entidade da empresa que, entre outras funções, é responsável pela contabilidade da empresa e manutenção do Livro de Notas Fiscais. Este departamento já possui um sistema próprio e necessita apenas das informações para:

- Registrar as notas fiscais de saída no Livro Fiscal
- Contabilizar os pagamentos aprovados

5) Expedição

Entidade da empresa responsável por despachar o material.

- Emite nota fiscal de saída e envia uma cópia para o Financeiro.
- Despacha os pedidos que estão prontos para entrega.

6) Cia. Cartão de Crédito

Entidade externa à empresa responsável por aceitar/recusar um pagamento.

- Recebe o número do cartão e valor a ser debitado.
- Aceita ou recusa o pagamento.

4.4 Casos de uso e cenários

Um caso de uso é uma interação típica que o usuário tem com o sistema a fim de atingir um objetivo [FOW00]. A própria palavra interação reflete que um caso de uso descreve um sistema reativo, ou seja, um processo que se inicia com um estímulo externo e reage eventualmente devolvendo uma resposta. Um ator é um agente deste estímulo e pode ser identificado com um usuário humano ou outro sistema. A cada estímulo fornecido, o ator espera que o sistema execute tarefas visando atender a um objetivo específico. Cada objetivo identifica um caso de uso, ou seja, uma funcionalidade do sistema [SCH96]. A descrição de casos de uso é geralmente acompanhada por cenários, representando fluxos de execução de tarefas.

Utilizando as informações colhidas na fase de análise de requisitos, primeiramente são definidos os atores do sistema. Depois são identificados os casos de uso a eles associados utilizando os diagramas de casos de uso e cenários.

4.4.1 Atores do sistema Casarão

A Tabela 4.1 identifica cada ator e sua natureza:

ATOR	NATUREZA
Cliente	Grupo de usuários externos
Cia. Cartão de Crédito	Sistema externo à empresa
Estoque	Grupo de funcionários
Financeiro	Sistema existente na empresa
Expedição	Grupo de funcionários
Atendimento ao cliente	Grupo de funcionários

Tabela 4.1 - Atores do sistema Casarão.

4.4.2 Casos de uso do sistema Casarão

Os diagramas apresentados nas Figs. 4.1 e 4.2 identificam os casos de uso para cada ator. Em seguida, os cenários e interfaces gráficas correspondentes.

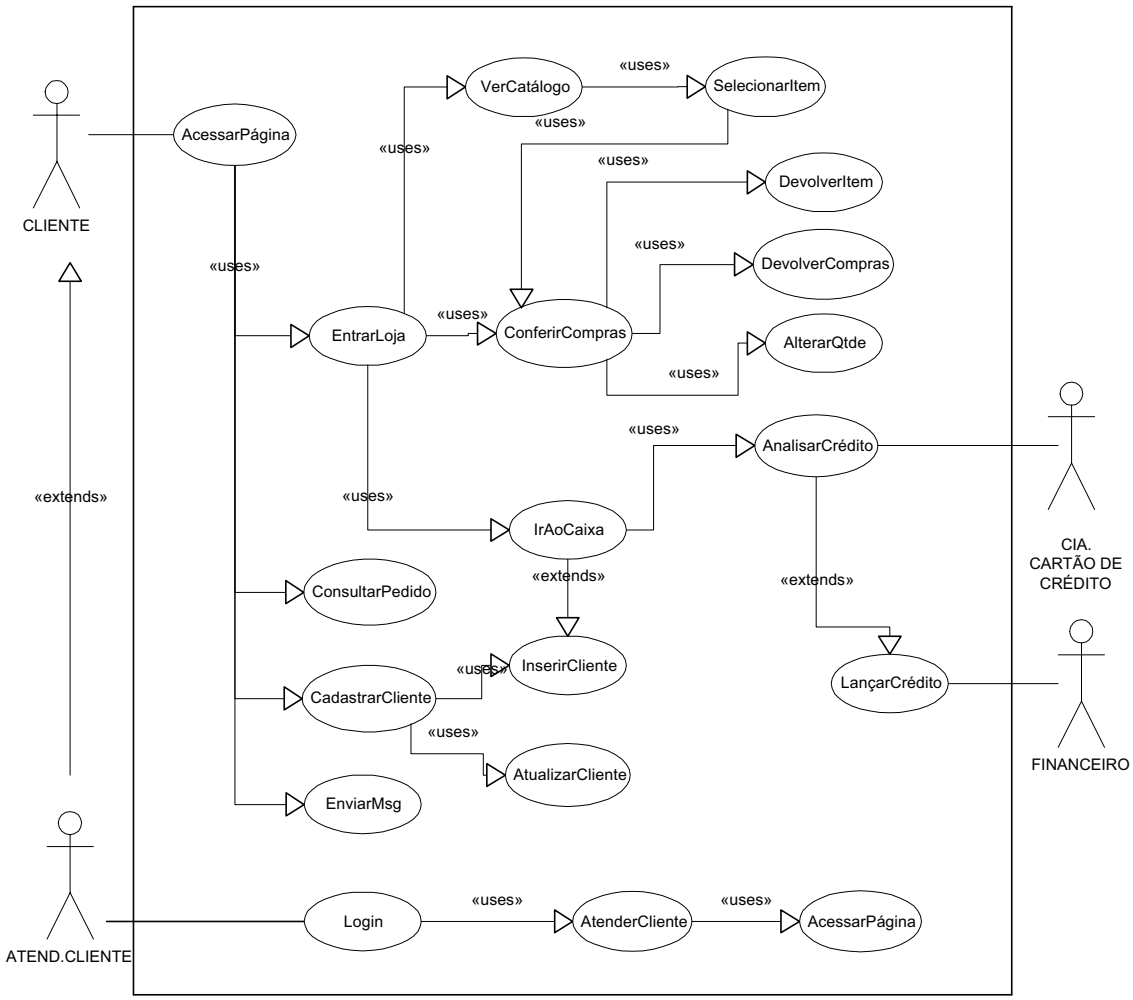

Fig. 4.1 - Diagrama 1: casos de uso relacionados ao ator Cliente.

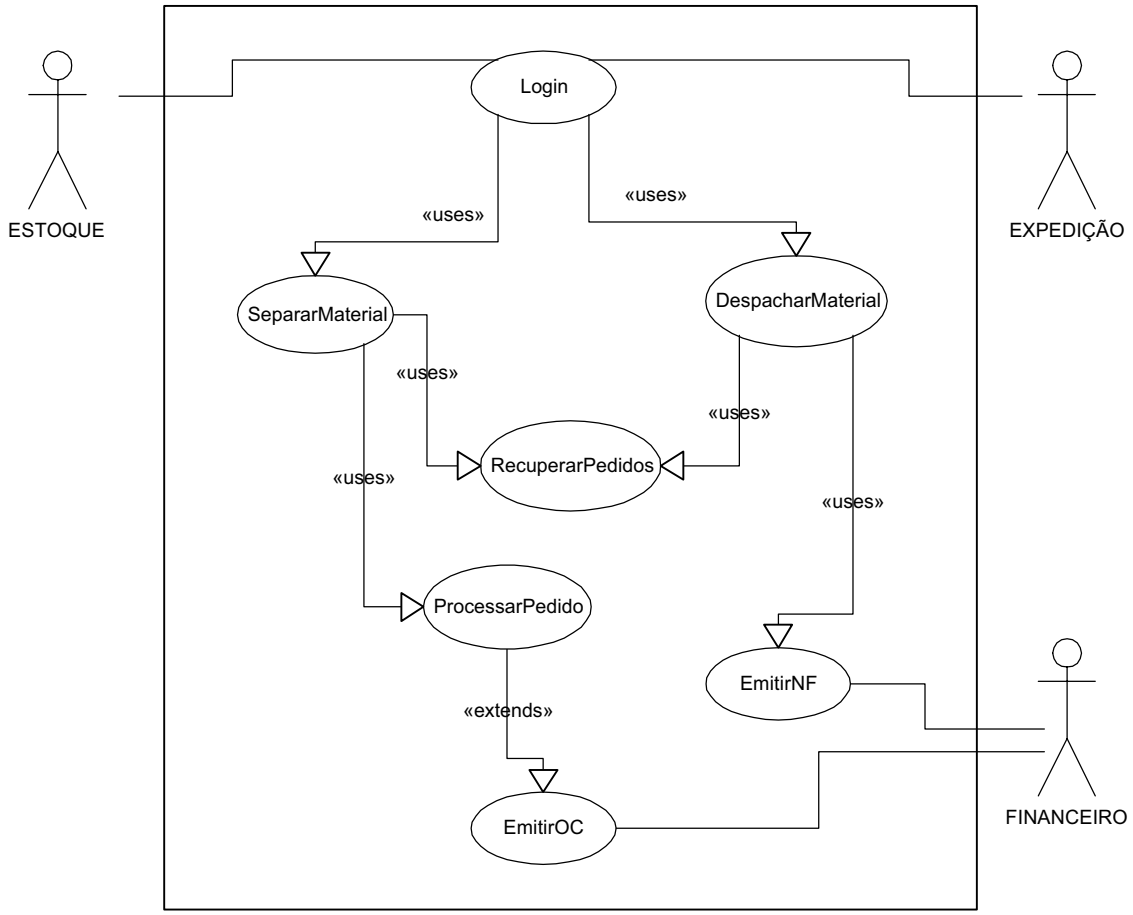

Fig. 4.2 - Diagrama 2: casos de uso não relacionados ao ator Cliente.

Caso de uso `AcessarPágina`

Descreve o acesso de um cliente à página principal do sistema.

Cenário principal

1) O usuário acessa a página principal da empresa pela Internet.
2) O sistema apresenta a página principal.
3) Se o usuário selecionar a opção *Loja Virtual – Faça suas compras...*, o caso de uso `EntrarLoja` é acionado.
4) Se o usuário seleciona a opção *Consulte aqui seus pedidos*, o caso de uso `ConsultarPedido` é acionado.
5) Se o usuário seleciona a opção *Cadastre-se como nosso cliente*, o caso de uso `CadastrarCliente` é acionado.
6) Se o usuário selecionar *Fale Conosco*, o caso de uso `EnviarMsg` é acionado.

Cenários secundários

1) O usuário não faz nenhuma seleção e sai do sítio.

Interfaces Gráficas

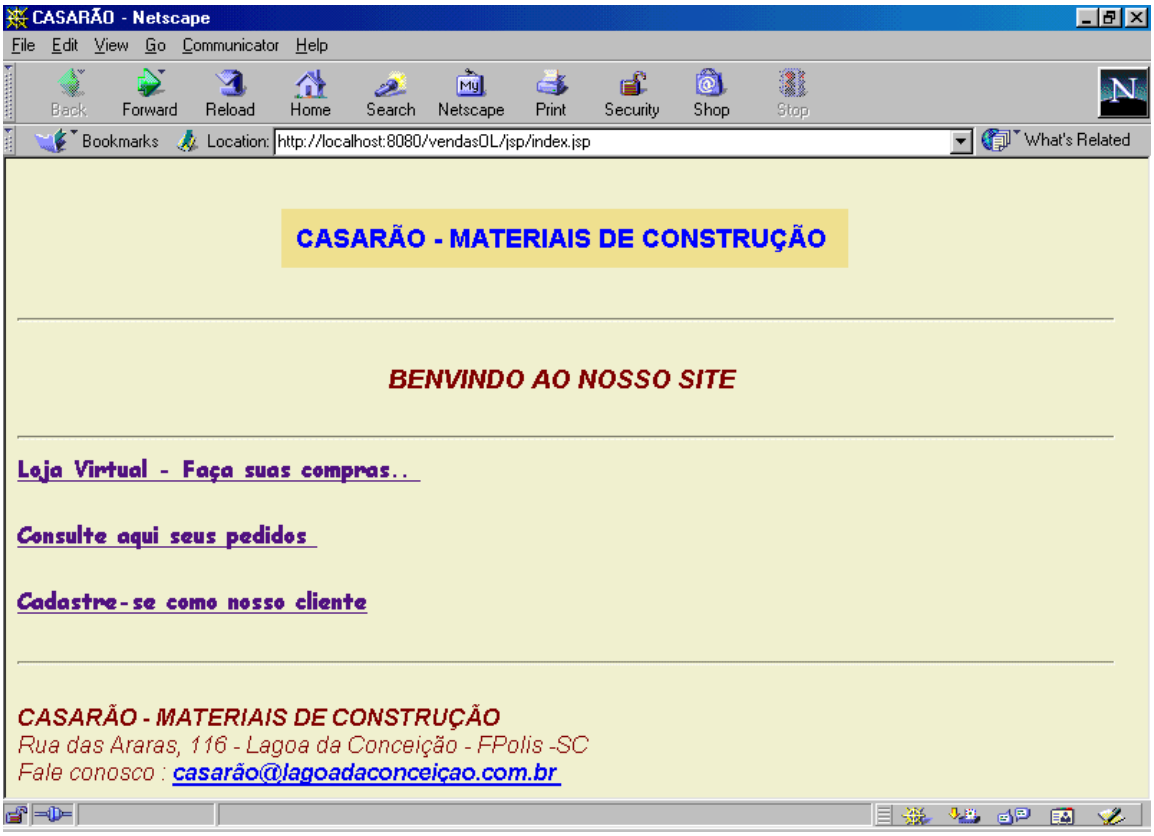

Fig. 4.3 - Página principal do sistema.

Caso de uso `EntrarLoja`

Descreve a seleção de uma categoria de produtos.

Cenário principal

1) O sistema apresenta uma página com diversas categorias de produtos.
2) O usuário seleciona uma categoria e o caso de uso `VerCatálogo` é acionado.

Cenários secundários

1) O usuário não faz nenhuma seleção e sai do sítio.
2) A qualquer momento o usuário pode optar por *Conferir Compras* : o sistema aciona o caso de uso `ConferirCompras`.
3) A qualquer momento o usuário pode optar por *Ir ao Caixa* : o sistema aciona o caso de uso `IrAoCaixa`.
4) A qualquer momento o usuário pode optar por *Voltar à página inicial* : o sistema aciona o caso de uso `AcessarPágina`.

Interfaces Gráficas

Fig. 4.4 - Lista de categorias.

Caso de uso `VerCatálogo`

Descreve a exibição do catálogo de uma categoria de produtos.

Cenário principal

1) Para cada item da categoria, o sistema apresenta uma linha contendo:
    - código
    - descrição do item e unidade
    - valor unitário
    - opção para selecionar o item para compra
2) O cliente seleciona um item.
3) O sistema aciona o caso de uso `SelecionarItem`.

Cenários secundários

1) O usuário não faz nenhuma seleção e sai do site.
2) A qualquer momento o usuário pode optar por *Conferir Compras* : o sistema aciona o caso de uso `ConferirCompras`.
3) A qualquer momento o usuário pode optar por *Índice do Catálogo* : o sistema aciona o caso de uso `EntrarLoja`.

Interfaces Gráficas

Fig. 4.5 - Lista de materiais disponíveis na categoria PISOS.

Caso de uso `SelecionarItem`

Descreve o processo de seleção de um item.

Cenário principal

Pré-condição: O usuário selecionou um item do catálogo.

1) O sistema apresenta o código, descrição, unidade e valor unitário do item.
2) O usuário digita a quantidade desejada.
3) O sistema calcula o valor total do item, insere o item na lista de compras e aciona o caso de uso `ConferirCompras`.

Cenários secundários

1) Caso o item já tenha sido selecionado pelo cliente, o sistema aciona o caso de uso `ConferirCompras`, onde o cliente poderá alterar a quantidade do item escolhido ou removê-lo da lista de compras.
2) Caso o usuário não confirme a aquisição, o sistema não insere o item na lista de compras.
3) A qualquer momento o usuário pode optar por *Conferir Compras* : o sistema aciona o caso de uso `ConferirCompras`.
4) A qualquer momento o usuário pode optar por *Índice do Catálogo* : o sistema aciona o caso de uso `EntrarLoja`.

Interfaces Gráficas

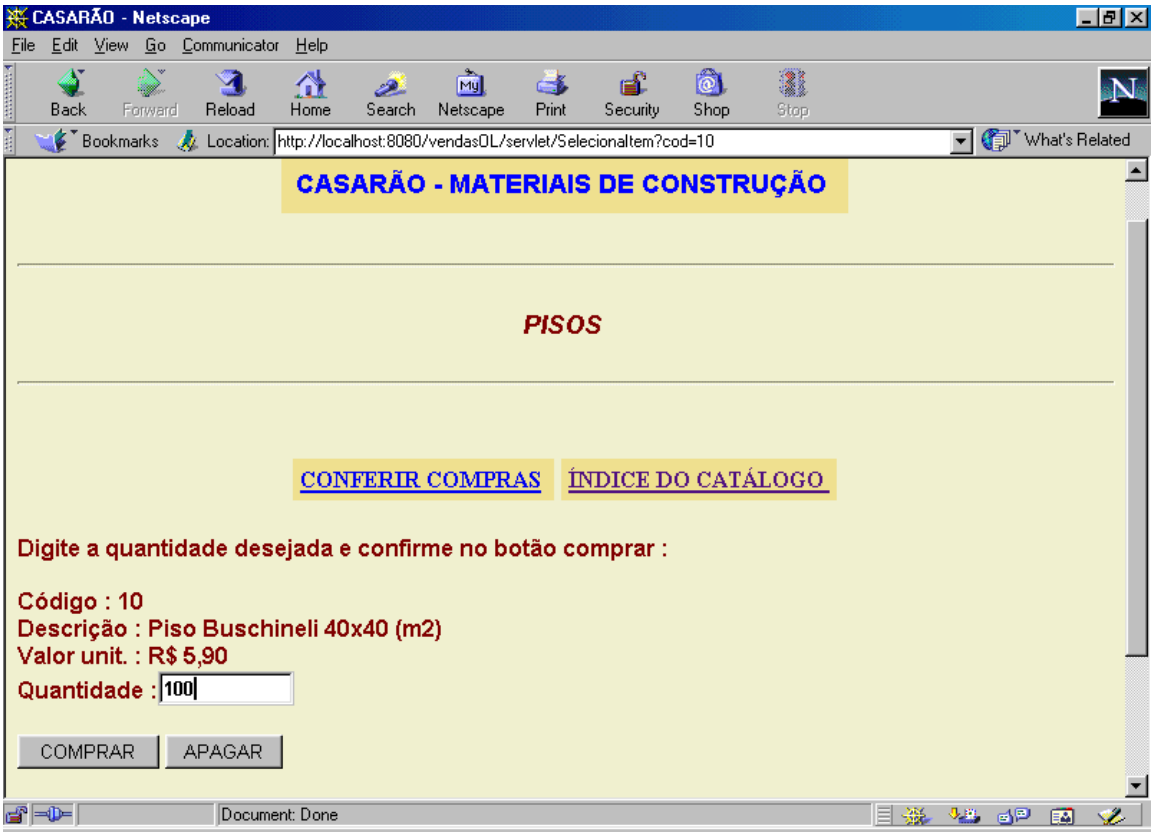

Fig. 4.6 - Seleção de um item para compra.

Caso de uso `ConferirCompras`

Descreve a conferência das compras até então e a remoção ou alteração de quantidades de produtos já selecionados.

Cenário principal

1) O sistema mostra uma página que contém as seguintes informações:
   a) Para cada item comprado são apresentadas as seguintes informações:
      - descrição do item comprado
      - valor unitário
      - quantidade comprada
      - valor total do item
      - opção de escolha para alterar a quantidade
      - opção de escolha para remover o item da lista
   b) Valor total das compras

2) Para cada item apresentado são oferecidas duas opções:
   a) Se o usuário optar por remover o item da lista o sistema aciona o caso de uso `DevolverItem`.
   b) Se o usuário optar por alterar a quantidade pedida do item o sistema aciona o caso de uso `AlterarQtde`.

3) O cliente opta por *Voltar* e é apresentada a página em que o cliente estava quando este caso de uso foi acionado.

Cenários secundários

1) A lista de compras está vazia, então não aparece nenhuma linha de dados e o valor total é zero.
2) A qualquer momento o cliente pode optar por *Voltar* e é apresentada a página em que o cliente estava quando este caso de uso foi acionado.
3) A qualquer momento o usuário pode optar por *Apagar todas as compras*: o sistema aciona o caso de uso `DevolverCompras`.

Interfaces Gráficas

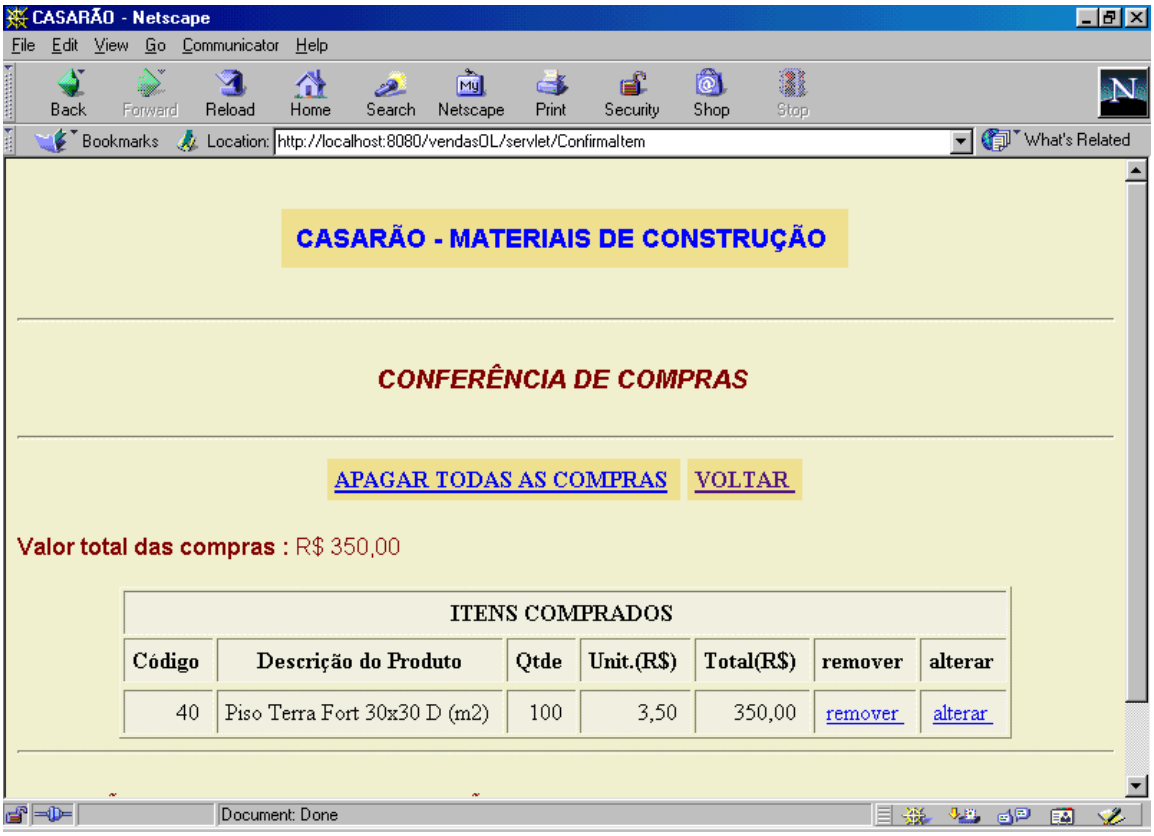

Fig. 4.7 - Consulta da lista de compras.

Caso de uso `DevolverItem`

Descreve o processo de remoção de um item da lista de compras.

Cenário Principal

Pré-condição: O usuário selecionou um item da lista para remoção.

1) O sistema subtrai o valor do item do valor total da lista
2) O sistema remove o item da lista.
3) O sistema aciona o caso de uso `ConferirCompras`.

Caso de uso `AlterarQtde`

Descreve o processo de alteração de quantidade pedida de um item da lista de compras.

Cenário Principal

Pré-condição: O usuário selecionou um item do lista para alteração de quantidade.

1) O sistema apresenta o código, descrição, unidade e valor unitário do item, quantidade pedida e valor total.
2) O usuário digita a nova quantidade.
3) O sistema calcula o valor total do item, insere o item na lista de compras e soma o valor total do item ao valor total da lista.
4) O sistema aciona o caso de uso `ConferirCompras`.

Cenários secundários

1) O usuário pode desistir de alterar a quantidade e a quantidade anterior não é alterada.
2) A qualquer momento o usuário pode optar por *Conferir Compras* : o sistema aciona o caso de uso `ConferirCompras`.

Interfaces Gráficas

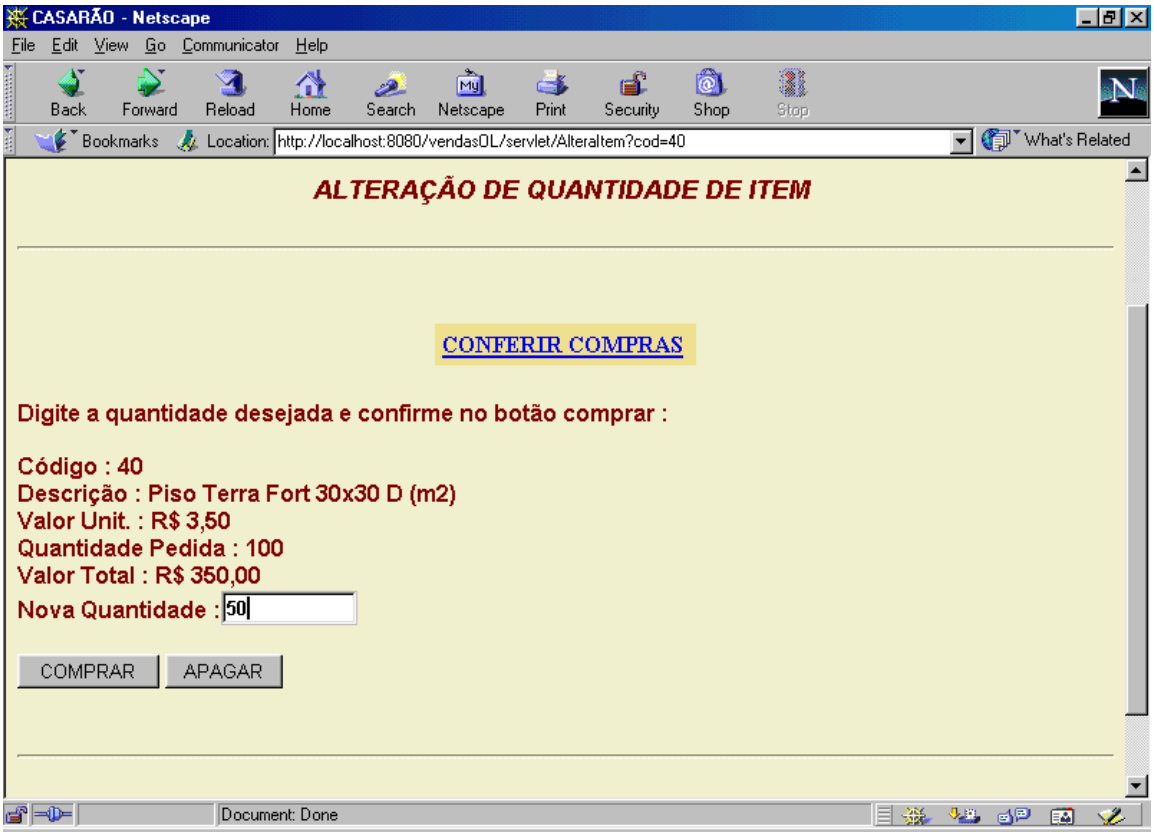

Fig. 4.8 - Alteração de quantidade de item da lista de compras.

Caso de uso `DevolverCompras`

Descreve o processo de remoção de todos os itens da lista de compras.

Cenário Principal

1) O sistema coloca o valor 0 (zero) no valor total da lista.
2) O sistema remove todos os itens da lista.
3) O sistema aciona o caso de uso `ConferirCompras`.

Cenários secundários

1) Se a lista estiver vazia o sistema não faz nada.

Caso de uso `IrAoCaixa`

Descreve o processo de compra dos itens escolhidos pelo cliente.

Cenário principal

1) O sistema pede ao cliente que digite sua identificação no sistema.
2) Se o cliente ainda não possui uma identificação, o sistema oferece um atalho para o caso de uso `InserirCliente`, que cadastra uma identificação para o cliente.
3) O cliente digita sua identificação.
4) O sistema recupera o cadastro do cliente referente à identificação fornecida, mostra uma página contendo o nome e e-mail do cliente e pede que o cliente entre com tipo e número do cartão de crédito e o endereço para entrega do material.
5) O cliente entra com os dados e submete o formulário.
6) O sistema monta um pedido com os dados contidos no cadastro de clientes, na lista de compras e os dados fornecidos pelo cliente. Apresenta então uma página com nome, endereço para entrega, e-mail, tipo e número do cartão de crédito, valor total da compra e itens do pedido feito e pede ao cliente que confirme a compra.
7) O cliente confirma a compra.
8) O sistema coloca o pedido em estado “Em análise”.
9) O sistema aciona o caso de uso `AnalisarCrédito` fornecendo o pedido eo cadastro do cliente.
10) O sistema recebe uma mensagem de aprovação do caso de uso `AnalisarCrédito`.
11) O sistema apresenta uma página informando ao cliente o sucesso da transação e o número de identificação criado para o pedido.

Cenários secundários

1) Se a lista de compras estiver vazia o sistema envia uma mensagem ao cliente.
2) Caso o cliente tenha esquecido sua identificação, o sistema oferece um atalho para o caso de uso `RecuperarID` que solicita o nome completo e CIC/CGC do cliente e fornece a identificação correspondente encontrada no cadastro de clientes.
3) Caso o cliente não confirme a compra, o sistema ignora o pedido.

4) Caso o pagamento tenha sido reprovado, o sistema apresenta ao cliente uma mensagem de reprovação de crédito com a justificativa fornecida pelo Sistema da Cia. de Cartões de Crédito .

Interfaces Gráficas

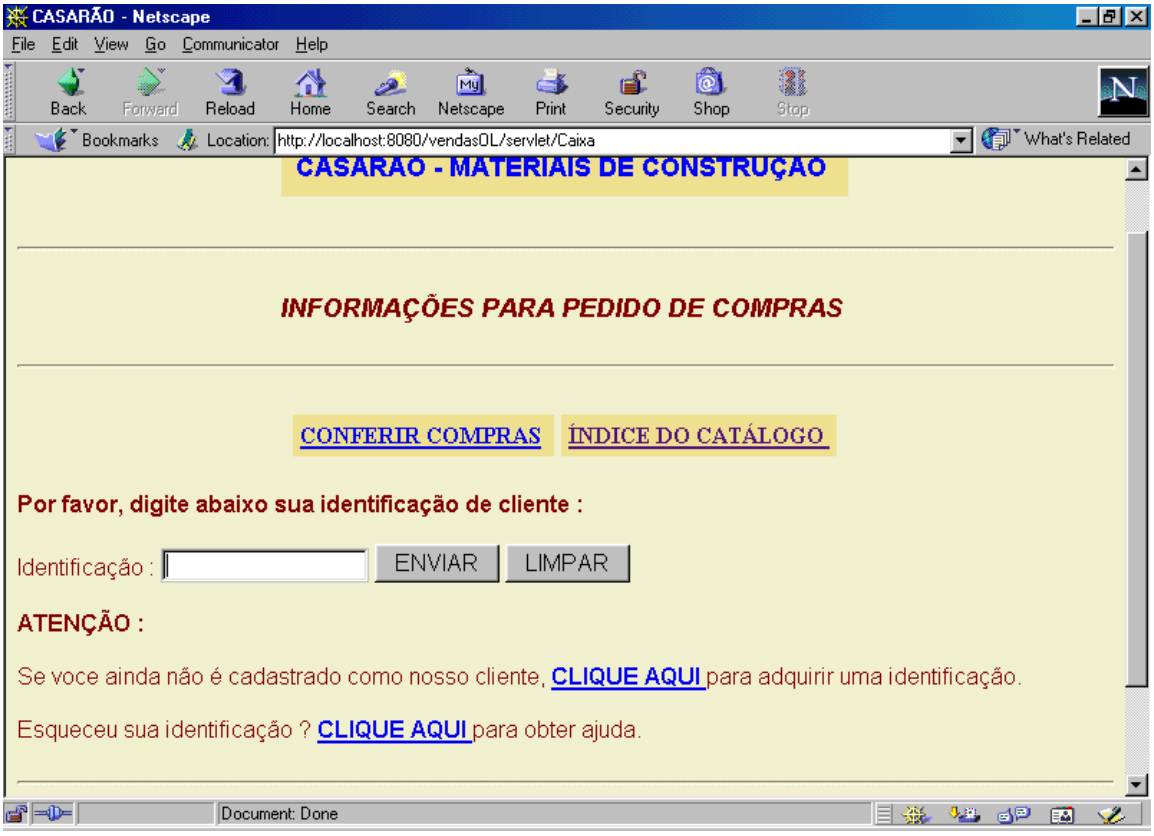

Fig. 4.9 - Solicitação de identificação do cliente para realização de uma compra.

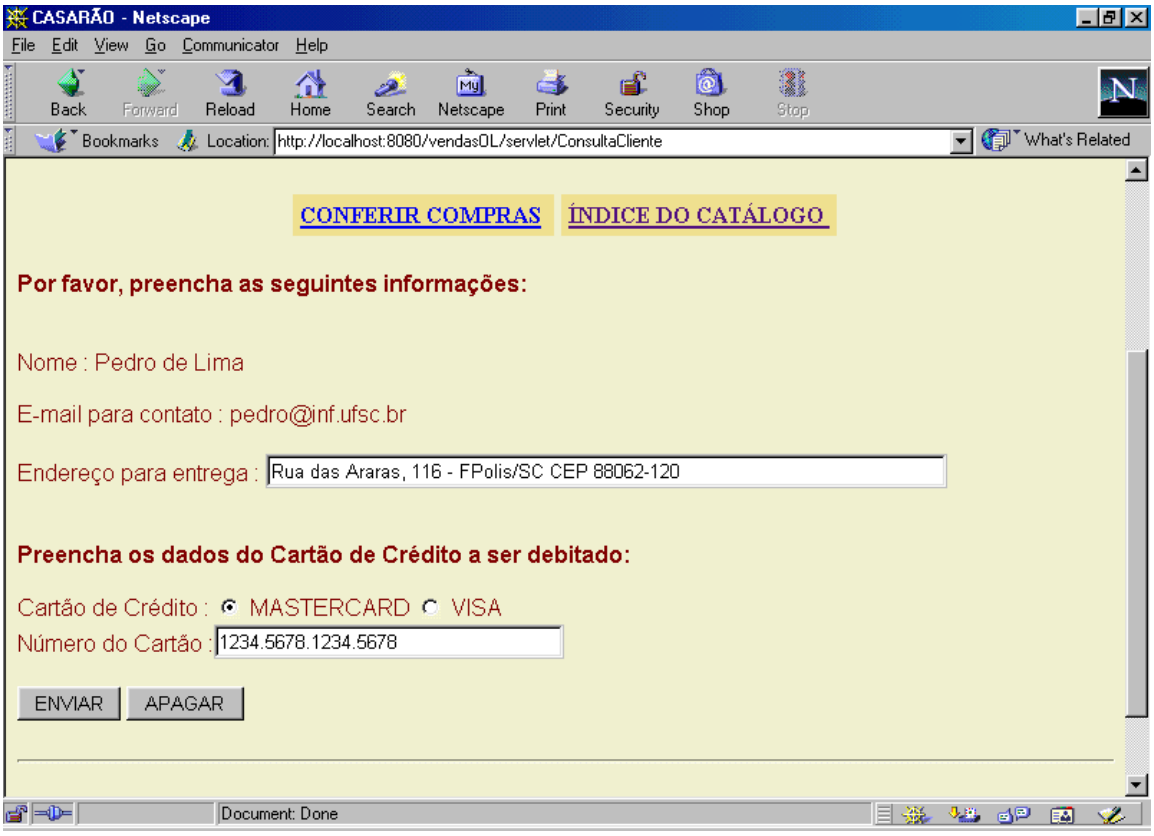

Fig. 4.10 - Formulário de informações para compras.

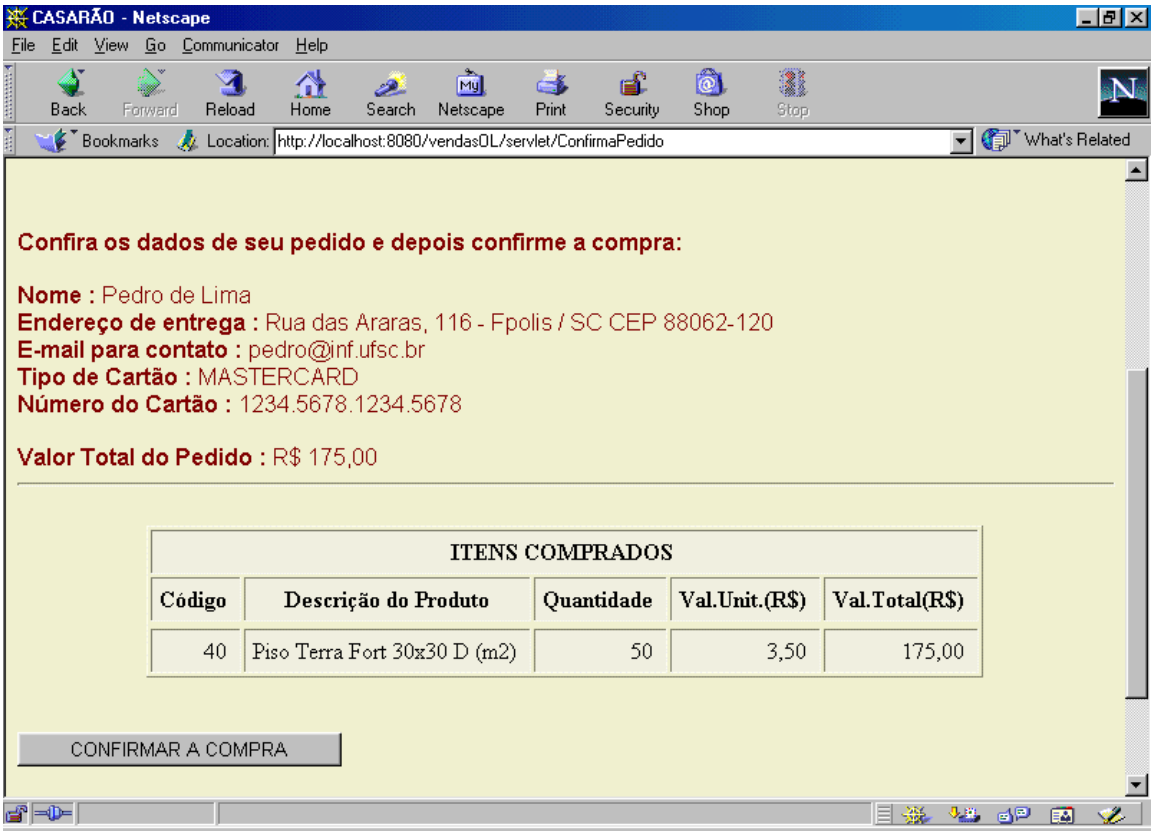

Fig. 4.11 - Formulário de solicitação de confirmação de compra.

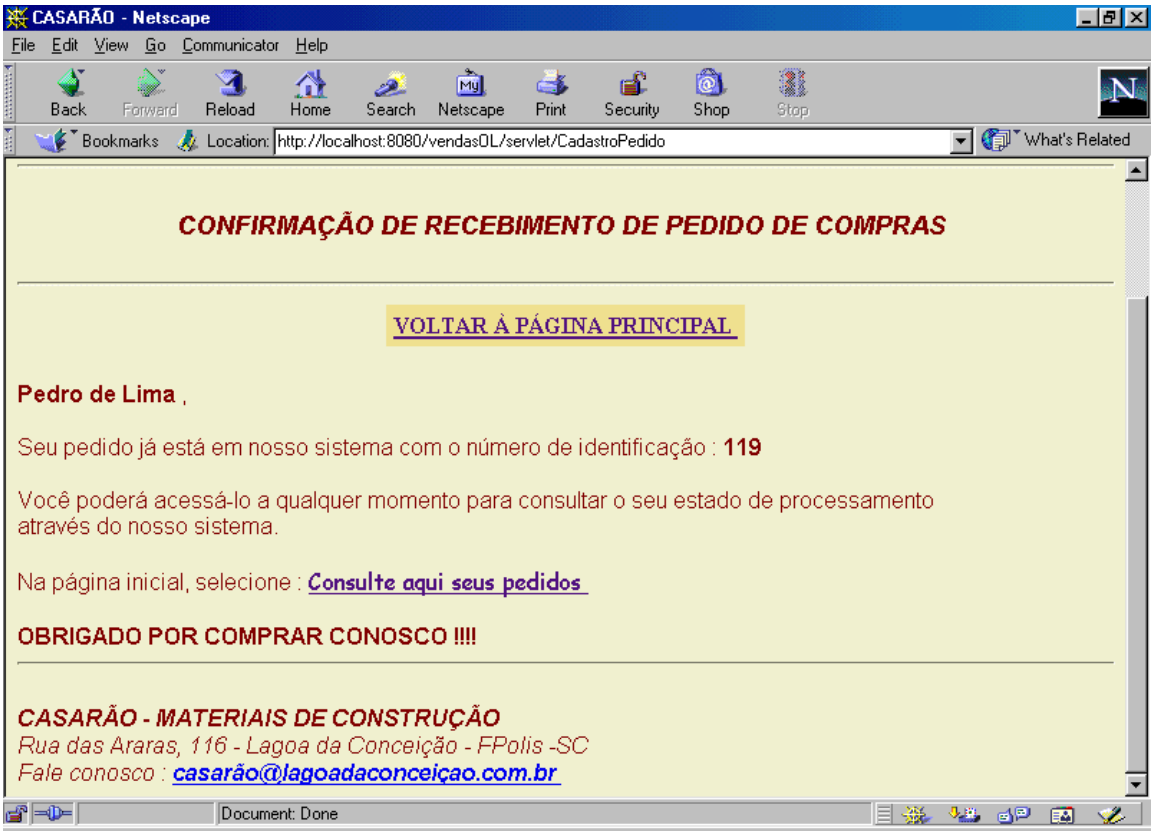

Fig. 4.12 - Mensagem de compra efetuada com sucesso.

Caso de uso `AnalisarCrédito`

Descreve como é efetuado o pagamento de um Pedido.

Cenário principal

Pré-condição: o caso de uso recebe o pedido a ser analisado e o cadastro do cliente.

1) O sistema envia o tipo e número do cartão e o valor a ser debitado ao Sistema da Cia. de Cartão de Crédito e solicita a aprovação.
2) O sistema recebe uma resposta de aprovação ou reprovação acompanhado de uma mensagem de justificação.
3) Se o pagamento foi aprovado as seguintes ações são tomadas:
    - o estado do pedido muda para "Aprovado", ficando disponível para ser processado pelo Estoque.
    - o sistema aciona o caso de uso `LançarContabil fornecendo` o número do pedido e o valor debitado do cartão.
4) Caso contrário, o estado do pedido muda para "Recusado".
5) O caso de uso retorna a resposta e a mensagem de justificação recebida.

Cenários secundários

1) O Sistema da Cia. de Cartão de Crédito não responde à solicitação.

Caso de uso `LançarCrédito`

Descreve como é feito o lançamento contábil para o sistema financeiro.

Cenário principal

Pré-condição: o caso de uso recebe um valor e o número do pedido.

1) O sistema faz um lançamento contábil de crédito contendo as seguintes informações:
    - ID do pedido
    - valor pago
    - data do pagamento

Caso de uso `CadastrarCliente`

Descreve o processo de cadastramento/atualização de um cliente no sistema.

Cenário principal:

1) O sistema apresenta uma página que contém as opções:
   - *Novo cliente*
   - *Atualização de cadastro*
2) Se o cliente optar por *Novo cliente* o sistema aciona o caso de uso `InserirCliente`.
3) Se o cliente optar por *Atualização de cadastro* o sistema aciona o caso de uso `AtualizarCliente`.

Cenários secundários

1) O usuário opta por *Voltar à página inicial* e o sistema aciona o caso de uso `AcessarPágina`.

Interfaces Gráficas

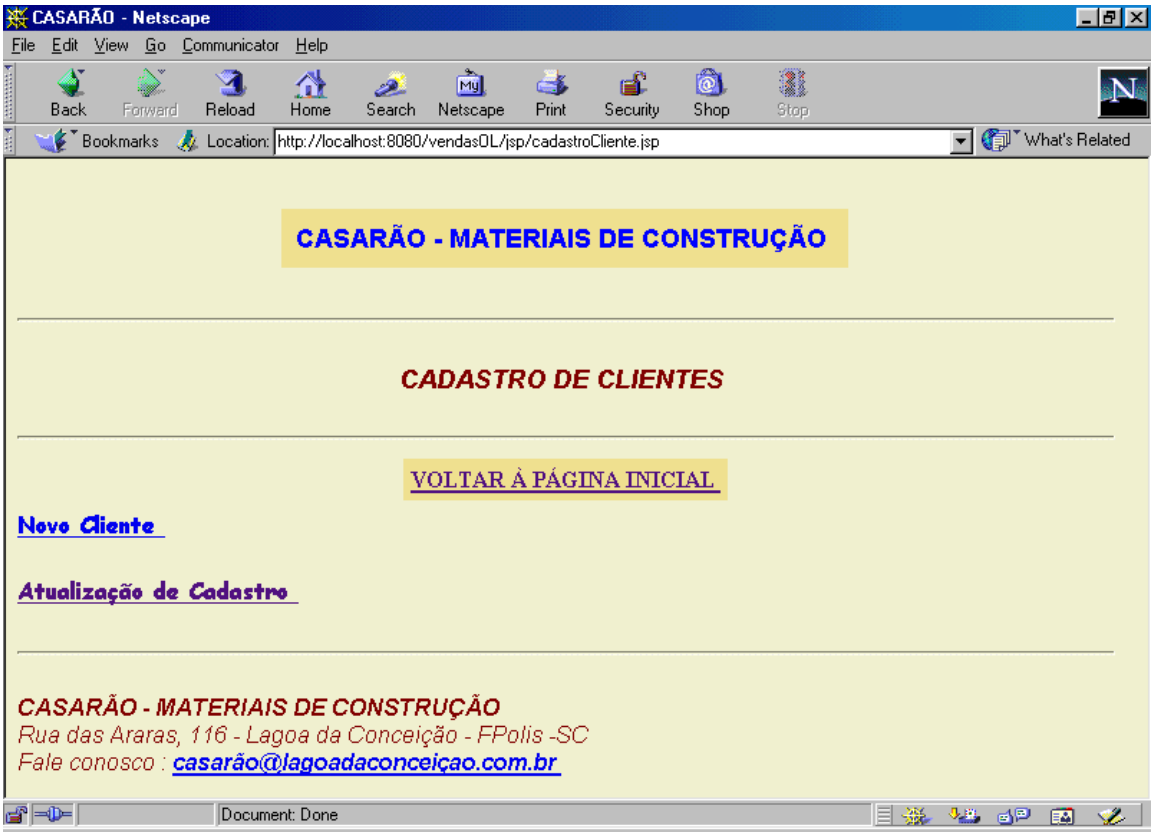

Fig. 4.13 - Página de cadastramento de clientes.

Caso de uso `InserirCliente`

Descreve o processo de cadastramento de um novo cliente no sistema.

Cenário principal

1) O sistema apresenta um formulário que solicita as seguintes informações:
   - Nome completo
   - Endereço para correspondência
   - e-mail
   - telefone
   - tipo de cliente (pessoa física ou jurídica)
   - CIC/CGC
   - Identificação a ser escolhida pelo cliente
2) o usuário preenche o formulário e submete as informações
3) o sistema cria um número único para o cliente
4) o sistema insere os dados no cadastro
5) o sistema devolve uma mensagem informando o sucesso da transação.
6) O cliente pode optar por *Voltar ao caixa* e o caso de uso `IrAoCaixa` é acionado, ou *Voltar à página de clientes* e o caso de uso `CadastrarCliente` é acionado.

Cenários secundários

1) A identificação escolhida já existe no banco de dados e o sistema retorna uma mensagem solicitando a troca da identificação.
2) Problemas na conexão com o banco de dados.

Interfaces Gráficas

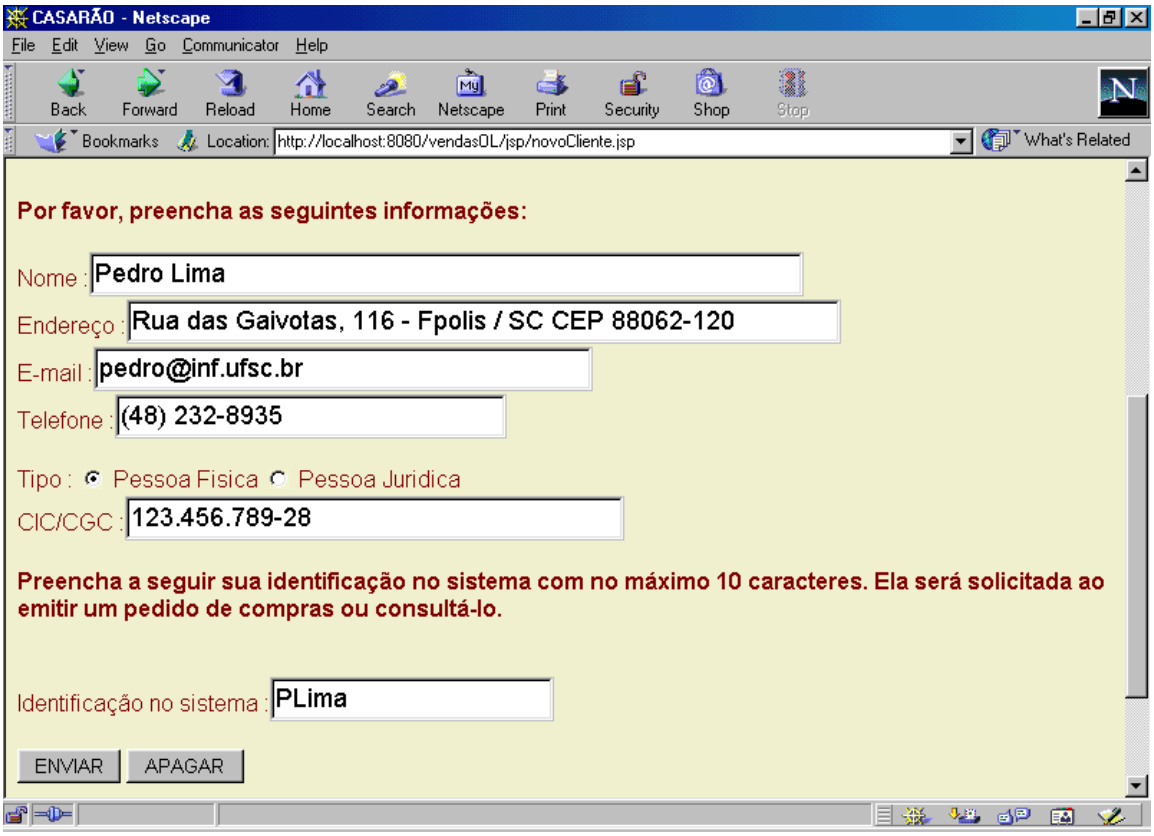

Fig. 4.14 - Página de cadastramento de novos clientes.

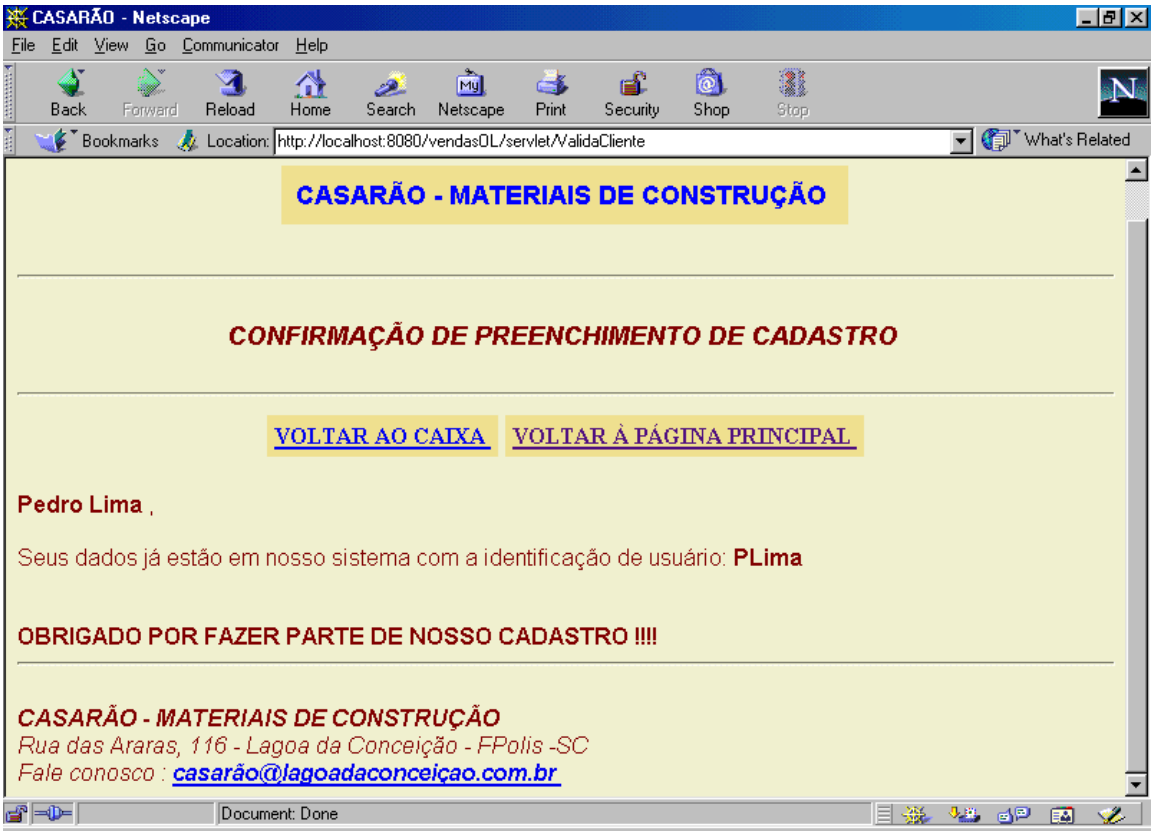

Fig. 4.15 - Página de mensagem de sucesso da transação.

Caso de uso `AtualizarCliente`

Descreve o processo de atualização do cadastro de um cliente no sistema.

Cenário principal

1) O sistema solicita que os usuário digite sua identificação.
2) O usuário informa sua identificação.
3) O sistema apresenta um formulário com os dados do cadastro.
4) O usuário modifica as informações necessárias e submete o formulário.
5) O sistema grava as novas informações no banco de dados e envia uma mensagem de sucesso de transação.
6) O usuário seleciona *Voltar* e o caso de uso `CadastrarCliente` é acionado.

Cenários secundários

1) Caso o cliente tenha esquecido sua identificação, o sistema oferece um atalho para o caso de uso `RecuperarID` que solicita o nome completo e CIC/CGC do cliente e fornece a identificação correspondente encontrada no cadastro.
2) Problemas na conexão com o banco de dados.
3) O usuário seleciona *Voltar* e o caso de uso `CadastrarCliente` é acionado.

Interfaces Gráficas

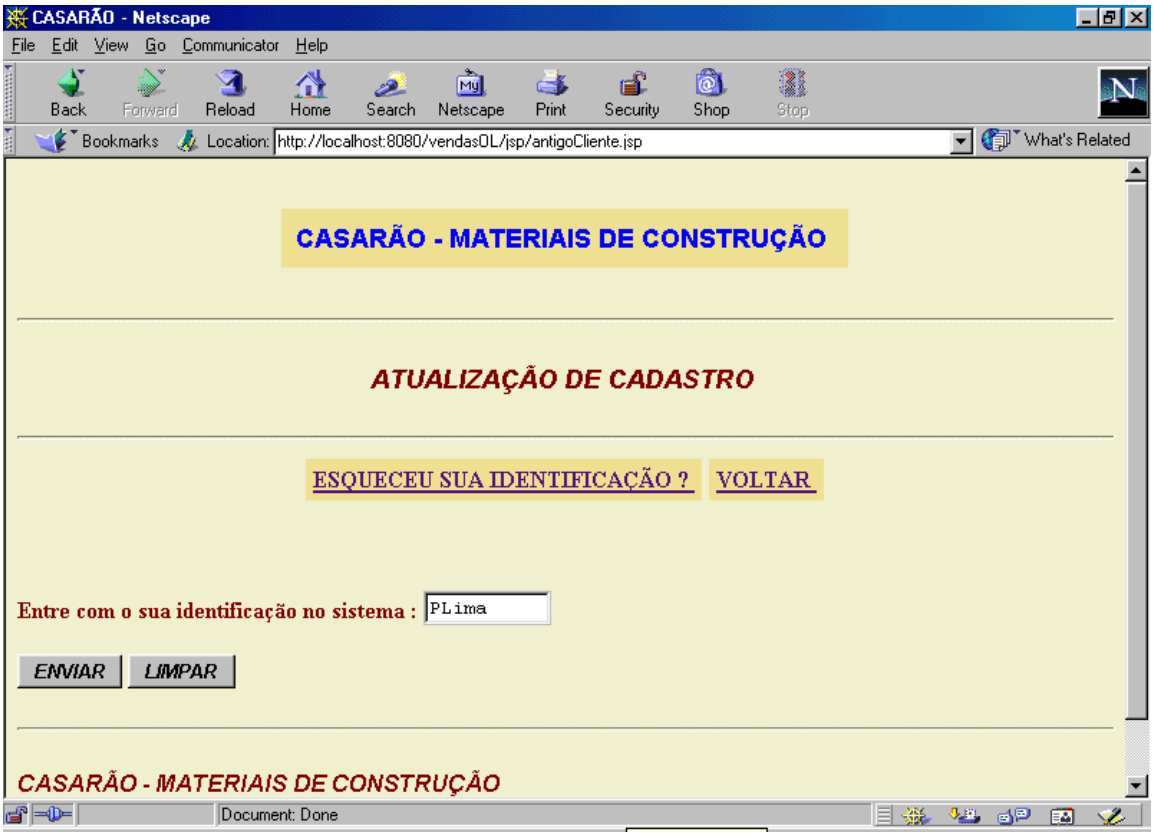

Fig. 4.16 - Página de solicitação de identificação para atualização de cadastro.

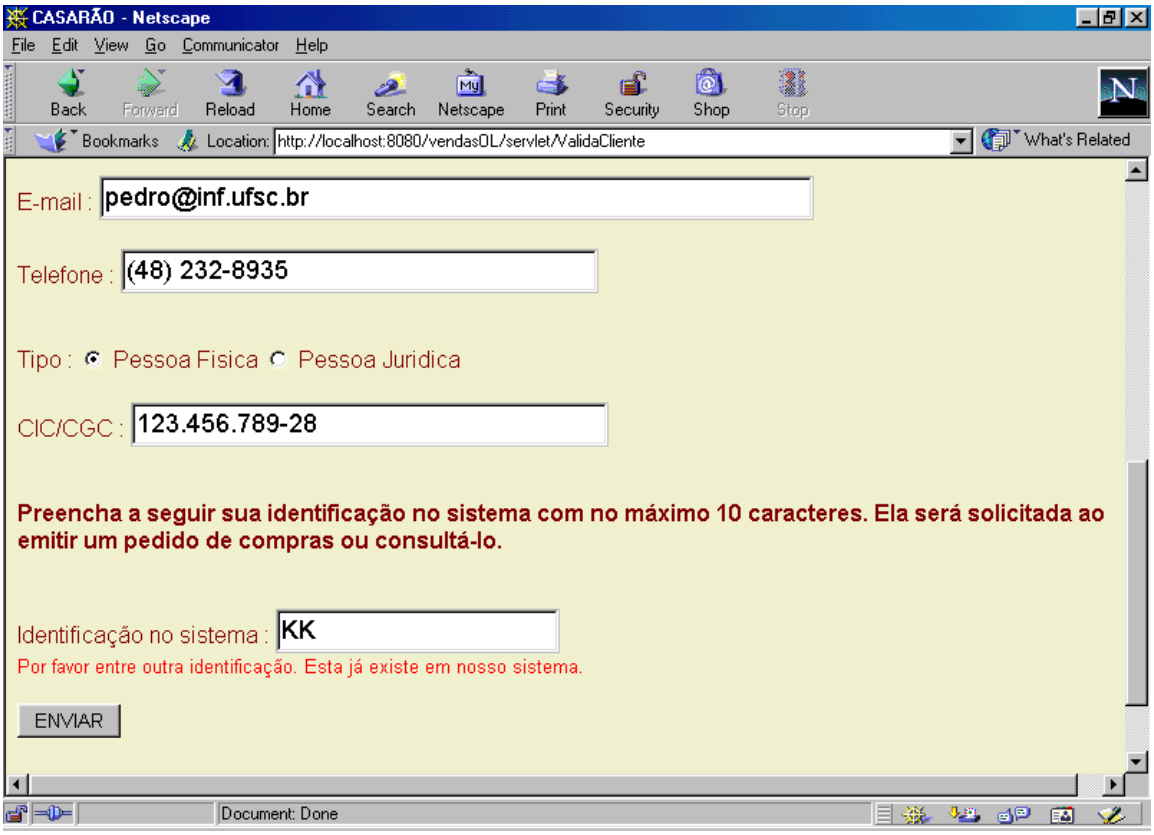

Fig. 4.17 - Página de atualização de cadastro de clientes

Caso de uso `RecuperarID`

Descreve como uma identificação é recuperada caso o cliente tenha esquecido a identificação.

Cenário principal

1) O sistema pede ao cliente que entre com o nome completo e CIC/CGC.
2) O cliente informa os dados pedidos e submete o formulário.
3) Se o sistema acha as informações no cadastro de clientes, ele informa ao cliente a identificação encontrada.
4) Senão o sistema solicita ao cliente que entre em contato com o setor de atendimento ao cliente via e-mail ou telefone.
5) O cliente opta por *Voltar* e é apresentada a página que o cliente estava quando este caso de uso foi acionado.

Cenários secundários

1) A qualquer momento o cliente pode optar por *Voltar* e é apresentada a página em que o cliente estava quando este caso de uso foi acionado.

Caso de uso `EnviarMsg`

Descreve o processo de troca de mensagens entre cliente e setor de Atendimento ao Cliente.

Cenário principal

1) O sistema apresenta uma tela de envio de mensagens.
2) O sistema preenche o campo de destinatário com o endereço e-mail do Atendimento ao Cliente.
3) O usuário escreve a mensagem.
4) O usuário opta por *Enviar* a mensagem.
5) O sistema envia a mensagem.
6) O cliente opta por *Voltar* e é apresentada a página que o cliente estava quando este caso de uso foi acionado.

Cenários secundários

1) A qualquer momento o cliente pode optar por *Voltar* e é apresentada a página em que o cliente estava quando este caso de uso foi acionado.

Caso de uso `ConsultarPedido`

Descreve o processo de consulta de pedidos pelo cliente.

Cenário principal

1) O sistema pede ao usuário que entre com sua identificação.
2) O usuário fornece sua identificação.
3) O sistema acessa o cadasto do cliente para validar sua identificação.
4) O sistema recupera todos os pedidos do cliente no banco de dados.
5) O sistema mostra uma página com um resumo de cada pedido da lista recuperada, contendo a data do pedido, o valor total, o estado do pedido e um atalho para ver os detalhes do pedido.
6) O usuário escolhe ver detalhes de um pedido.
7) O sistema mostra uma página com nome, endereço para entrega, e-mail, valor total da compra, itens do pedido e estado do pedido escolhido.

Cenários secundários

1) Caso o usuário tenha esquecido a identificação a página contém o atalho onde o sistema aciona o caso de uso `RecuperarID`.
2) Se não achou nenhum pedido para esta identificação, o sistema emite uma mensagem.
3) O usuário resolve não ver nenhum detalhe do pedido e optar por *Voltar à página inicial*, e o caso de uso `AcessarPágina` é acionado.

Interfaces Gráficas

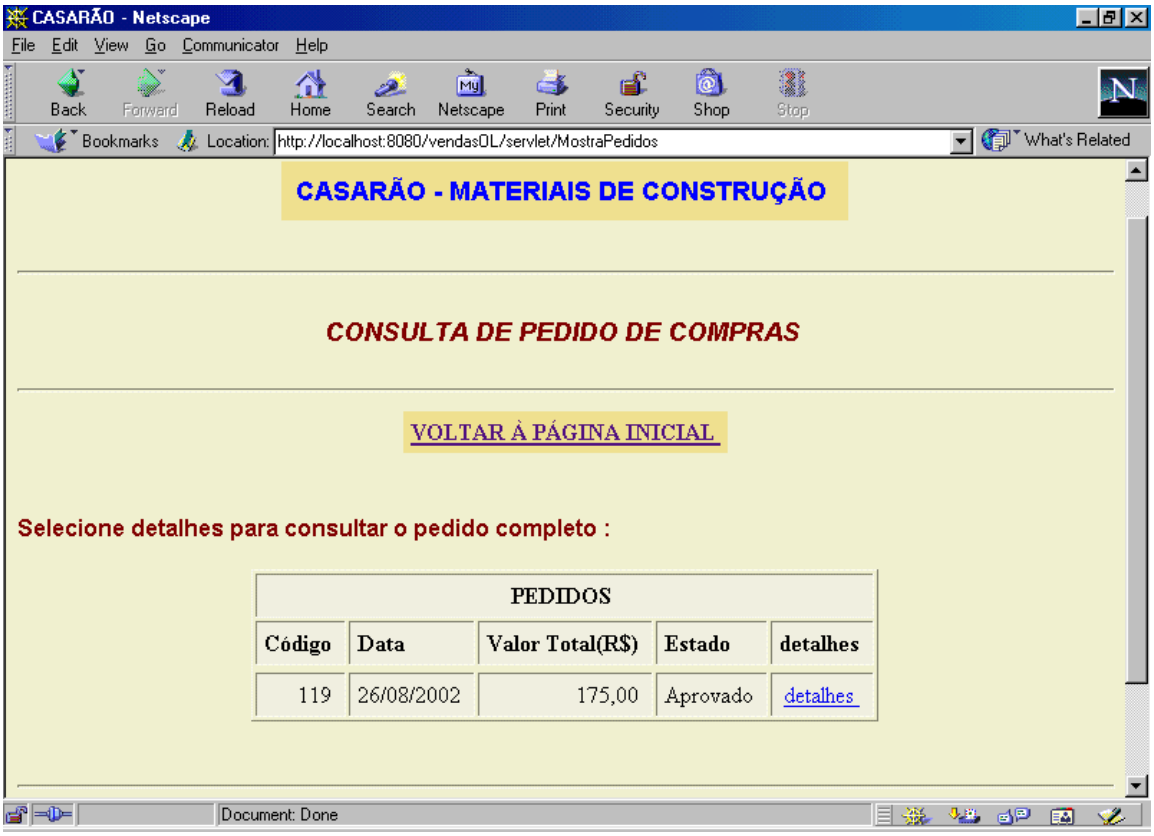

Fig. 4.18 - Página de consulta de pedidos do cliente.

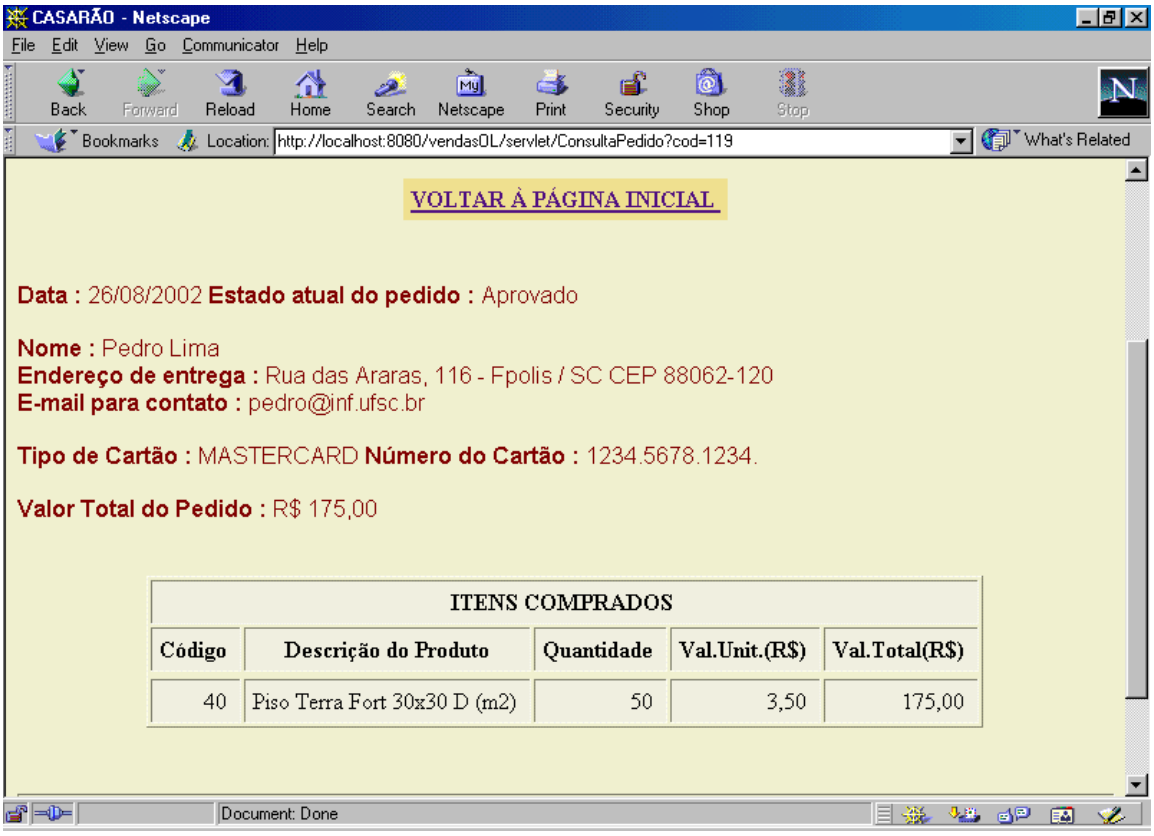

Fig. 4.19 - Página de consulta de detalhe de um pedido.

Caso de uso `Login`

Descreve como um usuário de Estoque, Expedição ou Atendimento ao Cliente faz seu acesso ao sistema.

Cenário principal

Pré-condição: os usuários devem estar pré cadastrados no sistema.

1) O usuário acessa a URL da página do sistema Casarão de acesso ao sistema interno da empresa .
2) O sistema pede para que o usuário informe seu nome de usuário e senha.
3) O usuário entra com os dados e solicita o acesso ao sistema.
4) O sistema recupera as informações sobre usuário no banco de dados e analisa-os.
5) Se o nome do usuário ou senha estiverem incorretos o sistema emite uma mensagem de erro.
6) Caso contrário, o sistema executa as seguintes ações:
   a) Se é um usuário de Atendimento ao Cliente o sistema aciona o caso de uso `AtenderCliente`.
   b) Se é um usuário de Estoque o sistema aciona o caso de uso `SepararMaterial`.
   c) Se é um usuário de Expedição o sistema aciona o caso de uso `DespacharMaterial`.

Cenários secundários

1) Problemas na conexão com o banco de dados.

Interfaces Gráficas

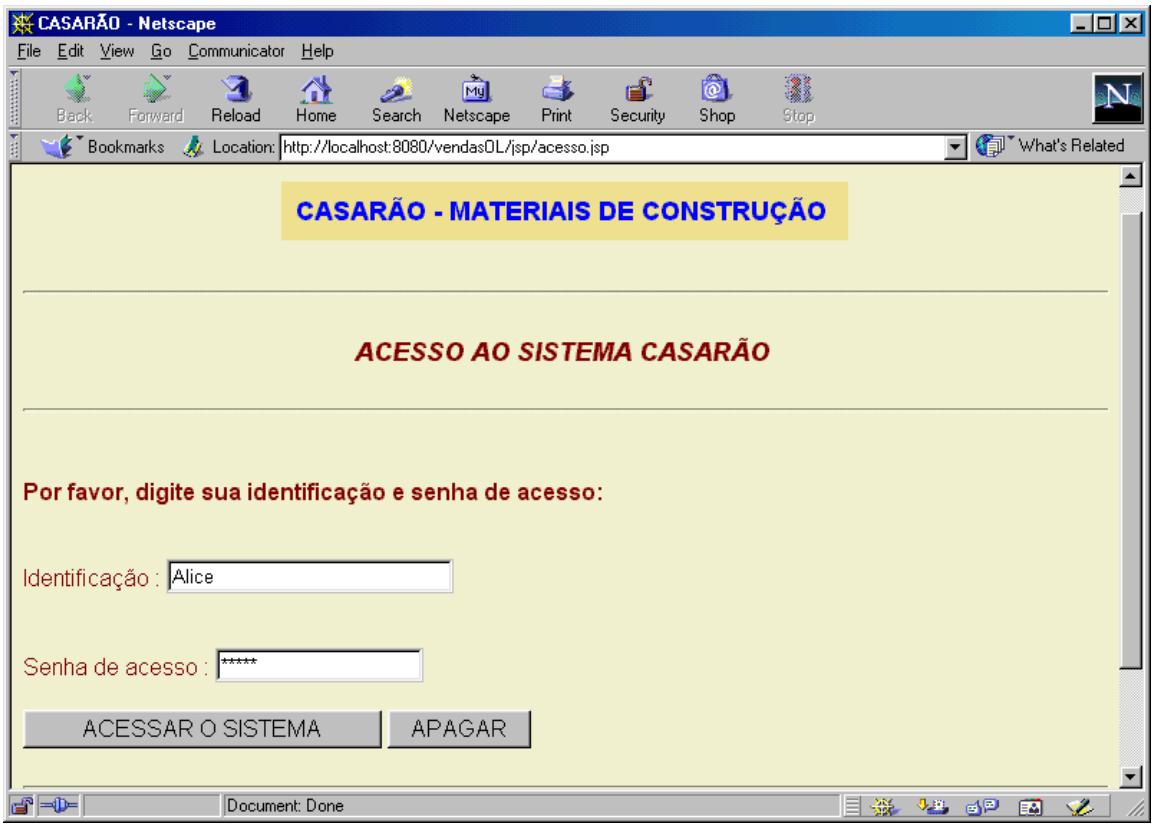

Fig. 4.20 - Página de acesso ao sistema Casarão.

Caso de uso `AtenderCliente`

Descreve as opções de atendimento ao cliente.

Cenário principal

Pré-condição: o usuário é do tipo Atendimento ao Cliente.

1) Se o usuário escolher a opção *Simular Cliente*, o sistema aciona o caso de uso `AcessarPágina`.
2) Se o usuário escolher a opção *Mensagens com Cliente*, o sistema aciona o programa de e-mail que permite ler/enviar mensagens de/para clientes.
3) Se o usuário escolher a opção *Cadastrar Reclamação*, o sistema permite inserir alguma observação associada ao um pedido do cliente.

Caso de uso `SepararMaterial`

Descreve o processo de separação de material pelo estoque.

Cenário principal

1) O sistema aciona o caso de uso `ResgatarPedidos`.
2) O sistema mostra uma lista de pedidos, onde cada item da lista contém número do pedido, data do pedido, estado do pedido e uma opção de seleção para processar o pedido.
3) Para cada pedido, se o usuário seleciona o pedido o caso de uso `ProcessarPedido` é acionado.
4) O usuário seleciona *Sair do sistema* e o caso de uso `Login` é acionado.

Cenários secundários

1) Não há nenhum pedido para ser processado.

Interfaces Gráficas

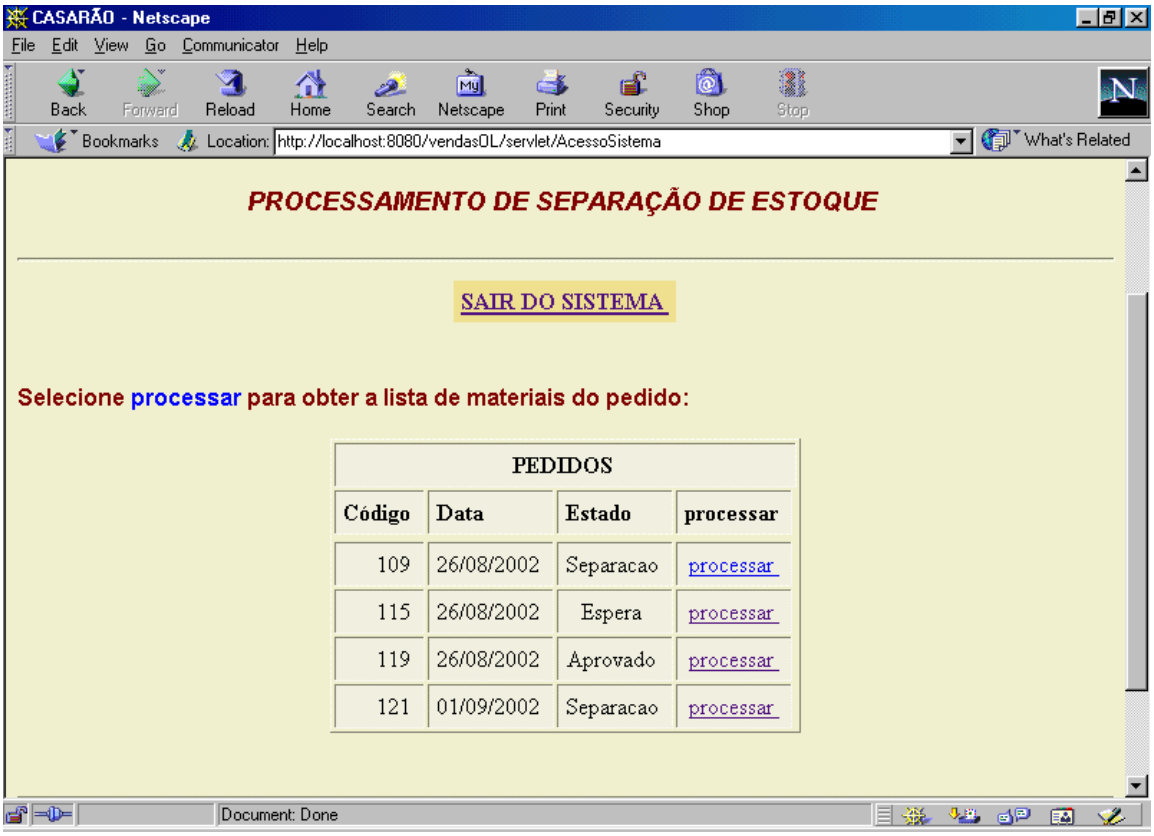

Fig. 4.21 - Lista de pedidos a serem processados pelo Estoque.

Caso de uso `ProcessarPedido`

Descreve o processo de separação de material dos itens do pedido pelo estoque.

Cenário principal

1) O sistema mostra uma lista com os itens do pedido onde cada linha contém o código, descrição, quantidade pedida e estado da item. Para os itens que ainda não foram separados (estado de "Aprovado" ou Espera") é fornecida uma opção de processamento para o item.
2) Para cada item selecionado pelo usuário as seguintes ações são possíveis:
   a) Se há quantidade disponível, as seguintes ações são executadas:
      1. O item passa a estado de "Atendido".
      2. O sistema atualiza a quantidade de estoque.
      3. Se a quantidade resultante no estoque for menor do que a quantidade mínima permitida, o sistema aciona o caso de uso `EmitirOC`.
   b) Caso contrário, o sistema aciona o caso de uso `EmitirOC` e muda o estado do item para "Espera".
3) Se todos os itens foram atendidos o estado do pedido muda para "Atendido".
4) Se algum item ficou em estado de "Espera", o estado do pedido muda para "Espera".
5) Se o processo de separação foi interrompido e algum item continua em estado de "Aprovado", o estado do pedido continua sendo "Separação".
6) O sistema volta ao passo 1).

Cenários secundários

1) O usuário escolhe *Voltar à lista de separação* o sistema aciona o caso de uso `SepararMaterial`.

Interfaces Gráficas

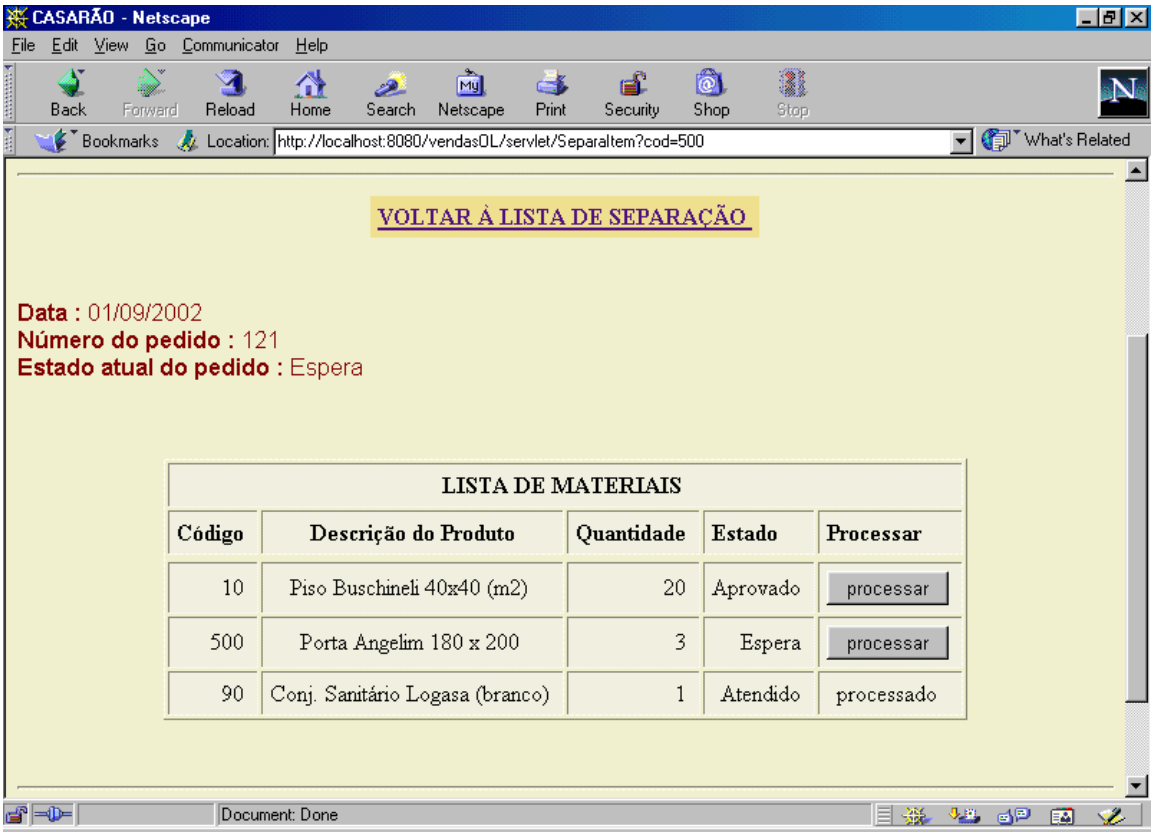

Fig. 4.22 - Lista dos itens a serem processados para um pedido.

Caso de uso `DespacharMaterial`

Descreve o processo de despachar o material do pedido pela Expedição.

Cenário principal

1) O sistema aciona o caso de uso `ResgatarPedidos`.
2) O sistema mostra uma lista de pedidos onde cada linha contém data, ID do pedido, estado do pedido, endereço de entrega e uma opção para emissão de nota fiscal.
3) Para cada pedido, apenas uma entre as seguintes ações é possível:
   a) Se o usuário selecionar *Emitir Nota Fiscal*, o sistema aciona o caso de uso `EmitirNF` para o pedido . Se a nota fiscal foi emitida (impressa) com sucesso, o estado do pedido muda para "Expedição".
   b) Se o usuário selecionar *Processar* então o sistema muda o estado do pedido para "Despachado". Neste caso a nota fiscal já deve ter sido emitida.
4) O sistema volta ao item 1).

Pós-condições :

1) Todos os pedidos no estado "Despachado" tem nota fiscal emitida.

Cenários secundários

1) Ocorreu algum problema no processo de emissão da nota fiscal. Neste caso o estado do pedido não muda para "Expedição".

Interfaces gráficas

Fig. 4.23 - Lista dos pedidos a serem processados pela Expedição.

Caso de uso `EmitirNF`

Descreve o processo de emissão de nota fiscal.

Cenário principal

1) O sistema apresenta uma página com os valores da nota fiscal utilizando os dados do pedido e do sistema (data e impostos) e solicita que o usuário informe o número da nota fiscal.
2) O usuário entra com os dados e seleciona *Emitir* a nota fiscal.
3) O sistema imprime a nota fiscal e grava o número da nota fiscal no pedido.
4) O sistema envia os dados da nota fiscal para o Financeiro.
5) O usuário escolhe *Voltar à lista* e o sistema aciona o caso de uso `DespacharMaterial`.

Cenários secundários

1) Se ocorrer um erro de impressão, é enviada uma mensagem de erro e o caso de uso termina.
2) Se a nota fiscal já foi emitida, o sistema apresenta uma mensagem de erro.

Interfaces Gráficas

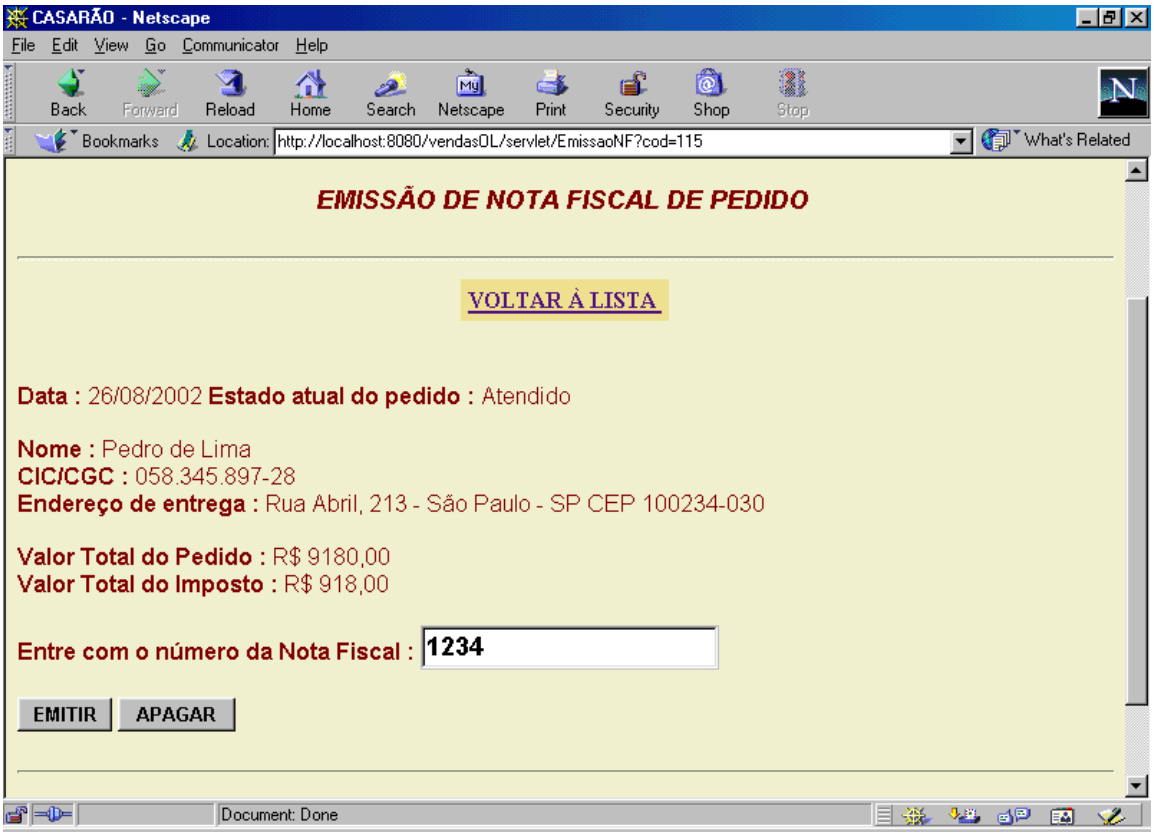

Fig. 4.24 - Página de emissão de nota fiscal.

Caso de uso `ResgatarPedidos`

Descreve o processo de leitura um conjunto de pedidos do banco de dados. Os pedidos a serem recuperados dependem do tipo de usuário que aciona o caso de uso.

Cenário principal

Pré-condição: o usuário é um usuário válido do tipo Estoque ou Expedição.

1) Se o usuário é do tipo Estoque o sistema solicita recupera do banco de dados os pedidos que tenham os estados "Aprovado", "Espera" ou "Separação" .
2) Senão se o usuário é do tipo Expedição o sistema recupera do banco de dados os pedidos que tenham o estado "Atendido" ou "Expedição".
3) O sistema devolve um conjunto com todos os pedidos encontrados.

Cenários secundários

1) Problemas na conexão com o banco de dados.
2) Não existem pedidos nestes estados e o sistema devolve um conjunto vazio.

Caso de uso `EmitirOC`

Descreve o processo de emissão de uma ordem de compra.

Cenário principal:

1) O sistema verifica se existe uma ordem de compra em aberto para o item.
2) Se não existe um ordem de compra em aberto, o sistema executa as seguintes ações:
    a) O sistema apresenta ao usuário uma ordem de compra para o item com a quantidade *default* de reposição de estoque.
    b) O usuário pode atualizar a quantidade.
    c) O usuário envia os dados e a confirma a emissão da ordem de compra.
    d) O sistema gera uma ordem de compra e envia ao Financeiro.
3) Senão o sistema apresenta a ordem de compra em aberto.
4) O usuário escolhe *Sair* e o caso de uso `ProcessarItem` é acionado.

Cenários secundários

1) O usuário desiste de emitir a ordem de compra.

4.5 Análise do comportamento do sistema Casarão

Para analisar o sistema do ponto de vista estático apresentamos primeiramente o diagrama de classes sob uma perspectiva conceitual. O aspecto dinâmico do sistema é analisado através de diagramas de interação. O comportamento dinâmico da classe `Pedido` é ilustrado através do seu diagrama de estados e uma análise da modularidade do sistema é apresentada no final.

4.5.1 Diagrama de classes

O diagrama a seguir descreve os tipos de objetos que existem no sistema e os relacionamentos entre eles sob uma perspectiva conceitual [FOW00] visando identificar algumas classes do sistema segundo os principais conceitos levantados na fase anterior.

O catálogo (`Catalogo`) é composto por várias linhas de catálogo (`LinhaCatalogo`). Cada linha é associada a um item de estoque (`ItemEstoque`). Um item de estoque pode estar associado a zero ou uma ordem de compra (`OrdemCompra`). Cada lista de compras (`ListaCompras`) está associada a um acesso do cliente ao sistema (`AcessoCliente`). Esta lista é composta por itens que o cliente seleciona (`ItemLista`) do catálogo. Um pedido (`Pedido`) é criado a partir das informações de uma lista de compras e associado a um cliente (`Cliente`). Um pedido é composto de várias linhas (`LinhaPedido`), onde cada linha é associada a um item de estoque. Cada lançamento contábil (`LanContabil`) é associado a um pedido assim como cada nota fiscal (`NotaFiscal`) é associada a um pedido e a um cliente.

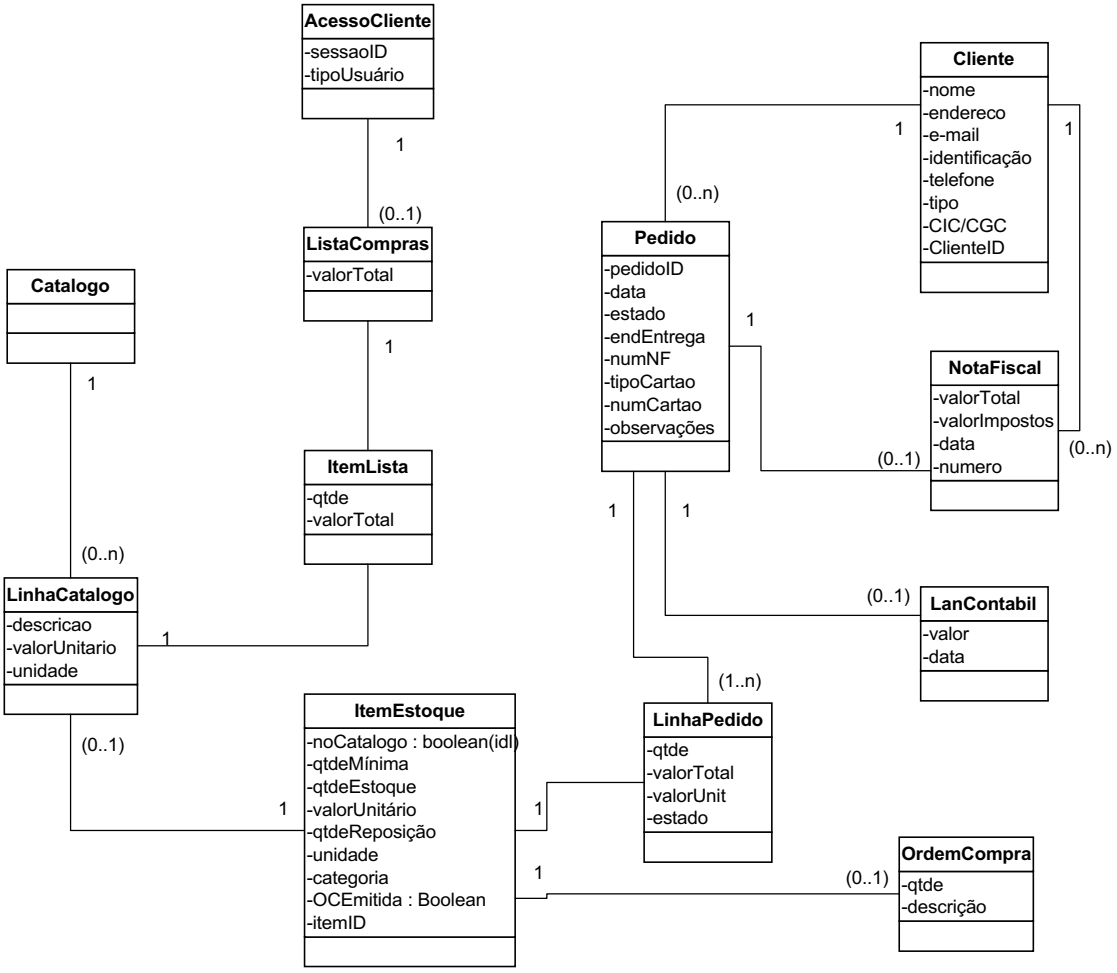

Fig.4.25 - Diagrama de classes do sistema Casarão

4.5.2 Diagramas de interação

Diagramas de interação visam esclarecer de que forma um grupo de objetos colabora entre si a fim de realizar uma determinada tarefa. Os processos de escolha de produtos pelo cliente, confirmação da compra de produtos, separação de material pelo Estoque e despacho de material pela Expedição são descritos através de diagramas de seqüência.

A figura a seguir apresenta o diagrama de seqüência que descreve a interação dos objetos no processo de seleção de itens a serem comprados pelo cliente. O cliente escolhe o produto através de um catálogo e vai adicionando à sua lista de compras. O cliente pode ainda alterar as quantidades de um item, devolver um item escolhido ou devolver todas as compras.

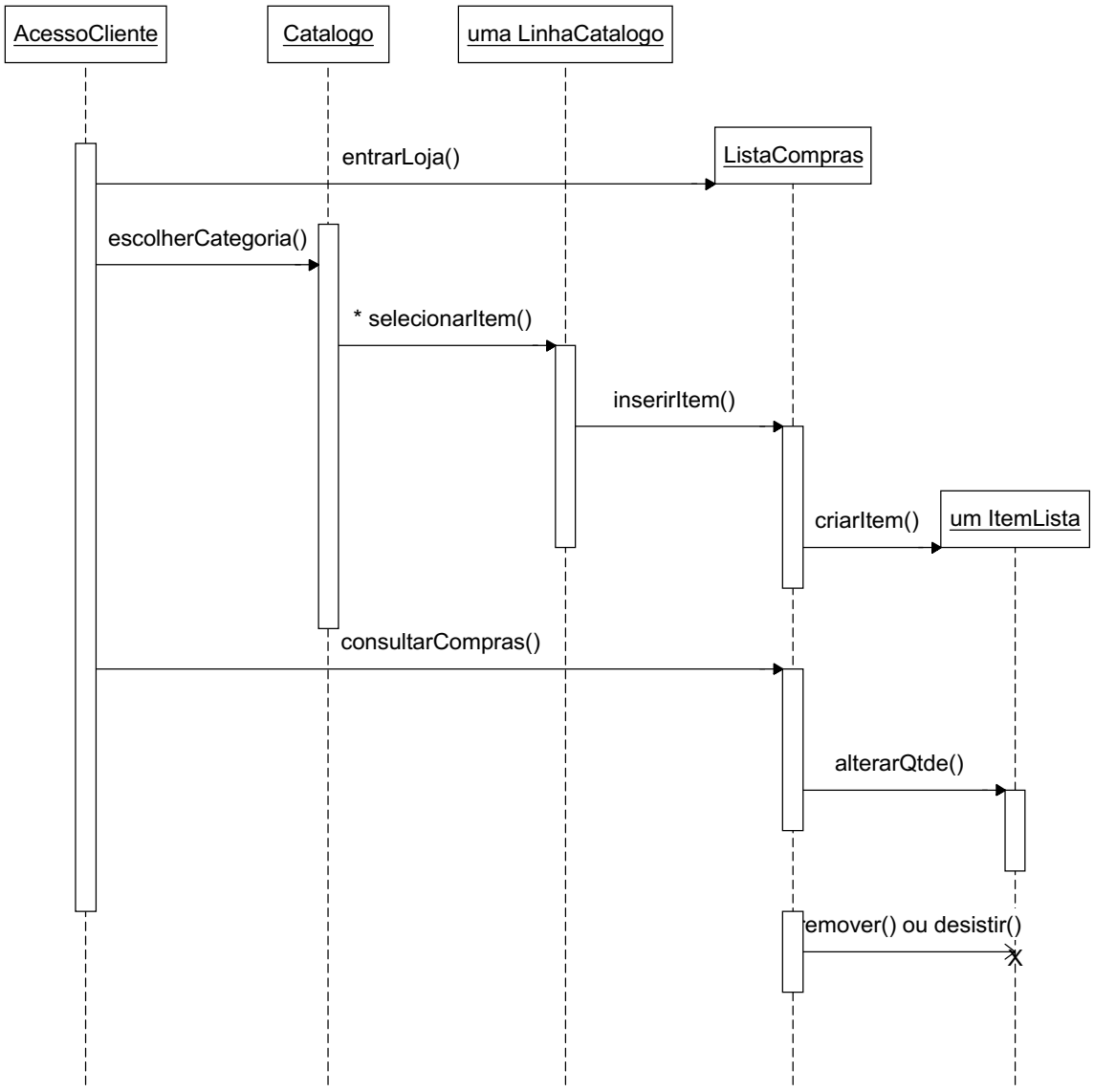

Fig. 4.26 - Diagrama de seqüência – escolha de itens na loja virtual

A figura a seguir apresenta o diagrama de seqüência que descreve a interação dos objetos quando um cliente confirma a compra de sua lista de itens e seu pagamento é aprovado.

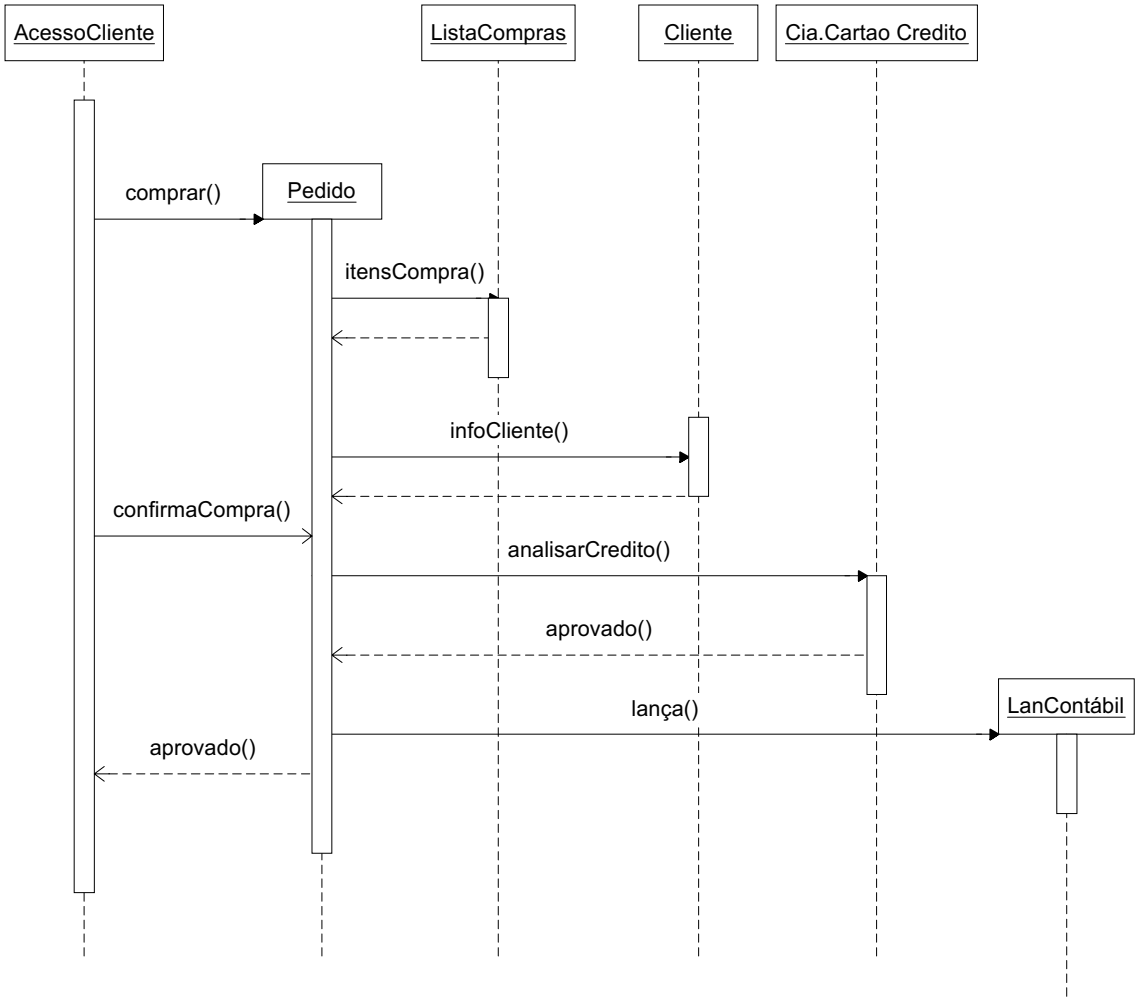

Fig. 4.27 - Diagrama de seqüência - confirmação de pedido de compra

A figura a seguir apresenta o diagrama de seqüência que descreve o processo de separação de material pelo estoque. Para cada item do pedido, o material é separado. Se não existir estoque suficiente para um determinado item, é emitida uma ordem de compra e o pedido fica em estado de espera. Quando o material é reposto, a linha do pedido referente ao item é atendida e o estado do pedido é recalculado.

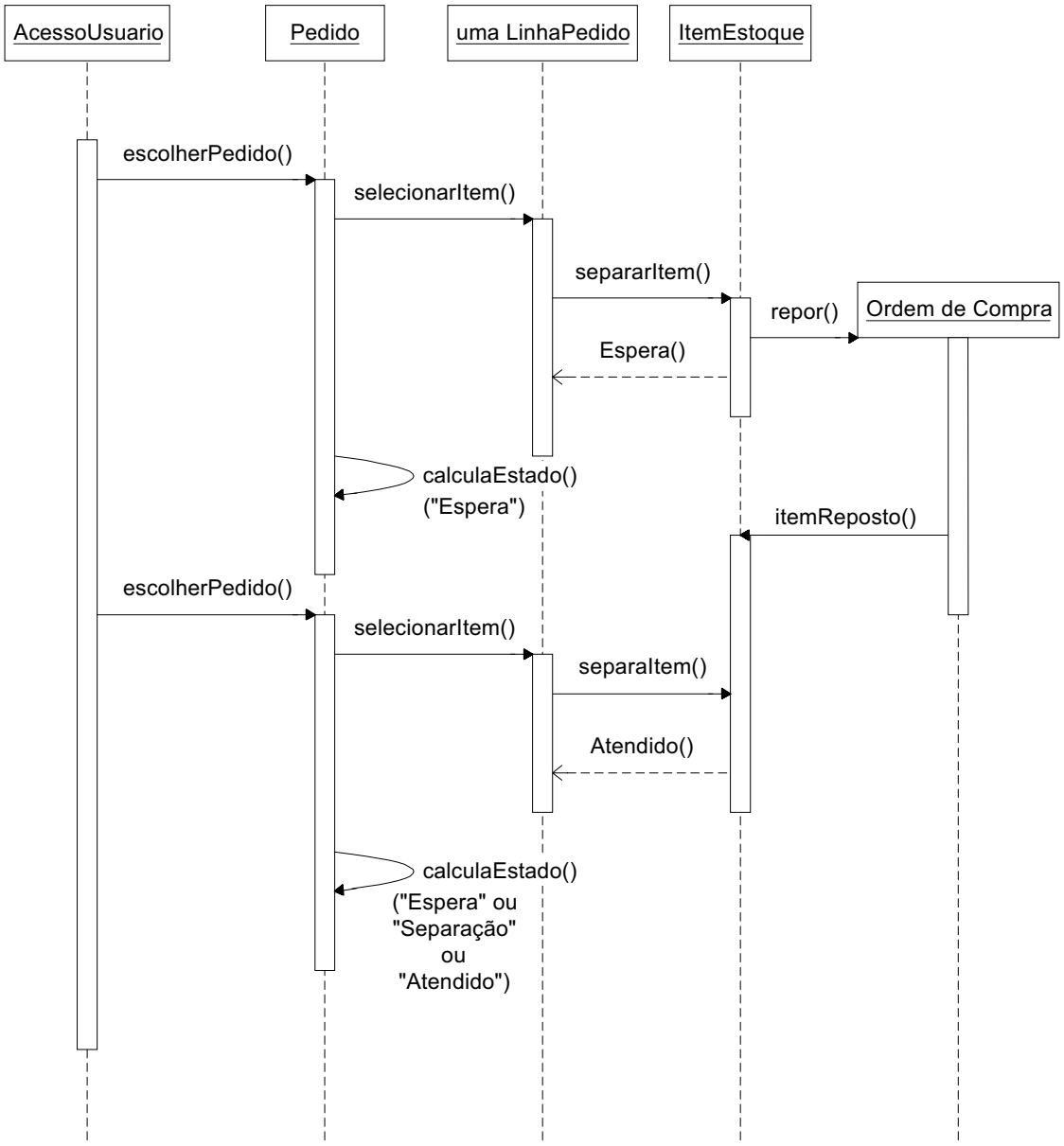

Fig. 4.28 - Diagrama de seqüência – separação de material de estoque

A figura a seguir apresenta o diagrama de seqüência que descreve o processo de expedição. Para cada pedido é emitida uma nota fiscal de saída e o pedido então é colocado em estado de “Expedição”. Quando o pedido é despachado, seu estado muda para o “Despachado”.

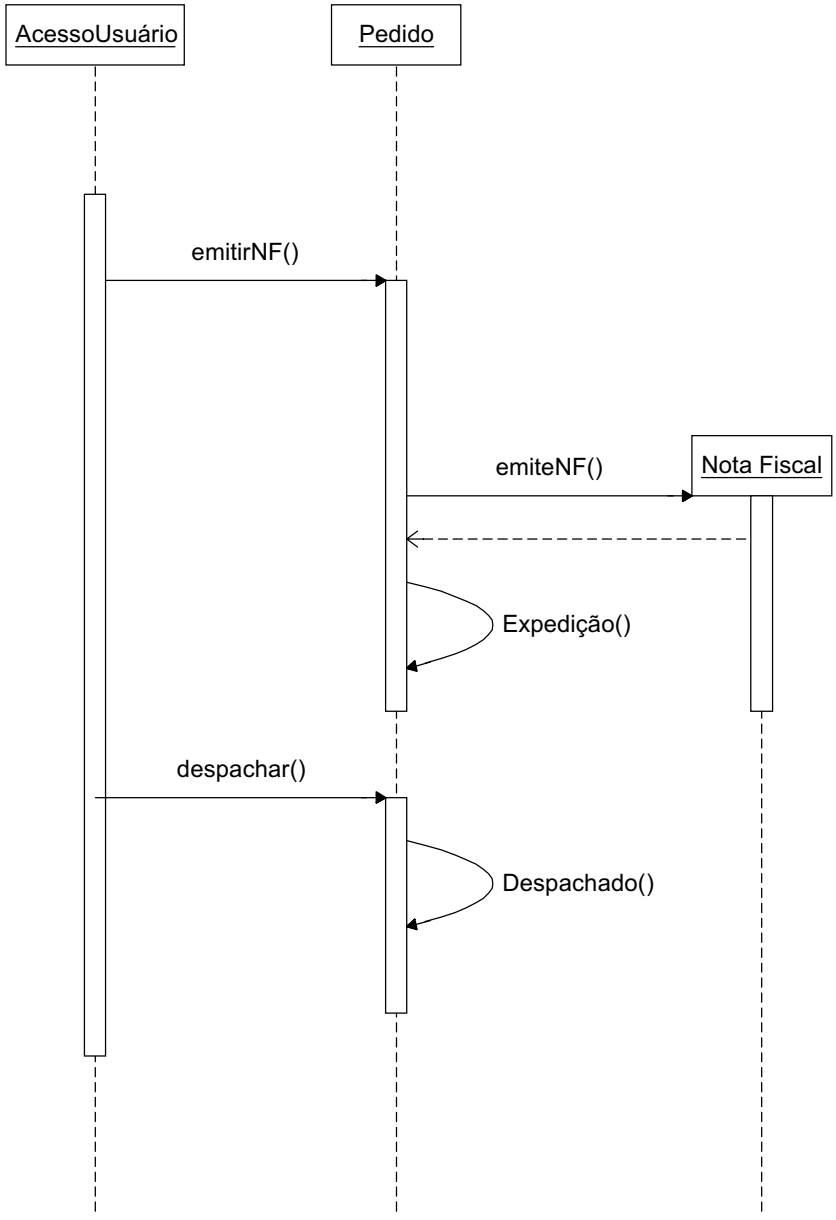

Fig. 4.29 - Diagrama de seqüência – expedição de material

4.5.3 Diagramas de estados

Diagramas de estados são utilizados para descrever os estados pelos quais um sistema evolui. A Fig. 4.30 ilustra o diagrama de estados associado a um objeto do tipo Pedido.

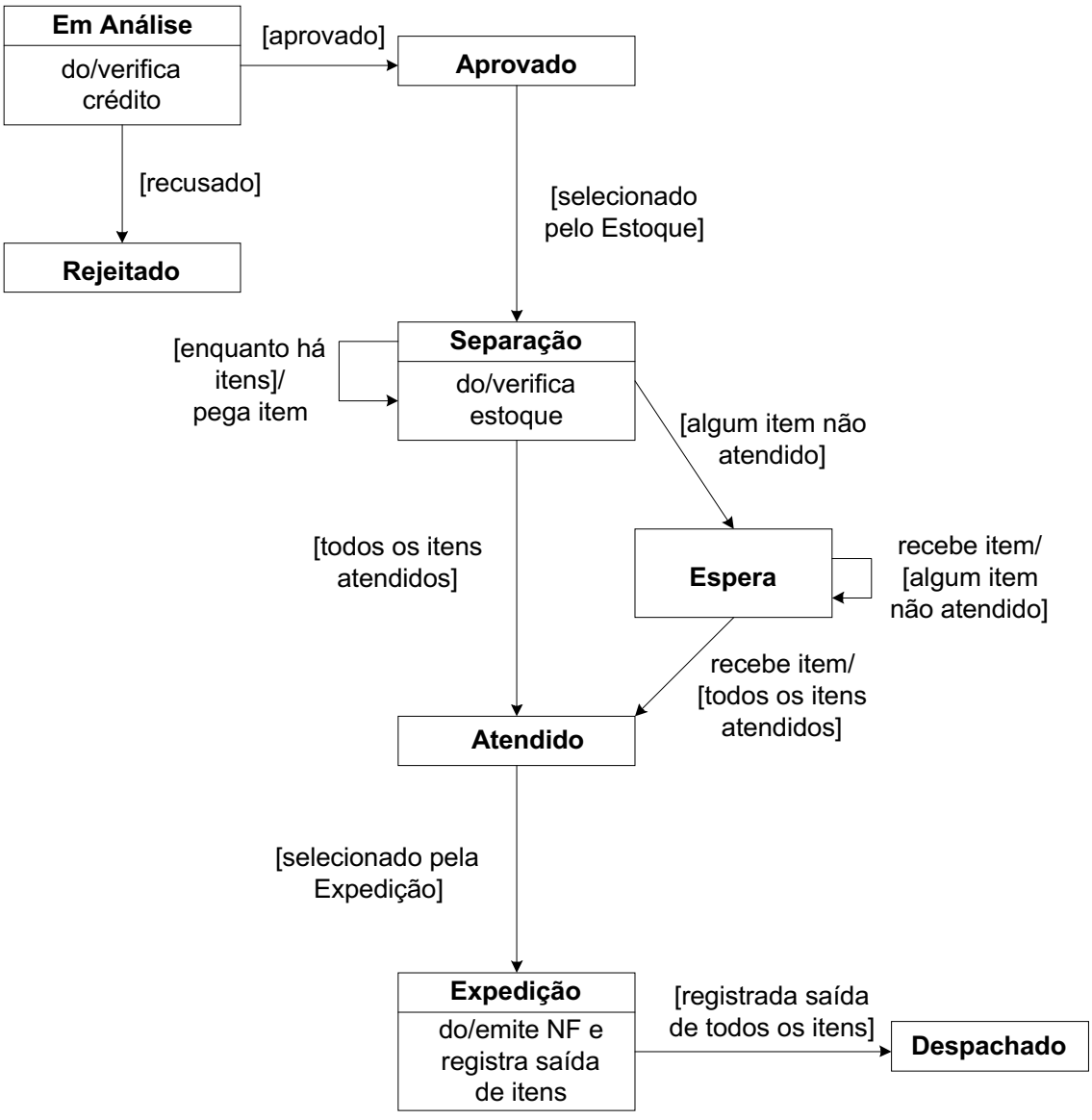

Fig. 4.30 - Diagrama de estados de objetos Pedido

4.5.4 Aspectos da modulariade do sistema Casarão

Através do levantamento apresentado, observamos que o tipo de interação existente entre o sistema Casarão e os atores Cia. de Cartão de Crédito e Financeiro podem ser vistos como mensagens trocadas entre sistemas independentes. Por outro lado, o conjunto de funcionalidades associadas tanto ao ator Estoque quanto ao ator Expedição quase não envolvem interação com os outros atores do sistema. Sendo assim, podemos considerar estes dois últimos como sendo sub-sistemas do sistema Casarão.

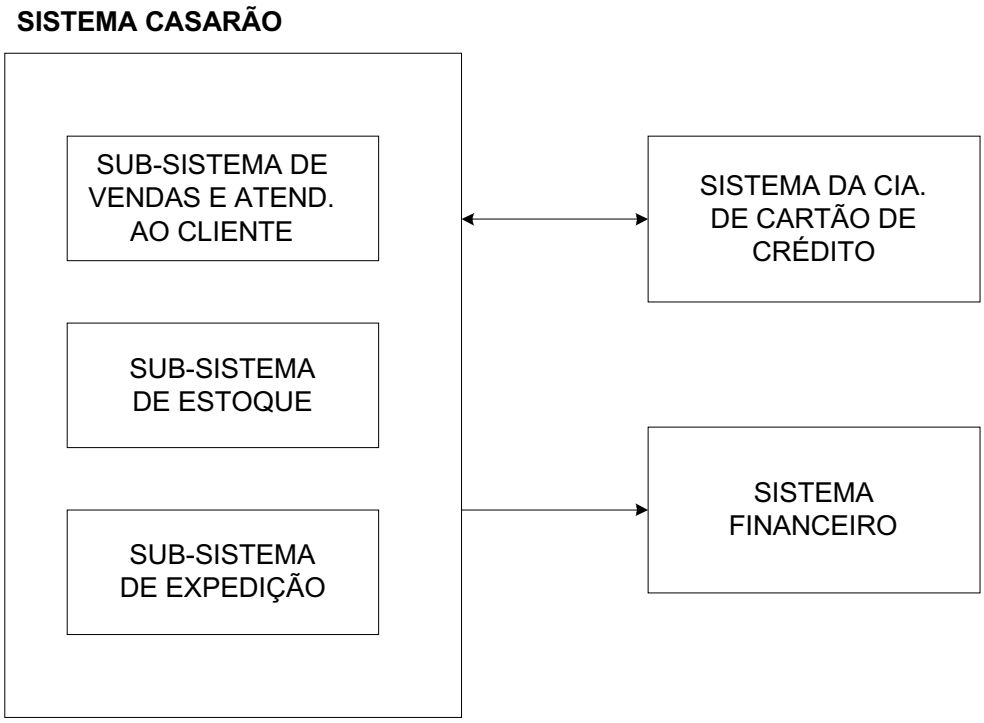

Fig. 4.31 - Decomposição do sistema Casarão

4.6 Conclusão

Este capítulo apresentou, a título de estudo de caso, uma análise detalhada dos requisitos funcionais e do comportamento de um sistema de vendas pela Internet denominado *Sistema de Vendas Online Casarão*. Foram utilizadas as ferramentas do UML para descrever o sistema sob os aspectos estático e dinâmico. O próximo capítulo apresenta uma proposta de projeto para este modelo de concepção.

Capítulo 5

Uma Arquitetura Baseada em Ambientes para o Projeto do Sistema Casarão

5.1 Introdução

Durante a última década foi possível observar o aumento do tamanho e da complexidade dos softwares. Desta forma, o desenho, a especificação e a análise da estrutura de um sistema como um todo deu origem a um novo tipo de problema, evidenciando a arquitetura de software como uma área importante da engenharia de software. A arquitetura de software pode ser entendida como sendo a fase de definição das características da estrutura de um sistema. Estas características incluem, entre outras, a organização do sistema em conjuntos de componentes, estruturas de controle globais, protocolos de comunicação, sincronização e acesso a dados, funcionalidades e composição dos elementos especificados na fase de desenho, escalonamento e eficiência, dimensionamento de evolução e a escolha entre as diferentes alternativas de projeto de sistemas [DAV01].

Dada a dinâmica da evolução tecnológica e freqüentes mudanças de contexto de um negócio, ao optar por uma solução para aplicações Web, as empresas buscam arquiteturas que, entre outros, possam satisfazer os seguintes requisitos [ANG00]:

a) prover consistência no tratamento de serviços;
b) ser flexível, escalonavel e adaptável para suportar as mudanças do ambiente de negócio;

c) reduzir os custos de aquisição de tecnologia;
d) considerar os recursos existentes - por exemplo, sistemas legados;
e) prover uma forma de realizar modificações a curto prazo que contribuam para a evolução coerente do sistema a longo prazo.

Dentro deste contexto existem várias opções tecnológicas para a implantação de sistemas de comércio eletrônico. Entre as atualmente mais utilizadas estão as tecnologias CGI (cf. ANEXO 1 Seção 3.4), ASP (Active Server Pages) [ASP02], PHP (Hypertext Preprocessor) [PHP02] e Servlets/JSP [SUN02]. Para este trabalho optou-se pela tecnologia Servlets/JSP por ser de distribuição gratuita, oferecer um conjunto completo de facilidades para a administrar pedidos HTTP e para construir dinamicamente páginas Web, ser multi-plataforma e principalmente por poder utilizar todas as funcionalidades diponibilizadas pelas APIs de Java.

Uma das propostas de utilização da tecnologia Servlets/JSP, conhecida como Modelo II (cf. Seção 3.2 do Cap. 3), propõe uma arquitetura de aplicação baseada no padrão MVC (*Model/View/Controller*) [GAM94]. Segundo esta proposta, enquanto os servlets são responsáveis por executar um fluxo de tarefas assumindo o papel de *Controller*, as páginas JSP ficam a cargo da apresentação da resposta ao usuário assumindo o papel de *View*. As informações utilizadas pelas páginas JSP para a construção das visões são disponibilizadas pelos servlets através da criação de objetos chamados *beans*, que assumem o papel de *Model* no padrão MVC. Essa arquitetura pode ser vista como um refinamento da arquitetura em três camadas com servlets e *beans* fornecendo um modelo de interação entre as camadas de apresentação e de lógica da aplicação. Para aplicações menos complexas, a camada de lógica da aplicação é totalmente absorvida pelo comportamento dos servlets.

Este capítulo propõe um projeto para o Sistema de Vendas *Online* Casarão seguindo a arquitetura baseada no padrão MVC, mas com servlets integrando-se a uma arquitetura para a camada de lógica de aplicação baseada em uma noção de ambientes, onde um ambiente é definido o mais próximo possível de um ambiente real de uma loja. Nesta proposta, servlets assumem o papel de agentes atuando nesses ambientes para atender as

diferentes funcionalidades do sistema. Os ambientes, por sua vez, oferecem os recursos necessários ao atendimento dessas funcionalidades. Nesses ambientes os servlets executam ações visando a criação dos *beans* que serão utilizados por uma página JSP com o objetivo de apresentar ao usuário a visão do ambiente que ele teria no momento do término da execução de uma ação como se estivesse em uma loja real.

Sem perda de generalidade, o projeto apresentado neste capítulo restringe-se à descrição da arquitetura que atende apenas às funcionalidades do diagrama de casos de uso associados ao ator cliente apontados pela análise apresentada no capítulo anterior (cf. Seção 4.4.2 do Cap. 4) que são os mais representativos para ilustrar essa proposta de arquitetura.

O restante deste capítulo é divido em cinco seções que abordam, respectivamente, uma introdução à arquitetura de software e ao conceito de ambientes, uma visão geral da arquitetura do sistema, a identificação e a definição dos ambientes, análise e definição dos servlets e páginas JSP, análise do comportamento do sistema e, por fim, a conclusão do capítulo.

5.2 Arquitetura de software

Alguns requisitos importantes de um software como portabilidade, escalabilidade, eficiência, corretude e interoperabilidade podem ser atingidos com a ajuda de um bom desenho arquitetural. A fase de definição da arquitetura de um software serve basicamente como uma ponte entre a fase de especificação dos requisitos do software e a fase de codificação, como ilustrado na Fig. 5.1 [GAR01].

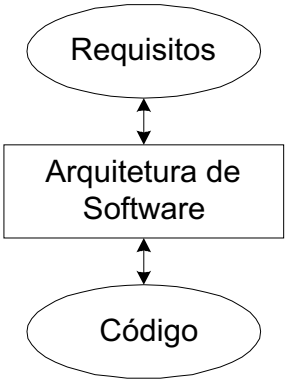

Fig. 5.1 - Arquitetura de software como uma ponte.

Por ser uma área relativamente nova, ainda não há um consenso com relação à terminologia, notação ou princípios bem definidos para a caracterização da arquitetura de um software complexo. Entretanto, surgiram algumas notações que provêem tanto um *framework* conceitual para a descrição da arquitetura de um software quanto uma sintaxe para sua caracterização. Estas notações são conhecidas como *Architecture Definition Languages* (ADLs) e servem a diferentes propósitos. Exemplos de ADLs incluem a notação *C2* utilizada para descrever sistemas de interface com usuário e a notação *Darwin* que é utilizada para a análise de sistemas distribuídos baseados em troca de mensagens. Entretanto, alguns pesquisadores têm advogado a utilização da notação UML para descrever arquiteturas de sistemas orientados a objetos [MED99].

Atualmente, é possível identificar vários padrões de arquitetura que permitem que analistas descrevam sistemas complexos utilizando abstrações que facilitem o entendimento do sistema como um todo. Embora não possuindo uma definição específica, este padrões podem ser descritos através de um vocabulário composto basicamente por três elementos principais, que são [GAR94]:

a) os componentes do sistema;
b) os conectores, que são as interações entre os componentes;
c) as restrições das possíveis combinações entre componentes do sistema.

A seguir são identificados alguns padrões de arquitetura bastante utilizados [GAR94, DAV01].

a) Tubos e Filtros

Cada componente do sistema tem um conjunto de entradas e um conjunto de saídas. Os componentes são organizados de forma que as saídas de um componente são conectados às entradas de outros.

b) Invocação implícita baseada em evento

A ocorrência de um determinado evento é anunciada pelo sistema a todos os componentes que se registraram para este evento. Desta forma, os componentes que recebem a notificação executam então as tarefas determinadas para a ocorrência deste evento.

c) Sistemas em camadas

Os componentes são organizados em forma de camadas. Cada camada fornece serviços à camada superior e solicita serviços à camada inferior.

d) Repositórios

Esta arquitetura é composta basicamente de duas entidades: uma estrutura central que representa o estado do sistema e um conjunto de componentes independentes que manipulam esta estrutura. A arquitetura *blackboard* é um exemplo desta arquitetura, onde o quadro negro representa a estrutura de dados central.

e) Cliente/Servidor

Apresenta os componentes classificados em dois grupos: servidores e clientes. Os componentes servidores oferecem serviços que são requisitados pelos componentes clientes. Esta é a arquitetura mais utilizada por aplicações Web.

f) Arquiteturas de domínio específico

São voltadas para uma família específica de aplicações. Por exemplo, sistemas de aviação ou sistemas ligados à área automobilística.

g) Arquiteturas híbridas

Os componentes do sistema podem apresentar arquiteturas diferentes.

h) Arquiteturas baseadas em agentes

Os componentes do sistema são vistos como entidades de software autônomas chamadas agentes. São muito utilizados em arquiteturas de sistemas distribuídos e multi-processamento distribuídos.

5.2.1 O conceito de ambientes

As arquiteturas baseadas em agentes estão sendo cada vez mais utilizadas por aplicações Web. No contexto da Internet, um agente pode ser interpretado como sendo "uma entidade autônoma que interage com seu ambiente e com outros agentes de forma pró-ativa a fim de atingir seus objetivos. Estes agentes tipicamente representam diferentes usuários e co-existem em um mesmo ambiente." [WOO00]

Arquiteturas baseadas em agentes evidenciam a necessidade da especificação dos ambientes onde os agentes possam co-existir. Um agente pode trocar informações com outros agentes que se encontram no mesmo ambiente e utilizar recursos específicos do ambiente em que se encontra para atingir seus objetivos. Apesar de muitos trabalhos terem sido dirigidos à tecnologia de agentes, pouca atenção é dirigida à definição de ambientes.

Um trabalho que aborda diretamente a formalização do conceito de ambientes é o modelo chamado *The Folder Calculus* [CAR99]. Neste trabalho, o autor ressalta que o fato da Internet ser uma WAN composta por diferentes tecnologias evidencia alguns aspectos que antes não eram observados em redes menores, como por exemplo, a existência de localizações virtuais delimitadas por barreiras de segurança, o aumento do tempo de

latência decorrente da grande distância entre as localizações físicas, maior sucetibilidade à falhas e maior probabilidade de flutuações de largura de banda. Neste contexo destaca-se a tecnologia de computação móvel que minimiza os problemas criados pelas quatro situações devido a sua característica básica : a mobilidade. Possuindo esta característica, um processo pode mover-se através de localizações virtuais atravessando barreiras (desde que tenha permissão para isto), pode diminuir o tempo de latência movendo-se para o ambiente em que tem que ser processado e depois retornando com as informações processadas, pode minimizar falhas alterando seu caminho para evitá-las, assim como pode aliviar as alterações de largura de banda escolhendo um caminho melhor ou levando consigo os parâmetros necessários para estabelecer conexões conforme as suas necessidades.

Este trabalho propõe a construção de um novo modelo que inclui ferramentas que possam manipular as noções de localidade, mobilidade e capacidade de transpor barreiras de segurança. A principal entidade abstrata deste novo modelo é o conceito de ***ambiente,*** que pode ser entendido como um espaço móvel onde os processos existem. Um ambiente é, por definição, um espaço que pode ser delimitado, é indentificado por um nome, pode conter processos ativos, pode conter sub-ambientes e move-se integralmente com todos os seus componentes. O modelo é descrito informalmente através de uma analogia com pastas de um arquivo, onde :

a) uma pasta representa um ambiente;
b) cada pasta tem uma etiqueta com seu nome;
c) cada pasta contém *gremlins* que fazem com que a pasta se mova de um lugar para outro.

As operações atômicas que representam mobilidade são descritas através das seguites ações, onde *m* e *n* representam pastas distintas:

a) entrar em uma pasta (*in m*), onde a pasta n entra na pasta m tornando-se uma subpasta de m.
b) sair de uma pasta (*out m*), onde ocorre o processo inverso, a pasta n deixa a pasta m

c) abrir uma pasta (*open m*), onde a pasta m é aberta, sua capa descartada e seu conteúdo deixado no lugar da pasta.
d) copiar uma pasta, onde uma pasta e seu conteúdo podem ser instantaneamente copiados.
e) criar um nome, onde para criar um nome é necessário criar um carimbo com esse nome. Cada carimbo pode carimbar o nome em várias pastas. No entanto, se este carimbo passa por um processo de cópia, o carimbo replicado deve ser visto como um novo carimbo ainda que cotenha o mesmo nome. É importante notar que dada uma pasta sempre é possível identificar qual carimbo a originou.

As operações atômicas que descrevem a comunicação são descritas através das seguintes ações :

a) ler uma mensagem, onde os *gremlins* de uma pasta podem ler uma mensagem que foi deixada nas proximidades da pasta;
b) o conteúdo da mensagem pode dar uma determinada habilidade ao leitor. Esta habilidade pode ser a de criar um nome ou a de "navegar" executando ações de entrar, sair e abrir uma pasta. Esta ações podem ser compostas definindo assim um caminho (*path*).

Como resultado do estudo sobre o modelo de ambientes, o autor aponta algumas construções básicas que não são encontradas nas técnicas de programação atuais e que devem estar presentes na implementação de uma linguagem de programação para uma rede geograficamente distribuída. Algumas destas construções são:

a) Utilização do conceito de ambiente como principal entidade abstrata de programação;
b) Os ambientes devem ser referidos pelos nomes e não por *pointers*;
c) A natureza das localidades podem ser definidas como tipos de ambientes, pois alguns ambientes são móveis e outros não;

d) Na migração de um ambiente, todo o seu conteúdo é migrado em tempo de execução;
e) A comunicação no modelo estudado se restringe a interações locais, sempre dentro de um ambiente (entre ambientes pai e filho ou entre ambientes irmãos). Comunicações remotas necessitariam de outras primitivas em um nível mais alto, não descritas neste formalismo;
f) Estruturas de dados diversas podem ser construídas através das primitivas descritas no modelo. Cada estrutura pode ser descrita como sendo um ambiente que pode conter vários sub-ambientes. Qualquer referência à estrutura ou a alguma parte dela deve ser feita através de nomes para preservar a mobilidade;
g) Segurança é introduzida através da capacidade dos ambientes criarem chaves e testarem estas chaves nos limites de seu espaço.

5.3 Visão geral do sistema

A análise feita do *Sistema de Vendas Online Casarão* no capítulo anterior revelou dois pontos importantes a serem considerados na definição da arquitetura do sistema:

a) A comunicação entre cliente e sistema é feita através da rede Internet onde a troca de mensagens segue o padrão solicitação/resposta, caracterizando o sistema Casarão como sendo um conjunto de aplicações Web do tipo cliente/servidor.

b) Grande parte dos processos descritos manipula informações persistentes. Para garantir a integridade e consistência destas informações, o sistema Casarão utiliza um banco de dados comum à outras aplicações da empresa.

Reunindo as características acima às informações do capítulo anterior, propõe-se implantar o Sistema Casarão através de uma arquitetura do tipo cliente/servidor de três camadas utilizando os recursos das tecnologias Servlet e JSP através do servidor Tomcat

v.4. Esta proposta de arquitetura baseia-se no Modelo II (cf. Fig. 3.2 Seção 3.2) e tem na noção de ambientes o seu suporte intuitivo principal. A seguir é descrita a estrutura proposta em cada camada, que a seguir são ilustradas pela Fig. 5.2.

Camada de apresentação

Esta camada é responsável pelo gerenciamento das solicitações/respostas dos clientes feitas através da rede e respectivas interfaces gráficas A camada de apresentação é composta por um conjunto de Sevlets e páginas JSP. As páginas Web que fazem a interface entre cliente e sistema são todas implantadas através de páginas JSP. Cada solicitação é redirecionada pelo servidor Tomcat a uma entidade receptora, que pode ser uma página JSP ou um servlet. Caso a solicitação envolva passagem de dados, esta entidade é responsável pela consistência dos dados recebidos. Neste caso, se os dados estiverem com problemas, *beans* com mensagens de erro são anexados ao objeto implícito `request` (cf. Tabela 3.1 Cap. 3) e a solicitação é imediatamente redirecionada para uma página JSP que é encarregada de administrar uma resposta ao cliente. Caso contrário, a entidade receptora "visita" os ambientes definidos na camada de lógica da aplicação e utiliza os recursos neles disponíveis para criar os *beans* que serão utilizados para responder à solicitação do cliente. Em seguida, anexa os *beans* aos objetos implícitos correspondentes e redireciona a solicitação para uma página JSP. Esta, por sua vez, faz a inclusão dos dados contidos nos *beans* anexados à uma estrutura HTML e envia a resposta ao usuário.

Camada de lógica da aplicação

Esta camada contém a estrutura lógica da aplicação e é refinada em duas sub-camadas. São elas:

a) Sub-camada de ambientes

   Esta sub-camada é composta por classes que representam ambientes definidos o mais próximo possível dos ambientes reais do estabelecimento. Cada ambiente

oferece recursos específicos. Estes recursos são disponibilizados na forma de atributos fixos que são utilizados na apresentação da visão ao usuário, por exemplo, o nome do ambiente em que se encontra irá compor o título da página JSP, ou na forma de funcionalidades oferecidas dentro de cada ambiente, por exemplo, a lista dos pedidos feitos pelo usuário.

b) Sub-camada de depósito de funcionalidades
Composta por classes que realizam as funcionalidades exigidas pelos recursos disponíveis nos ambientes. Caso seja necessário fazer algum acesso ao banco de dados, é solicitado o serviço da camada de acesso à base de dados para recuperar ou inserir informações persistentes.

Camada de acesso à base de dados

Esta camada é responsável pelo gerenciamento de acessos à base de dados. Estes acessos são feitos utilizando uma interface padrão de comunicação com um banco de dados relacional chamada JDBC (*Java Database Connectivity*). Para cada tabela da base de dados existe uma classe nesta camada responsável por construir os comandos SQL para deleção, inserção e modificações de informações. Estas classes utilizam uma conexão comum à todos os acessos administrada por um único objeto que obedece o padrão *singleton* [GAM94].

A Fig. 5.2 apresenta a visão geral da arquitetura do sistema Casarão.

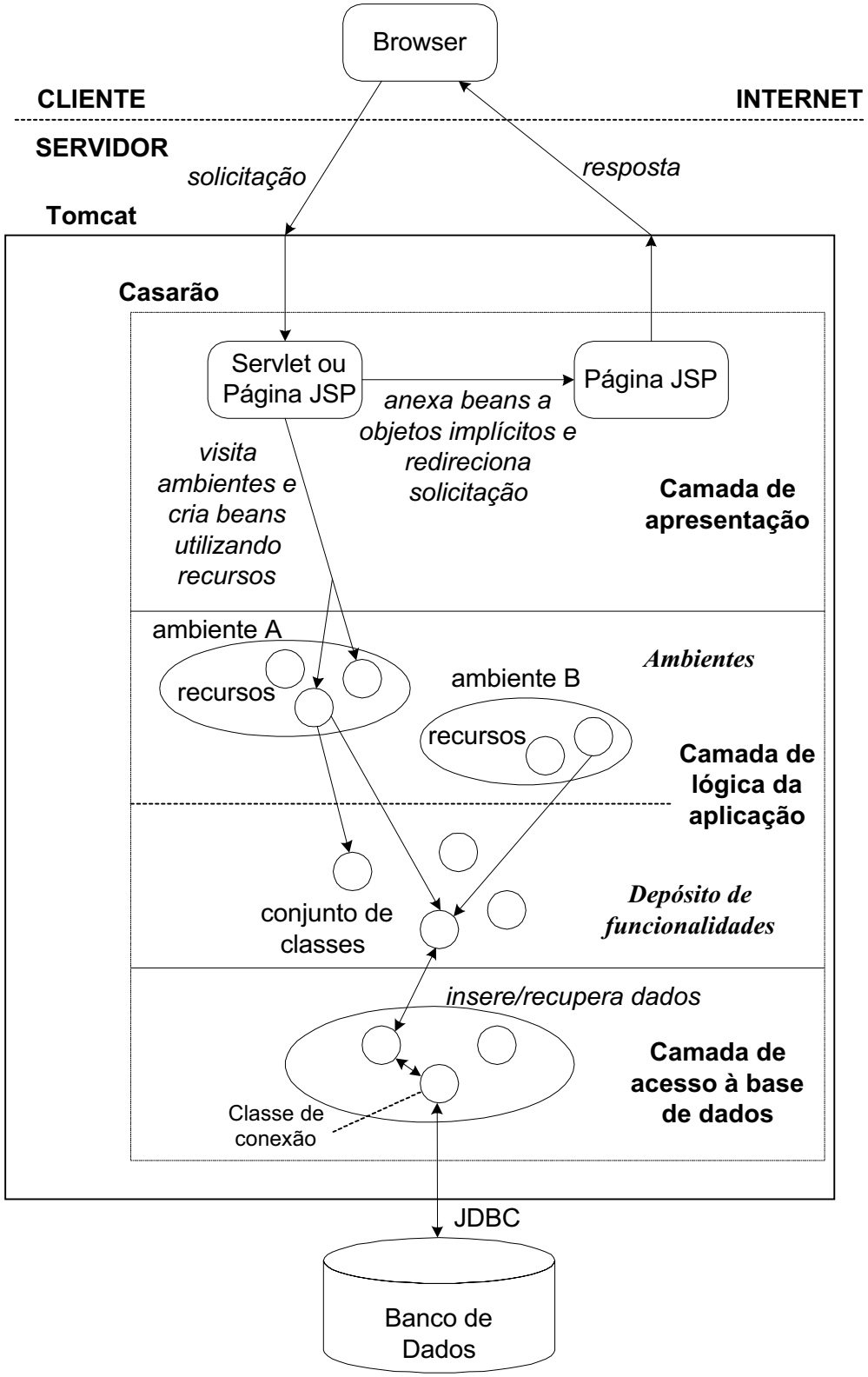

Fig. 5.2 - Visão geral da arquitetura do sistema Casarão.

A Fig. 5.3 ilustra o processo de criação e manipulação dos *beans* no sistema utilizando o exemplo no qual um cliente seleciona a opção PISOS entre as categorias oferecidas no catálogo de produtos (cf. Fig. 4.4 Cap. 4). O servlet solicitado para esta tarefa chama-se `CategCatalogo`. Primeiro o servlet adquire a identificação do cliente que servirá para obter a permissão de entrada no ambiente PISOS. Em seguida, entra no ambiente PISOS e utiliza os recursos disponíveis no ambiente, por exemplo, a lista de itens deste ambiente e o nome do ambiente, para construir os *beans* necessários. O servlet anexa estes *beans* ao objeto implícito `request` e redireciona a solicitação à página JSP correspondente, no caso, `VisaoDep.jsp`**.** Esta página então acessa os *beans* criados pelo servlet e monta a página de resposta ao cliente (cf. Fig. 4.5 Cap. 4), que, neste contexto, representa a visão do ambiente PISOS pelo cliente neste momento.

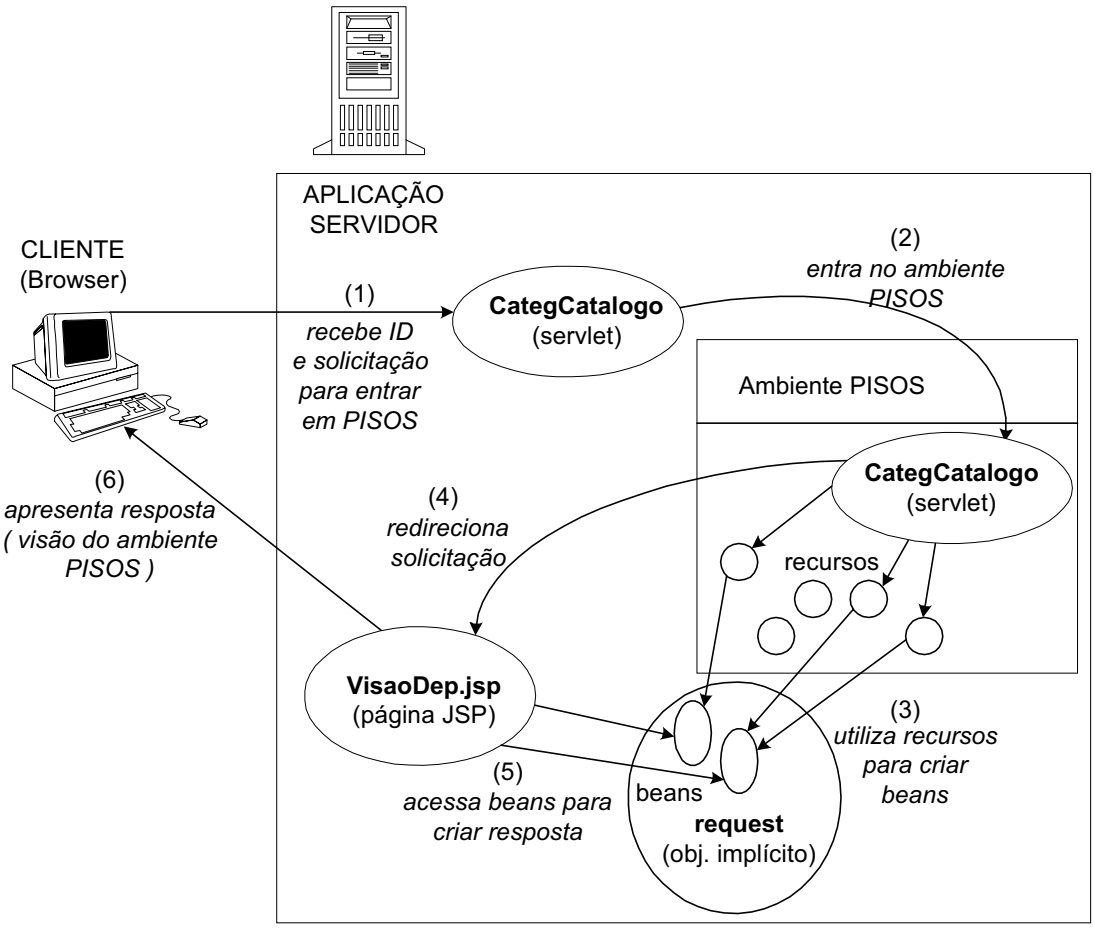

Fig. 5.3 - Exemplo de criação e utilização dos *beans*.

5.4 Ambientes

Ao entrar no estabelecimento Casarão, entre um grande leque de opções, um cliente pode optar por dirigir-se ao setor de pedidos para se informar sobre o estado de um pedido anterior, dirigir-se à loja para fazer novas compras, selecionar produtos, devolvê-los, dirigir-se ao caixa para efetivar sua compra, desistir das compras ou simplesmente sair da loja.

Para a identificação dos possíveis ambientes do sistema, foi elaborado uma planta arquitetônica virtual do estabelecimento Casarão, Fig. 5.4, que atendesse às funcionalidades apontadas na análise apresentada no capítulo anterior.

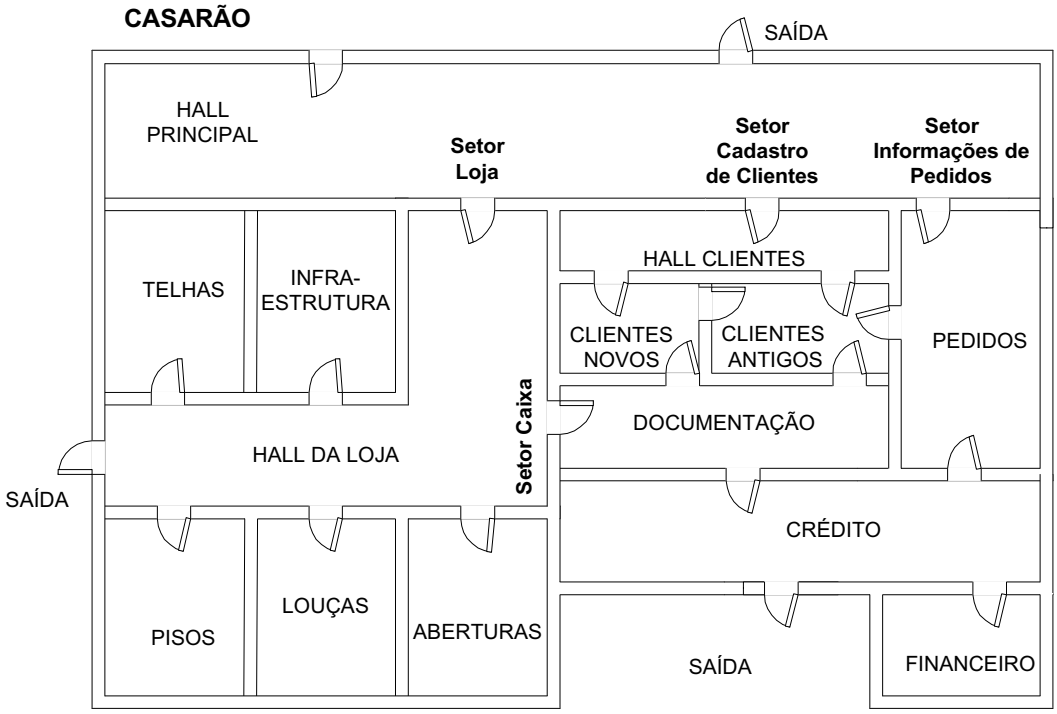

Fig. 5.4 - Planta arquitetônica virtual do estabelecimento Casarão.

Na planta virtual, cada ambiente é identificado por um nome. Exceto pelo ambiente global CASARÃO, todos os ambientes que são compostos por sub-ambientes são chamados de setores.

Segundo a planta virtual, um cliente ao entrar no ambiente HALL PRINCIPAL do estabelecimento Casarão pode escolher entre visitar o **Setor Loja**, dirigir-se ao **Setor de Cadastro de Clientes** ou ainda dirigir-se ao **Setor de Informações de Pedidos**.

Ao optar por visitar o **Setor Loja**, um cliente entra no HALL DA LOJA e pega um carrinho de compras. Em seguida faz suas compras nos ambientes PISOS, TELHAS, LOUÇAS, INFRA-ESTRUTURA E ABERTURAS. Estando no ambiente HALL DA LOJA, o cliente pode desistir das compras e dirigir-se ao ambiente SAIDA ou ainda voltar para o HALL PRINCIPAL. Caso queira efetivar sua compra, ele deve dirigir-se ao **Setor Caixa**.

Ao optar por visitar o **Setor Caixa**, o cliente será dirigido ao ambiente DOCUMENTAÇÃO. Neste ambiente é reunida toda a documentação necessária para formalizar o pedido de compra. Primeiro, o cliente é solicitado a apresentar seu cadastro de clientes. Se ele for um novo cliente, será dirigido ao ambiente CLIENTES NOVOS para fazer seu cadastramento. Caso contrário, seu cadastro será recuperado no ambiente CLIENTES ANTIGOS. Em posse do seu cadastro, o cliente fornecerá os dados para que o pedido de compra possa ser devidamente preenchido. O cliente é então dirigido ao ambiente CRÉDITO. Neste ambiente a Cia de Cartão de Crédito é consultada. Se o pagamento for recusado, o cliente é avisado e dirigido ao ambiente SAÍDA. Caso contrário, é feito um Lançamento Contábil no ambiente FINANCEIRO e o pedido é colocado no ambiente PEDIDOS para que sua entrega possa ser processada. É então fornecido um comprovante da compra ao cliente e o cliente é dirigido à SAIDA.

Estando no HALL DA LOJA, um cliente pode optar ainda por visitar o **Setor Cadastro de Clientes** ou o **Setor Informações de Pedidos**. No primeiro caso, o cliente será dirigido ao ambiente HALL CLIENTES. Neste ambiente o cliente pode escolher entre

entrar no ambiente CLIENTES ANTIGOS para atualizar ou consultar seu cadastro ou entrar no ambiente CLIENTES NOVOS para cadastrar-se como novo cliente. No segundo caso, o cliente será dirigido ao ambiente PEDIDOS. Neste ambiente, é solicitada a identificação de cliente no sistema Casarão (`UserID`[1]), que é confrontada com a identificação contida no seu cadastro. Caso a identificação esteja correta, o cliente é consultar todos os seus pedidos.

Ao analisar o possível comportamento de um cliente nos ambientes caracterizados pela Fig. 5.4, pode-se observar o seguinte:

a) As possíveis ações de um cliente podem ser executadas dentro de ambientes específicos. Por exemplo, se o cliente quiser informações sobre seu cadastro, ele o fará no ambiente CLIENTES ANTIGOS e não no ambiente PISOS.

b) Em um estabelecimento comercial, existem vários caminhos que um cliente ou funcionário pode percorrer. Estes possíveis caminhos são determinados pelas conexões existentes entre os ambientes – as portas. Por exemplo, um cliente não pode passar do ambiente PISOS diretamente para o ambiente CRÉDITO sem antes passar pelo ambiente DOCUMENTAÇÃO. Dessa forma as conexões entre ambientes determinam os possíveis fluxos de execução de tarefas a serem seguidos.

c) As entradas ou saídas dos ambiente podem exigir a execução de ações específicas. Por exemplo, ao entrar no ambiente HALL DA LOJA, o cliente *pega* um carrinho de compras e ao sair do ambiente CRÉDITO com um pedido aprovado, o cliente *recebe* um comprovante de suas compras.

d) As ações executadas sobre os ambientes dependem do estado do cliente. Eis alguns exemplos: a entrada no ambiente DOCUMENTAÇÃO só é permitida a clientes cujo carrinho de compras não está vazio; ao encontrar-se no ambiente PEDIDOS, um

[1] UserId é uma senha que o cliente fornece ao sistema para futuras consultas.

cliente pode consultar o estado dos pedidos referentes apenas à sua identificação; um cliente pode "pegar" um carrinho de compras somente se estiver entrando no ambiente HALL DA LOJA., isto é, vindo do ambiente HALL PRINCIPAL [2].

e) As possíveis saídas de um ambiente podem depender do último ambiente onde o cliente esteve. Por exemplo, um cliente que acabou de preencher o cadastro no ambiente CLIENTES NOVOS deve poder voltar apenas para o ambiente de onde veio, no caso, ou para o ambiente DOCUMENTAÇÃO ou para o ambiente HALL CLIENTES.

f) Alguns ambientes reais são compostos por sub-ambientes. Neste caso, podemos identificar certas características de um ambiente que vale também para seus sub-ambientes, indicando desta forma a existência de uma estrutura hierárquica. Por exemplo, o ambiente **Setor Loja** é composto pelos ambientes TELHAS, ABERTURAS, INFRA-ESTRUTURA, LOUÇAS e **Setor Caixa**. Se o número do telefone de contato do **Setor Loja** for modificado, então qualquer visão de um cliente que se encontra em qualquer sub-ambiente deste setor deve apresentar o novo número de telefone.

De acordo com a planta do estabelecimento Casarão, os ambientes do sistema foram primeiramente classificados em ambientes e sub-ambientes. Em seguida foi feita uma análise das circulações possíveis entre ambientes e, por último, foram identificados os recursos oferecidos por cada ambiente.

---

[2] Um cliente pode "entrar" no ambiente HALL DA LOJA ao sair do ambiente PISOS. Neste caso ele já deve possuir um carrinho de compras.

5.4.1 Identificação dos ambientes do sistema

A Tabela 5.1 apresenta o resultado da identificação dos ambientes do Sistema Casarão.

AMBIENTES	**SUB-AMBIENTES**
CASARÃO	HALL PRINCIPAL **Setor Loja** **Setor Cadastro de Clientes** **Setor Informações de Pedidos** FINANCEIRO SAÍDA
Setor Loja	HALL DA LOJA TELHAS PISOS INFRA-ESTRUTURA ABERTURAS LOUÇAS **Setor Caixa** CARRINHO
Setor Caixa	DOCUMENTAÇÃO CRÉDITO
Setor Cadastro de Clientes	HALL CLIENTES CLIENTES NOVOS CLIENTES ANTIGOS
Setor Informações de Pedidos	PEDIDOS

Tabela 5.1 – Ambientes e sub-ambientes do sistema Casarão.

5.4.2 Circulação entre ambientes

A Fig. 5.5 descreve as possíveis circulações entre os ambientes. As flechas identificam a existência de uma conexão e o sentido da circulação entre dois ambientes. O carrinho de compras pode ser visto como um ambiente móvel, que acompanha o cliente durante a sua estadia no estabelecimento. Os asteriscos indicam os ambientes onde um carrinho de compras pode ser utilizado. As observações associadas a um caminho entre dois ambientes indicam as restrições de entrada no ambiente destino ao utilizar este caminho.

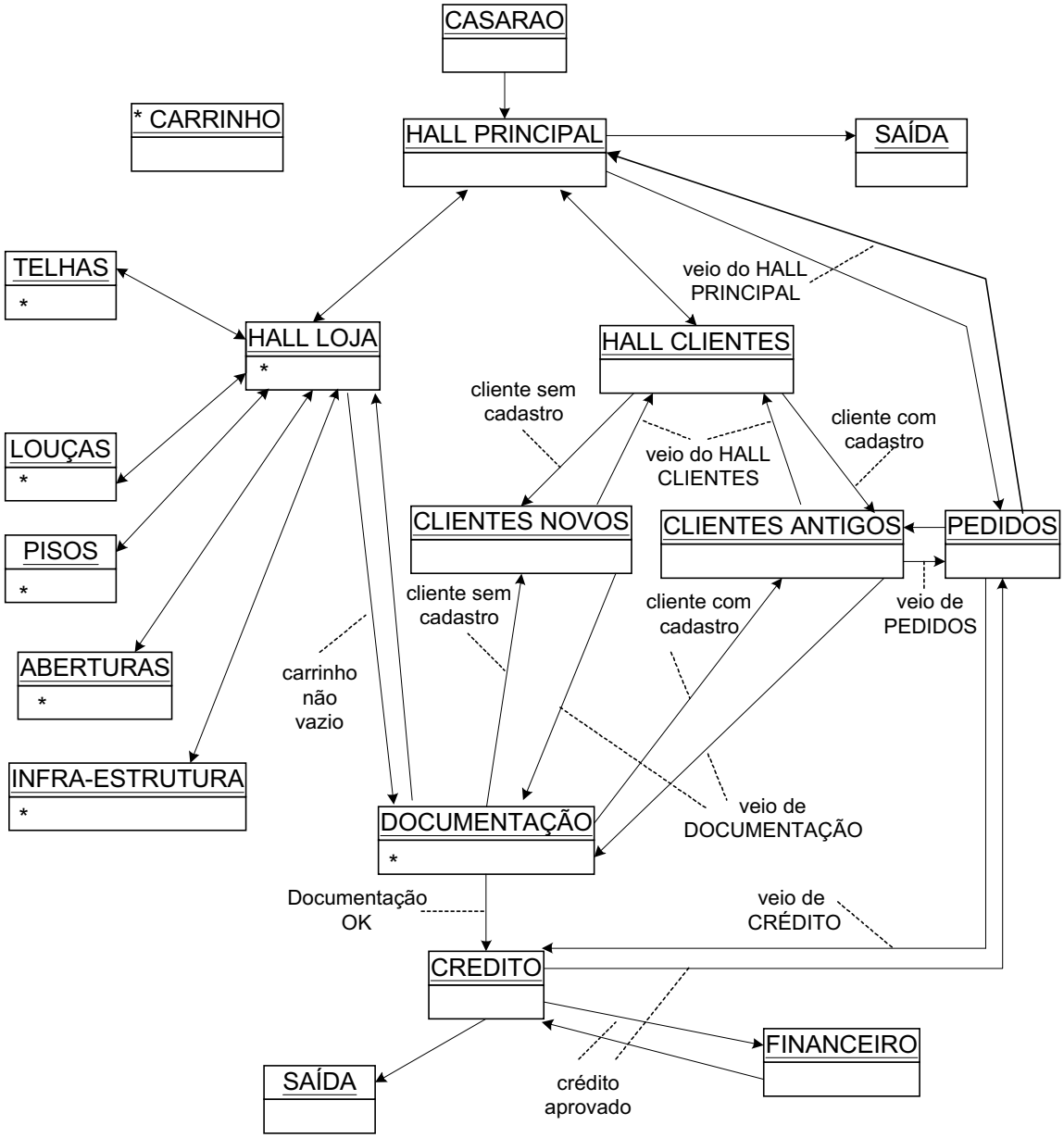

Fig. 5.5 - Circulação entre os ambientes do sistema Casarão.

5.4.3 Recursos oferecidos pelos ambientes do sistema

Em um ambiente são identificados dois tipos de recursos: os fixos e os funcionais. Os recursos fixos representam as características do ambiente como nome do ambiente e possíveis saídas. Os recursos funcionais representam as possíveis ações a serem executadas em cada ambiente. Algumas ações exigem que determinados pré-requisitos estejam satisfeitos. A tabela 5.2 apresenta os recursos funcionais disponíveis em cada ambiente. As observações entre parênteses indicam restrições à utilização do recurso.

AMBIENTE	**RECURSOS FUNCIONAIS**
HALL PRINCIPAL	Adquirir uma identificação.
HALL DA LOJA	Adquirir carrinho (permitido apenas a clientes vindos do ambiente HALL PRINCIPAL que ainda não possuam carrinho)
PISOS, ABERTURAS, LOUÇAS, TELHAS e INFRA-ESTRUTURA	Ver catálogo do ambiente
	Selecionar um item para ver detalhes.
	Adquirir certa quantidade de um item selecionado.
DOCUMENTAÇÃO	Adquirir formulário para pagamento (deve estar em posse do cadastro de clientes).
	Adquirir o pedido de compras formalizado.
CRÉDITO	Solicitar aprovação ou reprovação do crédito do cliente à Cia de Cartões de Crédito.
CARRINHO	Colocar determinada quantidade de um item no carrinho.
	Devolver um item.
	Devolver todas as compras.
HALL CLIENTES	
CLIENTES NOVOS	Adquirir ficha cadastral a ser preenchido.
	Solicitar inserção da ficha cadastral no Cadastro de Clientes.
CLIENTES ANTIGOS	Verificar a existência de um UserID no Cadastro de Clientes.
	Adquirir a ficha cadastral do cliente utilizando o UserID.
	Solicitar alteração da ficha no Cadastro de Clientes.

AMBIENTE	RECURSOS FUNCIONAIS
	Recuperar UserID de um cliente utilizando o nome do cliente e e CIC/CGC.
PEDIDOS	Adquirir todos os pedidos relativos a um UserID.
	Ver detalhes de um pedido selecionado.
	Receber novo pedido (o pedido deve estar no estado "Aprovado").
FINANCEIRO	Emitir lançamento contábil (para pedidos aprovados)
SAÍDA	Invalidar identificação do cliente.

Tabela 5.2 – Recursos funcionais disponíveis em cada ambiente.

5.4.4 O tipo **`Ambiente`**

O projeto definiu uma classe abstrata Ambiente com as seguintes características:

a) O nome do ambiente.
b) Um conjunto de entradas válidas.
c) Um conjunto de saídas válidas.
d) Um método booleano `entra()` que permite ou não a entrada de um servlet no ambiente. Este método tem como argumento um objeto `Indentificação` que identifica o usuário e será mais detalhado na Seção 5.5.1. Caso a entrada seja negada, uma mensagem identificando a causa da recusa é retornada em um objeto `Erro`.
e) Um método `inicializa()` que é invocado uma única vez na criação do ambiente. Fornece valores aos atributos do ambiente – `nome`, `entradas`, `saidas`, `saidasJSP` e `título` que farão parte das páginas que representam os ambientes na visão do usuário.
f) Um conjunto de métodos e atributos associados aos recursos específicos do ambiente. Caso ocorra algum problema no processo da utilização de algum recurso, o método correspondente retorna uma mensagem de erro em um objeto `Erro`.

As Figuras 5.6 e 5.7 apresentam os diagramas de classes dos ambientes identificados no sistema. Os parâmetros que retornam as mensagens de erro na utilização de recursos do ambiente foram suprimidos por uma questão de clareza. As classes `Cliente` e `Pedido` são definidas conforme o diagrama de classes apresentado no capítulo anterior (cf. Fig. 4.25 Cap. 4) .

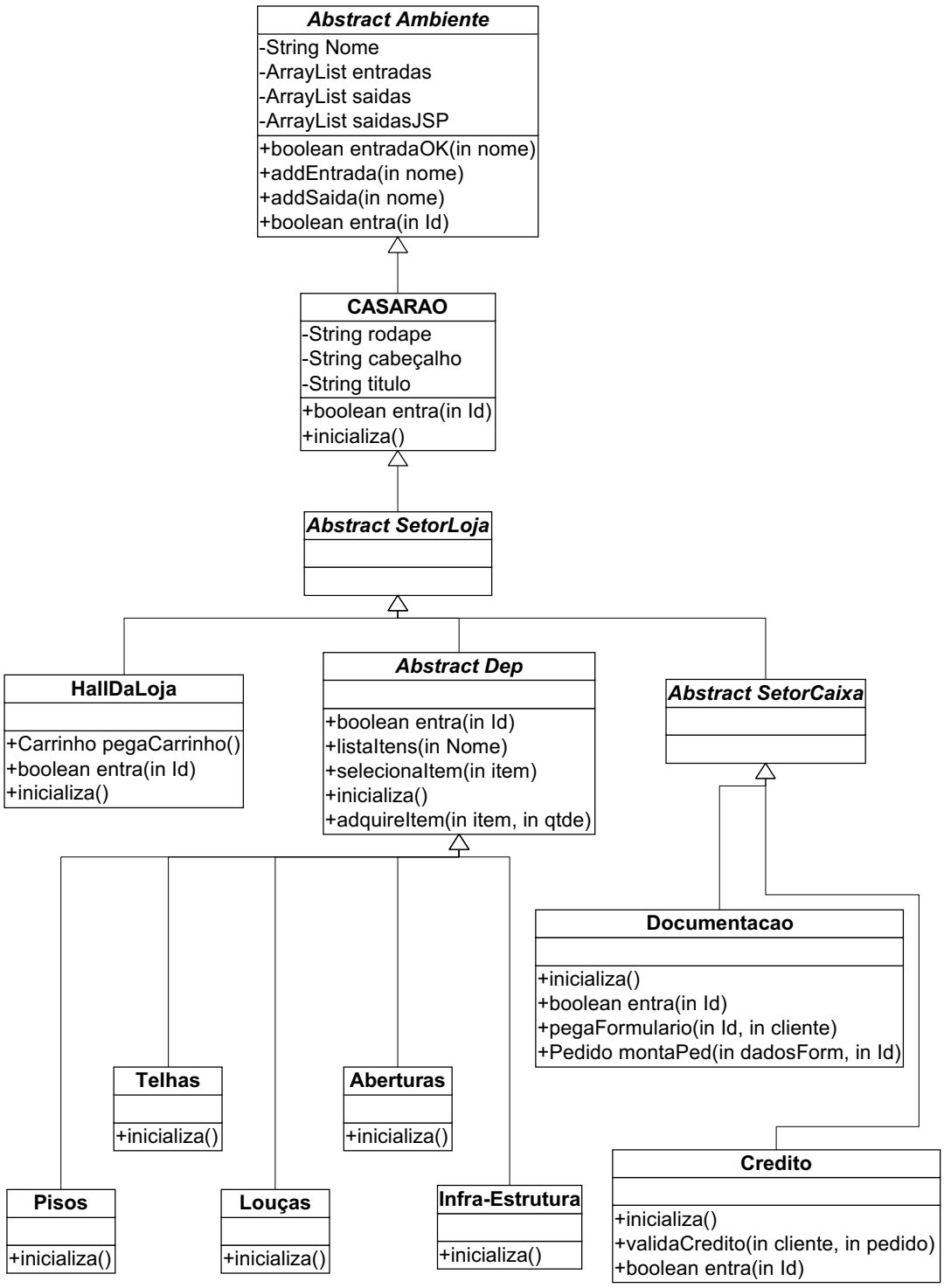

Fig. 5.6 - Diagrama de classes dos ambientes do sistema Casarão.

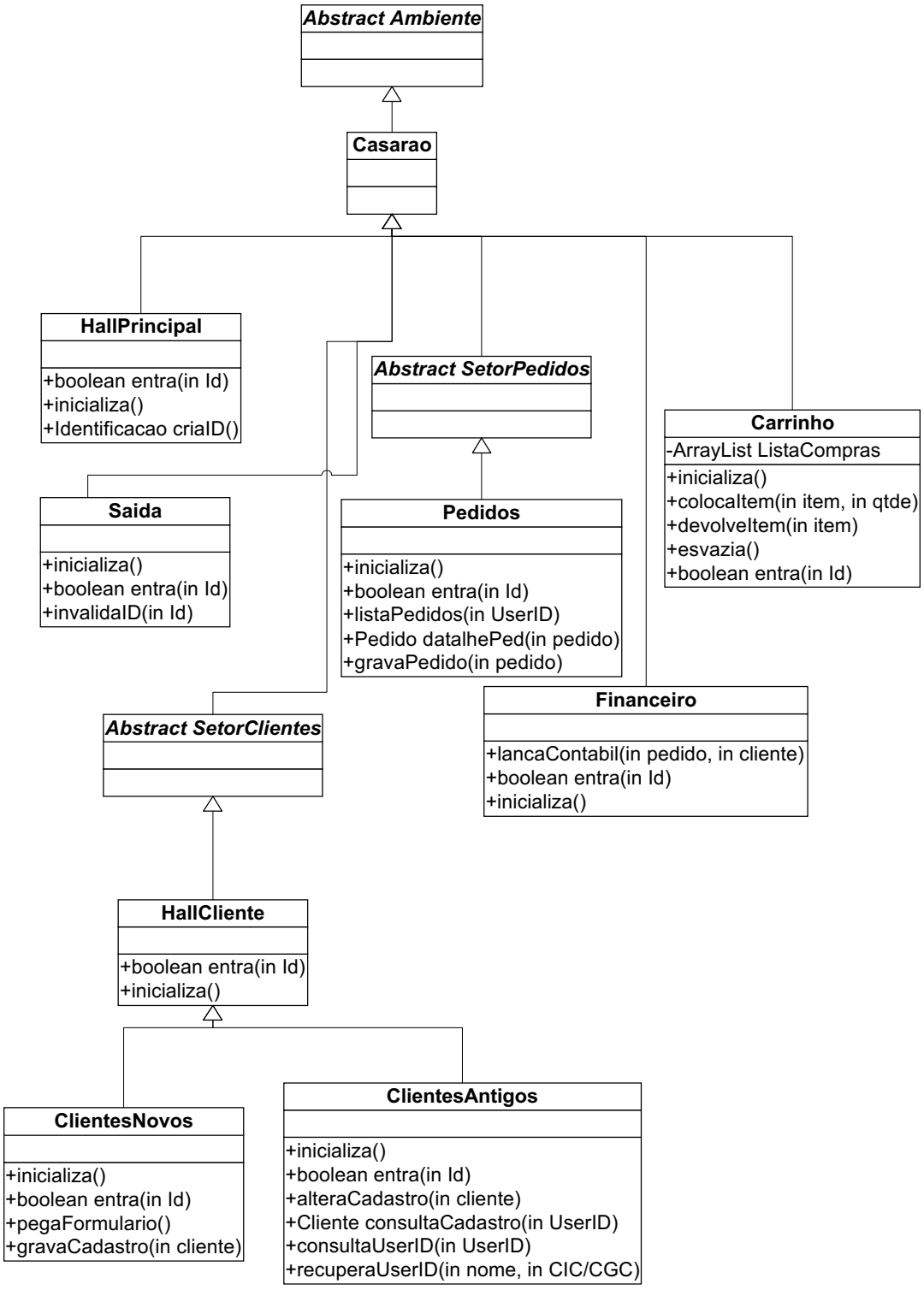

Fig. 5.7 - Diagrama de classes dos ambientes do sistema Casarão (continuação).

Segue-se agora algumas observações sobre esses diagramas.

a) Qualquer classe que represente um ambiente deve estender a classe abstrata `Ambiente`.

b) Ambientes semelhantes tais como os ambientes PISOS, TELHAS, ABERTURAS, LOUÇAS e INFRA-ESTRUTURA são definidos como sub-classes de uma classe abstrata chamada `Dep` que contém a definição dos recursos disponíveis nestes ambientes e as restrições de entrada.

c) Recursos como cabeçalho e rodapé, definidos na classe `HallPrincipal` são herdados por todos os ambientes. Já o recurso `saídas` e `saídasJSP` definidos na classe `Dep` são herdados por todas as classes `Pisos, Telhas, Aberturas, Louças e Infra-Estrutura`, uma vez que a única saída destes ambientes é o ambiente HALL DA LOJA.

d) Alguns recursos oferecidos pelos ambientes envolvem acesso às informações de um banco de dados. Para isto, cada ambiente utiliza serviços oferecidos por um conjunto de classes independente da estrutura de ambientes do sistema. Este conjunto de classes tem por objetivo disponibilizar as funcionalidades necessárias para que os recursos oferecidos por cada ambiente sejam manipulados adequadamente. Não será detalhado aqui porque insere-se no projeto como um subsistema independente.

5.5 Servlets e páginas JSP

A interação entre usuário e sistema segue o modelo solicitação/resposta. Cada solicitação, representando uma possível ação de um cliente no estabelecimento Casarão, é traduzida em ações a serem executadas pelos servlets correspondentes nos ambientes do

sistema. Por sua vez, os servlets resgatam informações que serão utilizadas por uma página JSP que apresentará ao cliente uma visão momentânea do ambiente em que ele se encontra.

5.5.1 Identificando clientes

Clientes são identificados pelos servlets através de um objeto `Identificacao` descrevendo o estado do cliente. É criado no ambiente HALL PRINCIPAL e destruído no ambiente SAÍDA. Compõe-se dos seguintes objetos:

a) Carrinho de compras.
b) `UserId`**,** que é uma senha do cliente para cadastro e consulta de pedidos.
c) Uma lista dos ambientes percorridos pelo cliente durante a seção.

5.5.2 Servlets como representantes de clientes

Foi identificado dois tipos de servlets: o tipo *guia* e o tipo *funcionário*.

Os servlets guias têm como funcionalidade dirigir um usuário até um ambiente. Eles criam os *beans* necessários para que uma página JSP possa apresentar ao cliente uma visão inicial do ambiente.

Exemplo

Ao receber do cliente a solicitação "leve-me ao ambiente Pisos", o servlet `CategCatalogo` , cf. Fig. 5.3, executa os seguintes passos:

1) Recebe a identificação do cliente (`Id`).
2) Solicita permissão de entrada no ambiente PISOS fornecendo a identificação recebida.

3) Se a entrada for permitida ele entra. Caso contrário, emite uma mensagem de erro ao cliente.
4) Utiliza o recurso `ListaItens` do ambiente que contém a lista de materiais disponíveis para venda.
5) Cria um *bean* com as características do ambiente (`titulo, saidas, saidasJSP, cabecalho e rodape`).
6) Delega à página JSP correspondente, no caso `VisaoDep.jsp`, a tarefa de apresentar ao cliente a visão correspondente do ambiente em que o cliente se encontra.

Os servlets funcionários são utilizados para executar ações dentro dos ambientes.

Exemplo

O servlet `EfetivaCompra` tem como tarefa providenciar a aprovação de crédito do cliente e efetivar a compra caso o pagamento tenha sido aprovado. A execução desta tarefa segue os seguintes passos:

1) Recebe a identidade do cliente.
2) Solicita a entrada no ambiente CRÉDITO. É importante ressaltar que se a entrada foi permitida, o cliente já está de posse do seu cadastro de clientes (`cliente`) e de seu pedido de compras (`pedido`), pois acabou de passar pelo ambiente DOCUMENTAÇÃO.
3) Utiliza o recurso `validaCredito(cliente,pedido)` solicitando a aprovação de crédito.
4) Se o crédito for reprovado, anexa o *bean* `Erro` que contém a mensagem de retorno do método ao objeto implícito `request` e solicita à página `CreditoReprovado.jsp` que retorne a visão ao cliente.

Caso o crédito tenha sido aprovado:

5) Solicita entrada no ambiente PEDIDOS;
6) Utiliza o recurso `gravaPedido(pedido);`

7) Volta para ambiente CRÉDITO;
8) Solicita entrada no ambiente FINANCEIRO;
9) Utiliza recurso `lançaContabil(pedido,cliente);`
10) Volta para ambiente CRÉDITO;
11) Cria *beans* com as informações de aprovação do pedido e informações do ambiente CRÉDITO e solicita à página `CreditoAprovado.jsp` que apresente a visão ao cliente.

Segue-se agora algumas observações a respeito da realização do protótipo.

a) Disponibilidade de ambientes

Os ambientes do sistema são disponibilizados aos servlets através de um objeto do tipo `Hashtable` cuja chave é o nome do ambiente. Ele está associado ao contexto da aplicação para que os ambientes possam ser utilizados simultaneamente por todos os servlets. A sincronização do acesso aos recursos deve ser explicitamente feita na definição do recurso em cada ambiente. Ao entrar em um ambiente, o servlet sempre adiciona o ambiente ao seu caminho.

b) Escolha das visões a serem requisitadas pelo servlet.

A visão de um ambiente depende do ambiente em que o cliente se encontra e do recurso utilizado. Por exemplo, admitindo que dois clientes estejam dentro do ambiente ABERTURAS, um pode estar vendo o ambiente como um todo (recurso `listaItens(nome)`) e outro pode estar vendo detalhes de um item (recurso `selecionaItem(item)`). No protótipo desenvolvido para o Sistema Casarão, cada servlet sabe qual visão requisitar em cada situação.

5.5.3 Páginas JSP como visões de um ambiente.

As páginas JSP são solicitadas pelo servlet que criou os *beans* necessários para que a página apresente a visão ao cliente. Estes *beans* podem conter dois tipos diferentes de informação, são elas:

a) Dados do ambiente: cabeçalho, rodapé, título, saídas e saídasJSP. O atributo `saidasJSP` contém o endereço dos servlets a serem invocados para cada saída oferecida.

b) Dados específicos da resposta, por exemplo, a lista de pedidos feitos por um determinado cliente, ou os itens que compõe o catálogo do ambiente PISOS.

A Fig. 5.8 ilustra a visão apresentada ao cliente pela página JSP `VisaoDep.jsp` da Fig. 5.3. Esta visão representa a lista de produtos disponíveis no ambiente PISOS e pode ser associada à interface gráfica especificada na Fig. 4.5 do Cap. 4. Conforme a especificação, o usuário tem duas possibilidades de navegação além de escolher um produto: conferir suas compras (CONFERIR COMPRAS) ou voltar ao índice do catálogo (ÍNDICE DO CATÁLOGO). Segundo o contexto de ambientes, estas possibilidades podem ser identificadas respectivamente com as saídas CARRINHO e HALL DA LOJA do ambiente PISOS.

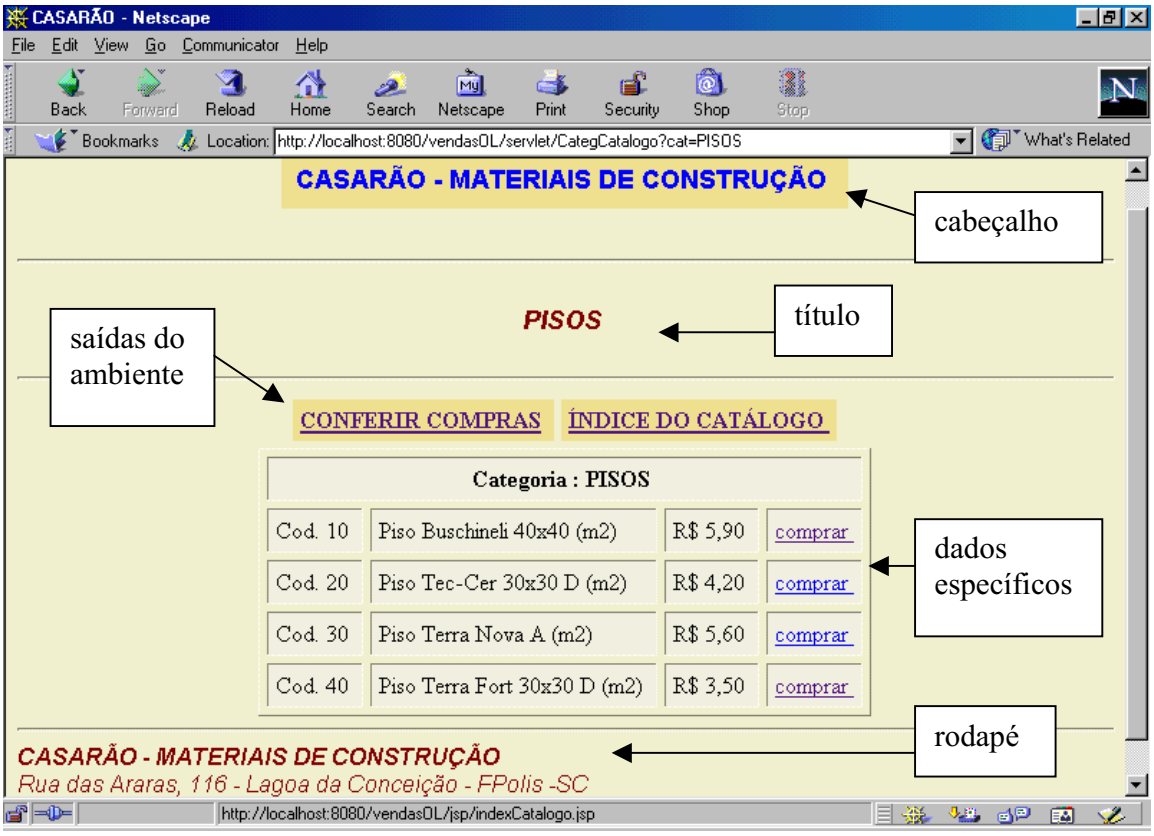

Fig. 5.8 - Visão apresentada ao usuário pela página `VisaoDep.jsp` da figura 5.3.

5.6 Análise do comportamento do sistema

A figura a seguir apresenta o diagrama de seqüência do comportamento do sistema ao receber uma solicitação do cliente. Cada solicitação é recebida por um servlet. Dependendo da solicitação, o servlet entra em um ambiente e utiliza os recursos nele disponíveis para criar os *beans*. Em seguida redireciona a solicitação para uma página JSP que utiliza os *beans* criados para apresentar a resposta ao cliente.

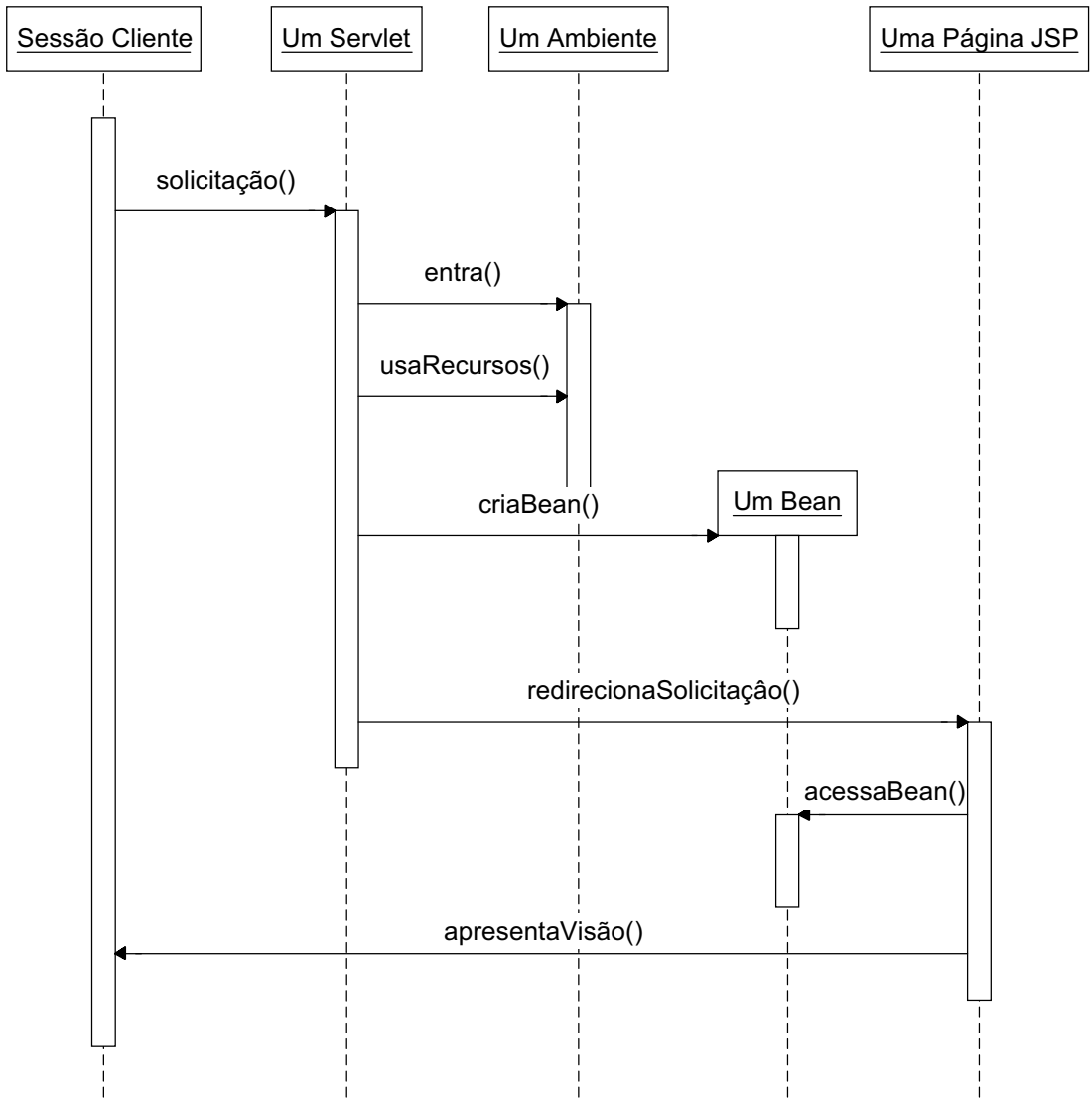

Fig. 5.9 - Diagrama de seqüência do comportamento do sistema a cada solicitação.

5.7 Conclusão

Este capítulo apresentou um projeto para o Sistema Casarão que utiliza uma arquitetura do tipo cliente/servidor de três camadas. Esta arquitetura utiliza os recursos das tecnologias Servlet e JSP e tem no conceito de ambientes o seu suporte intuitivo principal. Foi desenvolvido um protótipo baseado neste projeto.

A camada de apresentação é composta por Servlets e páginas JSP. A camada de lógica da aplicação é dividida em duas sub-camadas, onde a primeira engloba a definição dos ambientes e seus recursos e a segunda engloba a definição da forma pela qual os recursos dos ambientes são disponibilizados. Segundo esta proposta de arquitetura, um servlet pode ser visto como um agente de software que, ao receber uma solicitação, adquire a identificação do cliente, "visita" os ambientes do sistema e utiliza os recursos neles disponíveis para criar os *beans* que serão utilizados pela página JSP na apresentação da resposta à solicitação. Os recursos disponíveis em cada ambiente utilizam as classes da camada de acesso à base de dados para manipular informações persistentes.

O protótipo desenvolvido do Sistema Casarão apresentado neste capítulo contempla os requisitos funcionais do sistema (cf. Cap. 4) e exibe uma arquitetura adequada à natureza das aplicações de vendas *online*.

Capítulo 6

Conclusão e Perspectivas de Continuação

O estudo desenvolvido nesta dissertação abordou o tema comércio eletrônico sob uma perspectiva arquitetural. Em síntese, os Capítulos 2 e 3 revisaram ferramentas para o desenvolvimento de aplicações Web (anexo A), o Cap. 4 apresentou a análise de um estudo de caso, o Sistema de Vendas *Online* Casarão e o Cap. 5 propôs um projeto para esse sistema utilizando a tecnologia Servlets/JSP disponibilizada pela empresa americana Sun Microsystems.

O projeto apresentado no Cap. 5 segue uma arquitetura que pode ser vista como um refinamento da arquitetura em três camadas com servlets e *beans* fornecendo um modelo de interação entre as camadas de apresentação e de lógica da aplicação. A camada de apresentação segue o modelo padrão proposto pela tecnologia Servlets/JSP. Por outro lado, a camada de lógica de aplicação foi arquiteturalmente concebida seguindo um modelo baseado na noção de ambientes. A apresentação de um estudo de caso explorando essa como um conceito arquitetural para o desenvolvimento de sistemas de vendas *online* é a principal contribuição desse trabalho, uma vez que ainda não há conceitos arquiteturais bem definidos no caso dos sistemas de comércio eletrônico. Foi também desenvolvido um protótipo para uma avaliação prática dessa proposta. Os resultados obtidos revelaram-se bastante satisfatórios. A seguir são feitas algumas observações importantes sobre a realização deste trabalho.

a) A arquitetura proposta mostra-se adequada para sistemas cujo contexto pode ser naturalmente identificado com a noção de ambientes como é o caso dos sistemas de vendas *Online*. A noção de ambientes como conceito arquitetural em projetos de sistemas distribuídos tem sido explorada também em outros contextos de aplicação.

Por exemplo, ambientes móveis [CAR99, MAR02], e ambientes de interação [CAS99].

b) A arquitetura proposta facilita a fase de especificação do sistema. Geralmente, a fase de especificação fica a cargo de um grupo heterogêneo de profissionais, composto por especialistas em informática e no negócio a ser desenvolvido. A utilização de conceitos arquiteturais, no nosso caso, a noção de ambientes empregada na camada de negócio, facilita a comunicação entre estes dois tipos de especialistas, oferecendo uma maior oportunidade ao especialista do negócio de compreender e intervir de forma mais ativa na fase de especificação.

c) A fase de realização do projeto também é facilitada. Primeiro porque a adoção da tecnologia Servlets poupa o programador dos transtornos de lidar diretamente com as requisições HTTP e gerenciamento de sessão de usuário. Segundo porque as páginas JSP permitem uma separação do conteúdo dinâmico e do conteúdo estático das páginas Web, permitindo assim uma clara separação de tarefas entre programadores e Web Designers. Terceiro porque, definidos os ambientes, a programação dos servlets fica muito simplificada uma vez que é seguido sempre o mesmo modelo: receber uma identificação de cliente, entrar num ambiente, executar ações e solicitar a apresentação de uma visão a uma página JSP.

d) As restrições de entrada nos ambientes e na utilização de seus recursos impõe restrições de navegabilidade do sistema, evitando assim que o sistema entre em algum estado inconsistente.

e) Informações associadas aos ambientes ficam fáceis de serem extraídas. Por exemplo, por quanto tempo um ambiente não é visitado ou quais os itens que são selecionados, mas não são comprados.

f) Novas funcionalidades são facilmente incorporadas bastando definir em que ambientes elas se encontram disponíveis e quais as restrições de sua utilização. O

lugar e a forma como são geradas estas funcionalidades independem da sua utilização devido à divisão da camada de lógica da aplicação em duas sub-camadas independentes.

g) A abordagem de ambientes apresenta algumas desvantagens com relação à algumas ações que normalmente são familiares a usuários da Web. Por exemplo, alterar a quantidade de um item comprado é uma ação que deve ser executada no ambiente ao qual este produto pertence. Nos sistemas de vendas *online* existentes, um cartão de compras é geralmente associado à sessão do cliente e este tipo de alteração é feita bastando o cliente modificar a quantidade do pedido no próprio cartão. Outra desvantagem desta abordagem é que os caminhos possíveis são definidos conforme os caminhos em uma loja real. Desta forma, a navegação do cliente fica restrita a ambientes adjacentes.

Segue-se agora algumas propostas para a continuação desse trabalho.

a) Apesar de funcional, a noção de ambientes que foi desenvolvida no projeto da camada de lógica da aplicação apresentada no Cap. 6 seguindo um modelo de ambientes é ainda bastante incipiente. Por exemplo, o projeto não oferece nenhum mecanismo de composicionalidade (a criação de ambientes complexos a partir de ambientes mais simples) e tão pouco exibe autonomia computacional (gerência independente de *threads*). É preciso um maior aprofundamento nessa questão visando o desenvolvimento de um modelo de ambientes mais adequado aos aspectos habituais de qualidade de software.

b) Na arquitetura proposta, dois usuários que estejam compartilhando o mesmo ambiente não se vêem. Uma proposta de continuação desse trabalho seria fazer com que dois usuários possam se comunicar a partir de um mesmo ambiente proporcionando assim a possibilidade de uma negociação entre um cliente e um gerente do estabelecimento.

ANEXO 1 - A Web

Este anexo aborda alguns dos principais aspectos da arquitetura da rede Internet e de aplicações Web.

1. Introdução

As primeiras redes de computadores foram desenvolvidas em ambiente acadêmico e começaram a ser amplamente utilizadas por universidades, centros de pesquisa e algumas empresas no início dos anos 80. Através de uma rede, vários usuários puderam compartilhar os mesmos recursos implicando em considerável economia de custos. A partir dos anos 90, os computadores pessoais ficaram bastante acessíveis à população, o que provocou um crescimento rápido da demanda de acesso às redes de computadores. Com o passar do tempo, mais e mais redes foram desenvolvidas e interligadas, culminando num conjunto de redes interconectadas de abrangência mundial que conhecemos hoje por Internet.

2. A Arquitetura de Redes e o Modelo de Referência TCP/IP

Uma rede de computadores é um conjunto de computadores autônomos capazes de trocar informações através de um sistema de comunicação.

Através de uma rede de computadores, usuários que estejam fisicamente distantes podem compartilhar de arquivos, impressoras e outros recursos computacionais. Segundo a distância física entre seus nós, as redes são classificadas em três tipos: locais (LANs – *Local Area Network*), metropolitanas (MANs – *Metropolitan Area Network*) e geograficamente distribuídas (WANs – *Wide Area Network*) [SOA97]. As redes locais são redes privadas, geralmente utilizadas dentro de uma empresa, cuja abrangência restringe-se a um espaço físico de até 1 km de extensão. As redes metropolitanas, bastante utilizadas pelas empresas de televisão a cabo, podem abranger toda uma cidade. As redes

geograficamente distribuídas podem ter uma abrangência de um país ou continente. Para cada tipo de rede são utilizadas diferentes tecnologias de software e hardware. Por exemplo, em uma WAN existem computadores chamados de roteadores cuja única finalidade é redirecionar as mensagens enviadas através da rede. Duas redes, ainda que utilizem tecnologias diferentes, podem ser interligadas através de *gateways*. Um conjunto de duas ou mais redes interligadas, de qualquer abrangência, chama-se Inter-Rede [TAN97].

Para que haja troca de informações entre dois computadores é necessário que haja um acordo entre eles para estabelecer a forma como as informações serão enviadas e recebidas. O conjunto de regras e convenções que controla o formato e o significado das informações trocadas entre dois computadores quaisquer de uma rede é chamado de protocolo [TAN97]. Existem dois principais modelos de protocolos de comunicação: o Modelo de Referência OSI e o Modelo de Referência TCP/IP.

A Internet é uma Inter-Rede de abrangência mundial onde o software de comunicação de rede utiliza o Modelo de Referência TCP/IP.

2.1 O Modelo de Referência TCP/IP

Devido à complexidade envolvida na transmissão de informações entre computadores, a arquitetura dos protocolos de comunicação é dividida hierarquicamente em uma pilha de camadas de tal forma que uma camada inferior colabora com a superior através de interfaces de serviços.

Os serviços oferecidos pelos protocolos são classificados em dois tipos [SOA96]:

a) Serviços orientados à conexão

   Exigem que uma conexão seja estabelecida entre o emissor da mensagem e o receptor, analogamente a uma ligação telefônica. Neste caso, as mensagens enviadas serão recebidas pelo receptor na mesma ordem em que foram enviadas.

b) Serviços sem conexão

Ao contrário, não garantem a ordem de chegada das mensagens. Estes enviam pela rede pacotes de informações que contém, entre outras, o endereço do destinatário.

Cada serviço pode ainda ser caracterizado pela sua qualidade. Quando a entrega de uma mensagem é garantida pelo serviço, o serviço é caracterizado como sendo *confiável*. Caso contrário, é *não confiável*.

A figura a seguir ilustra as camadas definidas no Modelo de Referência TCP/IP e alguns de seus protocolos.

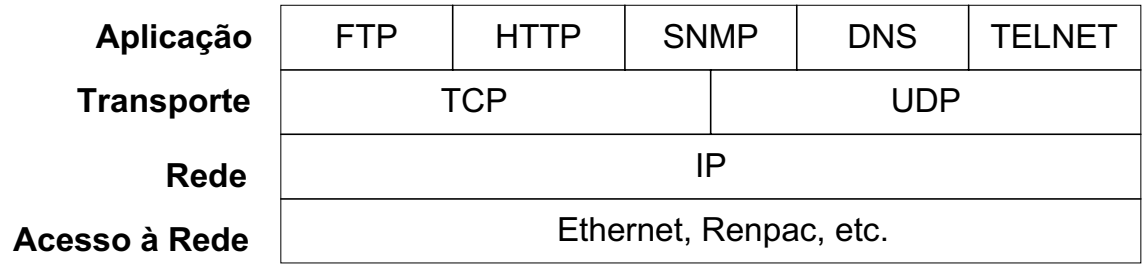

Fig.1 – Camadas e protocolos no Modelo de Referência TCP/IP.

A camada de Acesso à Rede é responsável pela transmissão física dos dados entre dois computadores quaisquer da rede.

A camada de Rede é responsável por receber e enviar os pacotes que circulam na rede. O protocolo IP (*Internet Protocol*) define a unidade de transmissão de dados conhecida pelo nome de datagrama. Ele é responsável pelo endereçamento, fragmentação e remontagem dos datagramas enviados e recebidos. Todas as camadas superiores da rede utilizam seu serviço, que é sem conexão e não confiável.

A camada de Transporte permite a troca de informações entre as camadas correspondentes do emissor e receptor final. Nesta camada são utilizados dois protocolos importantes : o TCP (*Transmission Control Protocol*) e o UDP (*User Datagram Protocol*). A principal diferença entre eles é que o TCP oferece um serviço *orientado à conexão confiável* e cuida também do controle de fluxo ao passo que o UDP oferece um serviço *sem conexão não confiável* para aplicações que não necessitam nem de controle de fluxo, nem da manutenção da seqüência de mensagens enviadas. Em contrapartida, o serviço oferecido pelo UDP é mais rápido que o oferecido pelo TCP. Sendo assim, as aplicações podem escolher o serviço a ser utilizado conforme suas necessidades.

Os protocolos da camada de transporte precisam conhecer o endereço do computador destino e a aplicação para a qual a mensagem está sendo enviada. O primeiro é identificado através do endereço IP e a aplicação é identificada através do número da porta do servidor que foi alocada para ela. Caso esteja utilizando o TCP, a aplicação no servidor conecta-se a uma porta através da qual vai esperar uma solicitação de algum cliente. Caso esteja utilizando o UDP, os datagramas serão entregues para o número da porta especificado no próprio datagrama.

A camada de Aplicação oferece serviços que utilizam os serviços das camadas inferiores para transmitir dados através da rede. Por exemplo, o protocolo FTP é utilizado para transferência de arquivos e utiliza o protocolo TCP da camada de transporte, enquanto o protocolo SNMP é utilizado pelas aplicações de correio eletrônico e utiliza o protocolo UDP da camada de transporte. O protocolo HTTP é o protocolo padrão utilizado pela Web.

3. World Wide Web

A World Wide Web (WWW ou apenas Web) é um serviço de rede que facilita o acesso de milhares de pessoas à Internet, permitindo que elas partilhem vários tipos de recursos. Teve sua origem em 1989 no CERN (Centro Europeu para Pesquisa Nuclear) com o objetivo de fornecer um meio através do qual cientistas de vários países pudessem trocar

informações sobre suas pesquisas [TAN97]. Primeiramente, foi desenvolvido e utilizado apenas em modo texto. Em 1993 surgiu a primeira versão com interface gráfica, o Mosaic. Nesta época, os computadores pessoais já estavam sendo bastante popularizados. Simultaneamente, surgiam novas tecnologias de transmissão na área de Redes de Computadores, por exemplo, a fibra óptica, que aumentou a eficiência do transporte de grande quantidade de informações.

O software WWW tornou-se então o principal software utilizado pela Internet por apresentar uma interface amigável com o usuário e oferecer um grande conjunto de funcionalidades. Dado a sua natureza multimídia, a Internet é hoje um meio de comunicação utilizado para diversos fins. Através das aplicações nela disponíveis, podemos ler jornais, fazer anúncios, trocar correspondências, fazer compras, etc.

Através da Web podemos acessar arquivos fisicamente espalhados pelo mundo. Na Web, estes arquivos são chamados de páginas. O *browser* é o software através do qual as páginas podem ser selecionadas e acessadas pelo usuário [TAN97]. Para que possam ser localizados na rede, cada página ou recurso acessível na Web possui uma referência única chamada URL (*Uniform Resource Locator*). O protocolo padrão utilizado para transmitir essas páginas através da rede é o HTTP. Para que possam ser interpretadas pelo *browser*, as páginas Web são construídas utilizando uma linguagem padrão chamada HTML (*Hypertext Markup Language*).

Esta seção apresenta primeiro uma arquitetura muito utilizada pelas aplicações Web denominada cliente/servidor. Em seguida, é feito um estudo mais detalhado do protocolo HTTP, das referências URL e da linguagem HTML.

3.1 Arquitetura cliente/servidor de três camadas

Grande parte das aplicações Web baseiam-se na arquitetura cliente/servidor que é composta por duas partes lógicas: o servidor que oferece determinados serviços e o cliente

que solicita estes serviços. Sendo assim, a comunicação entre cliente e servidor pode ser vista como uma troca de mensagens do tipo solicitação/resposta [TAN99].

A parte cliente da aplicação compreende um conjunto de interfaces que possibilita as seguintes ações ao usuário:

- selecionar um dos serviços disponíveis no servidor;
- solicitar a execução deste serviço ao servidor, eventualmente enviando dados;
- receber a resposta referente à solicitação.

A parte servidor da aplicação é composta basicamente por um conjunto de processos responsáveis pelas seguintes ações do servidor:

- executar os processamentos necessários para responder a cada solicitação;
- enviar a resposta ao cliente referente à solicitação feita;
- administrar solicitações concorrentes.

Um servidor pode atender vários clientes simultaneamente. Em especial, as aplicações cliente/servidor desenvolvidas em Java criam um servidor *multithreading* que pode gerenciar várias conexões com vários clientes concorrentemente [LEA00].

É muito comum que, para responder uma solicitação, o servidor tenha que acessar informações de um banco de dados compartilhado por várias aplicações. Muitas vezes, o próprio banco de dados incorpora as funcionalidades necessárias para atuar como um servidor de informações [OZS99]. Neste caso, a comunicação entre a aplicação servidor e o banco de dados pode ser feita também utilizando um modelo solicitação/resposta através da Internet.

As aplicações cliente/servidor com as características acima descritas podem ser consideradas aplicações distribuídas de três camadas *(three-tier system*) [DEI01] que compreende:

a) A camada de apresentação, realizando a interface servidor/cliente;
b) A camada de lógica de processamento, controlando e executando os processos necessários para que as solicitações sejam respondidas;
c) A camada de acesso à base de dados, fornecendo a interface entre o servidor e o banco de dados para atualizar e recuperar informações.

A camada de apresentação utiliza os serviços da camada de lógica de processamento que, por sua vez, utiliza os serviços da camada de acesso a base de dados. A Fig. 2 apresenta o esquema geral de uma arquitetura de três camadas de uma aplicação distribuída sobre a Internet.

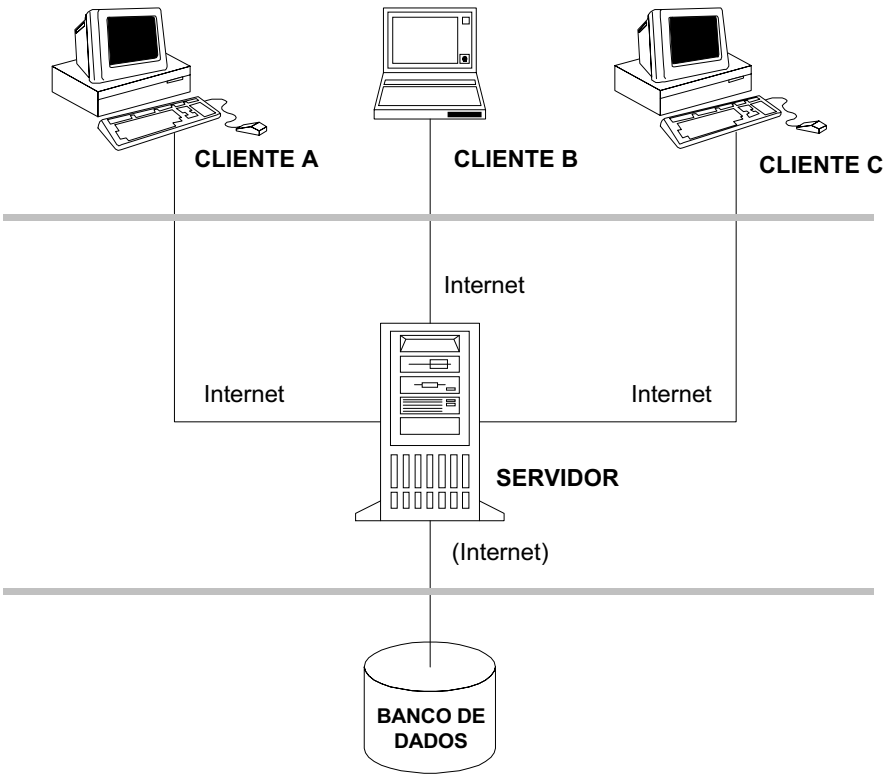

Fig. 2 – Arquitetura cliente/servidor de três camadas.

3.2 O protocolo HTTP (Hypertext Transfer Protocol)

Para que o *browser* da máquina cliente se comunique com um servidor, é preciso estabelecer uma conexão TCP com o servidor. O servidor atende solicitações através da porta 80 TCP (por *default*). Uma vez estabelecida a conexão, cliente e servidor trocam mensagens e por fim a conexão é fechada [TAN97].

O HTTP é o protocolo padrão utilizado na troca de mensagens entre cliente e servidor na Web. As solicitações de serviços feitas ao HTTP devem ter na primeira linha o nome do método que será executado na página Web.

Os métodos oferecidos pelo protocolo HTTP são os seguintes[TAN97]:

a) `GET`

Solicita uma página Web ao servidor. Se o cabeçalho *If-Modified-Since* for enviado, o servidor somente retornará a página caso esta tenha sido modificada desde a data enviada.

b) `HEAD`

Solicita o cabeçalho da página. É utilizado, por exemplo, para verificar a data de alteração da página ou se uma URL é válida.

c) `PUT`

Solicita ao servidor que grave uma página que está contida no corpo da solicitação. Esta solicitação pode conter informações sobre o tipo de informação que existe no corpo da página (*Content-type*) e pode ainda ter cabeçalhos de autenticação.

d) `POST`

Solicita ao servidor que grave uma página acrescida de dados novos. Esta solicitação é muito utilizada para enviar dados inseridos por usuários através de um formulário HTML.

e) `DELETE`

Solicita que o servidor remova uma página. Esta solicitação também pode conter cabeçalhos de autorização.

f) `LINK` e `UNLINK`

Solicita ao servidor que faça uma conexão/desconexão com outra página ou recurso.

As respostas enviadas pelo servidor utilizam a primeira linha como *linha de status*. Esta linha contém um código que corresponde a uma determinada mensagem convencionada pelo protocolo. Por exemplo, o código 304 informa que não houve alteração da página desde a data enviada pela solicitação do cliente.

3.3 URL (Uniform Resource Locator)

Todo recurso disponível na Web pode ser identificado unicamente através de uma URL e contém três partes: a primeira identifica o protocolo a ser utilizado, a segunda especifica o nome da máquina que hospeda a página e a terceira identifica o nome do arquivo que contém o recurso ou um atalho para ele. Por exemplo,

`http://www.guia.com.br/index.html`

- `HTTP` é o protocolo;
- `www.guia.com.br` é o nome máquina hospedeira;
- `index.html` é o nome do recurso que no caso é uma página HTML.

Quando um usuário solicita uma página a um *browser* através do protocolo HTTP, a seguinte seqüência de ações é executada pelo *browser* [TAN97]:

1) Faz uma análise da URL indentificando o protocolo, o nome da máquina e o recurso;
2) Solicita o número de IP da máquina a um servidor de nomes;
3) Recebe o endereço IP correspondente ao nome da máquina;
4) Solicita uma conexão através do protocolo TCP para o IP recebido;

5) Recebe como resposta um OK se a conexão foi estabelecida;
6) Envia uma solicitação do recurso utilizando o protocolo especificado.

3.4 HTML (Hypertext Markup Language)

HTML é a linguagem padrão utilizada para criar páginas Web. Através dela é possível descrever como as informações contidas em um documento devem ser apresentadas por um *browser*. Ela segue os padrões de formatação de texto SGML (*Standard Generalized Markup Language* – ISO 8879) [HTM02].

Um documento HTML é um arquivo texto composto por vários elementos. Um elemento é um componente básico da estrutura de um documento texto e pode ser composto por um texto ou por outro elemento ou ainda por ambos. Os elementos podem ser tabelas, formulários, cabeçalhos, listas, etc.

Cada elemento é identificado através de um par de marcas da forma `<instrução>` e `</instrução>` que denotam respectivamente o início e o final do elemento [TRU00]. Algumas instruções podem requerer parâmetros que são especificados como atributos da marca inicial. Existem também instruções que não necessitam da marca final. A tabela a seguir contém algumas instruções de formatação bastante utilizadas em documentos HTML [HTM02].

Instrução	**Descrição**
`<HTML> </HTML>`	Indica início e fim de um documento HTML
`<A> </A>`	Indica uma âncora de documento com o objetivo de criar *links* para outros recursos.
`<B> </B>`	Formata texto em negrito
`<I> </I>`	Formata texto em itálico
`<TABLE> </TABLE>`	Marca o início e fim de uma tabela
`<TR> </TR>`	Marca o início e fim de uma linha de uma tabela
`<TD> </TD>`	Marca o início e fim de uma célula de uma tabela

Instrução	Descrição
<FONT> </FONT>	Formata o texto de acordo com a fonte, tamanho e cor especificados
 	Muda de linha

Tabela 1 – Instruções HTML de formatação de conteúdo.

Além de especificar a formatação do texto a ser apresentado, a linguagem HTML oferece recursos para a construção de formulários interativos, através dos quais o usuário pode fornecer dados ao servidor. A tabela abaixo contém algumas instruções que definem elementos interativos [HTM02].

Instrução	Descrição
<INPUT TYPE=TEXT>	Caixa de texto
<INPUT TYPE=RADIO>	Botões de escolha exclusiva
<TEXTAREA> </TEXTAREA>	Caixa de texto com várias linhas
<INPUT TYPE=SUBMIT>	Envia os campos de um formulário para o recurso definido no parâmetro ACTION
<INPUT TYPE=RESET>	Restaura os valores *default* nos campos

Tabela 2 – Instruções HTML que permitem interação com o usuário.

Um formulário é um elemento delimitado pelas marcas <FORM> e </FORM>. A marca <FORM> pode ter os seguintes parâmetros:

a) ACTION

   Define qual a URL do recurso que receberá os dados do formulário.

b) METHOD

   Define qual método será utilizado ao enviar a solicitação HTTP. No caso do método GET, os dados são acoplados ao nome da URL que receberá a solicitação. No caso do método POST, os dados são concatenados em um *String* e enviados no corpo da solicitação.

Os dados de um formulário são enviados para um programa através da instrução SUBMIT. Quando o formulário é submetido, o *browser* envia uma solicitação HTTP com o método especificado no parâmetro METHOD para o recurso definido no parâmetro ACTION.

O modelo CGI

O recurso especificado no parâmetro ACTION de um formulário é geralmente um programa que implementa a interface CGI (*Common Gateway Inteface*) [NUG98]. Esta interface possibilita a troca de informações entre o browser do cliente e a aplicação servidor. Um programa CGI é via de regra armazenado no servidor em um diretório chamado *cgi-bin*. Este programa interpreta os dados enviados, processa as informações necessárias e geralmente devolve ao browser uma página HTML como resposta. Grande parte dos programas CGI fazem algum tipo de acesso a uma base de dados.

Exemplo

A seguir é apresentado o código de uma página HTML (Fig. 3) que exemplifica a utilização das instruções descritas nas tabelas acima. A Fig. 4 ilustra a apresentação desta página pelo *browser*.

```
<! Início do documento>
 <HTML>
<! Definição do cabeçalho>
<HEAD>
<! Definicao do titulo>
<TITLE> CASARÃO </TITLE>
<! Definição do título da página com fonte tipo ARIAL e
tamanho 4 na cor azul, no estilo itálico (I), negrito (B) e
centralizado>
<BR> <FONT FACE="ARIAL" SIZE="4" COLOR=blue>
<B> <I> <P ALIGN=center>
     INFORMAÇÕES PARA PEDIDO DE COMPRAS  </B> </I> </P>
</FONT>
</HEAD>

<! Definição da cor do background>
<BODY BGCOLOR=lightgoldenrodyellow> <BR>
```

```
<! Definição de uma tabela sem bordas, com fundo de cor
khaki, com
espaçamento entre colunas e linhas de tamanho 5 e
centralizada> <TABLE BORDER=0 BGCOLOR=khaki CELLSPACING=5
CELLPADDING=5
     ALIGN='center'>
<! Definição do começo da linha>
<TR>
<!Definição da primeira célula com um link para uma página
HTML>
<TD><A HREF='../html/MostraCartao.html'><B>
    CONFERIR COMPRAS</B></A></TD>

<!Definição da segunda célula com um link para uma página
HTML>
<TD><A HREF='../html/catalogo.html'><B>
    ÍNDICE DO CATÁLOGO</B></A></TD>

<! Definição da terceira célula com um link para uma página
HTML>
<TD> <A HREF='../html/index.html'> <B> HOME </B> </A> </TD>
 </TR>
</TABLE>

<! Definição de um texto simples com fonte do tipo ARIAL e
tamanho=3, na cor darkred e negrito >
<BR> <FONT FACE="ARIAL" SIZE="3" COLOR=darkred> <B>
  Por favor, preencha as seguintes informações: </B> <BR>

<! Definição de um formulário que solicita os seguintes
dados:
   a) Nome, utilizando uma caixa de texto;
   b) Endereço, utilizando uma área de texto;
   c) Tipo de cartão, utilizando botões de seleção exclusiva;
   d) Número do cartão, utilizando caixa de texto.
   Este formulário, ao ser submetido, envia os dados para o
   programa CGI RecebeDados utilizando o método POST.>

<FORM  METHOD="POST" ACTION="../cgi-bin/RecebeDados">
<BR> Nome :
       <INPUT TYPE=text NAME=nome SIZE=30 MAXLENGTH=50> <BR>
<BR> Endereço para entrega : <BR>
       <TEXTAREA NAME=endEntrega ROWS=1 COLS=30> </TEXTAREA>
<BR>
<BR> Cartão de Crédito     :
```

```
<INPUT TYPE=radio NAME=tipoCartao VALUE="MASTERCARD">
MASTERCARD
<INPUT TYPE=radio NAME=tipoCartao VALUE="VISA"> VISA
<BR> Número do Cartão      :
         <INPUT TYPE=text NAME=numCartao SIZE=19
MAXLENGTH=19>
<BR> <BR> </FONT>

<! Definição dos botões SUBMIT e RESET do formulário>
<INPUT TYPE=submit VALUE="ENVIAR">
<INPUT TYPE=reset VALUE="LIMPAR">

<! Indicação de final do formulário, do corpo da página e do
documento>
</FORM>
</BODY>
</HTML>
```

Fig. 3 – Código HTML de uma página Web chamada `exemplo.html`.

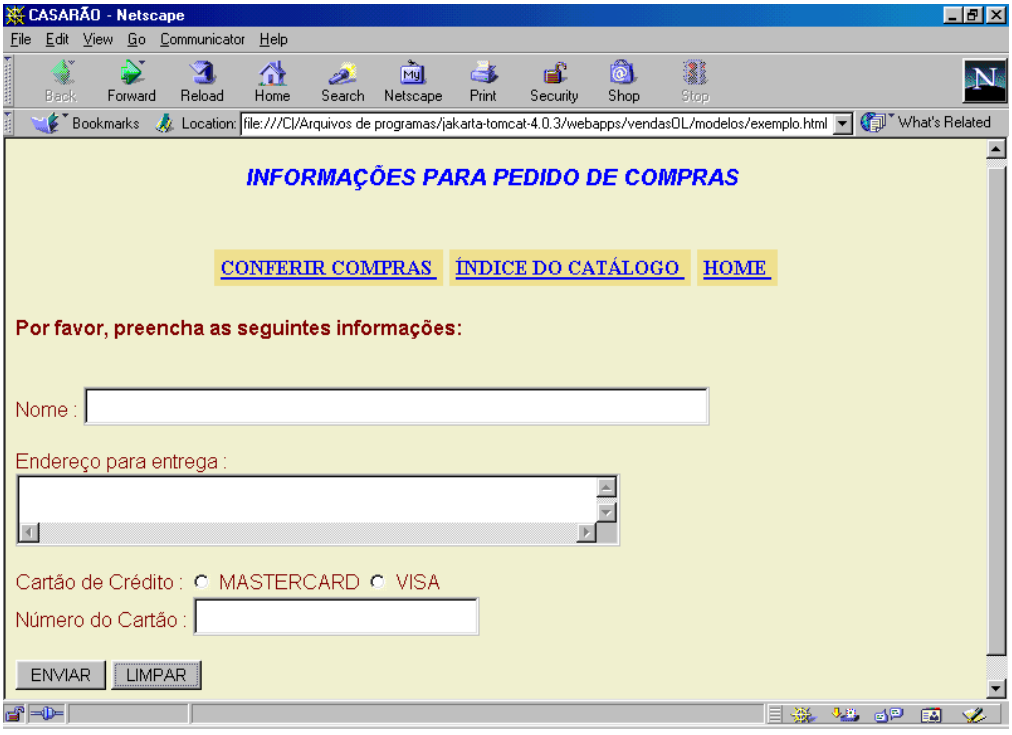

Fig. 4 – Apresentação da página `exemplo.html` pelo *browser*.

ANEXO 2 – Códigos Fonte

Este anexo apresenta o código fonte na liguagem Java de algumas classes referenciadas no texto.

1. Código fonte da classe RecebeDados.java

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class RecebeDados extends HttpServlet {
   private DepCredito credito; // objeto Departamento de Crédito

    public void init(ServletConfig config)
     throws ServletException
   {
   // cria objeto credito uma única vez
   credito = new DepCredito();
   }

   public void destroy()
   {
   // destroi objeto credito
   credito = null;
   }

   public void doPost( HttpServletRequest request,
                       HttpServletResponse response )
      throws ServletException, IOException
   {

      // Recupera as informações inseridas na página exemplo.HTML

      String _nome = request.getParameter("nome");
      String _endEntrega = request.getParameter("endEntrega");
      String _tipoCartao = request.getParameter("tipoCartao");
      String _numCartao = request.getParameter("numCartao");

      // Cria um objeto CartaoCliente com as informações recebidas
      CartaoCliente cartao = new CartaoCliente(_nome,_endEntrega,
                                _tipoCartao,_numCartao);

      // requisita a autorização do cartao ao objeto crédito
      credito.valida(cartao);
      // Abre um canal de comunicação
```

```
      PrintWriter canalSaida;
      response.setContentType ("text/html");
      canalSaida  = response.getWriter();

      // Envia uma página personalizada ao cliente informando
      // a validade do cartão.

      StringBuffer buf = new StringBuffer();
      buf.append( " <HTML> ");
      buf.append( " <BODY BGCOLOR=lightgoldenrodyellow>");
      buf.append( " <FONT FACE='ARIAL' SIZE='5' COLOR=darkred>");
      buf.append( " <BR> <BR> <BR> Sr(a). ");
      buf.append(  cartao.getNome() + ", <BR> <BR> ");

      if (cartao.getEstado())
         buf.append( " seu cartão foi aprovado. ");
      else
         buf.append( " seu cartão está fora de validade. ");
      buf.append( " </BODY> </HTML> ");
      canalSaida.println(buf.toString());
      canalSaida.close();

   }// doPost
}// RecebeDados
```

2. Código fonte da classe CartaoCliente.java

```
public class CartaoCliente {
  private String nome = "";
  private String endEntrega = "";
  private String tipoCartao = "";
  private String numCartao = "";
  private boolean estado = false;

public CartaoCliente () {
  nome = "";
  endEntrega = "";
  tipoCartao = "";
  numCartao = "";
  estado = false;
}
public CartaoCliente(String _nome, String _endEntrega,
                     String _tipoCartao, String _numCartao)
{
  nome = _nome;
  endEntrega = _endEntrega;
  tipoCartao = _tipoCartao;
  numCartao = _numCartao;
  estado = false;
```

```
}

public String getNome() { return nome; }
public String getEndEntrega() { return endEntrega; }
public String getTipoCartao() { return tipoCartao; }
public String getNumCartao() { return numCartao; }
public boolean getEstado() { return estado; }

public void setNome(String _nome){ nome=_nome; }
public void setEndEntrega(String _endEntrega)
      { endEntrega=_endEntrega; }
public void setTipoCartao(String _tipoCartao)
       { tipoCartao=_tipoCartao; }
public void setNumCartao(String _numCartao)
     { numCartao=_numCartao; }
public void setEstado(boolean _estado)
     { estado=_estado; }
} // class CartaoCliente
```

3. Código fonte da classe DepCredito.java

```
// classe do objeto que aprova crédito
import java.util.Random;

public class DepCredito{
 private String nome;

public DepCredito () {
  nome = "Departamento de crédito";
}

// valida crédito segundo número aleatório
public void valida(CartaoCliente cartao)
{
Random num_rand = new Random();
int numero = num_rand.nextInt(10);

if (numero > 5)
  cartao.setEstado(true);
else
  cartao.setEstado(false);
}
} // class DepCredito
```

4. Código fonte da página RecebeDados.jsp

```
<%-- importa as classes a serem utilizadas --%>
<%@ page import = "CartaoCliente, DepCredito" %>

<%-- cria o bean cartao no escopo da solicitação --%>
<jsp:useBean id="cartao" class="CartaoCliente" scope="request" />

<%-- recebe os dados do form via introspecção --%>
<jsp:setProperty name="cartao" property="*" />

<%-- utiliza o objeto credito do escopo da aplicação --%>
<jsp:useBean id="credito" class="DepCredito" scope="application"
/>

<%-- invoca o objeto credito para validar o cartao --%>
<% credito.valida(cartao); %>

<%-- redireciona a solicitação para a página Responde.jsp --%>
<jsp:forward page="Responde.jsp" />
```

5. Código fonte da página Responde.jsp

```
<%-- importa a classe do bean CartaoCliente --%>
<%@ page import = "CartaoCliente" %>

<%-- utiliza o bean cartao criado pela pagina RecebeDados --%>
<jsp:useBean id="cartao" class="CartaoCliente" scope="request"/>

<%-- cria resposta HTML --%>
<HTML>
<BODY BGCOLOR=lightgoldenrodyellow>
<FONT FACE='ARIAL' SIZE='5' COLOR=darkred>
<BR> <BR> <BR>
<%-- recupera nome no bean --%>
 Sr(a). <%= cartao.getNome() %>
<BR> <BR>

<%-- envia a resposta conforme a propriedade estado do bean --%>
<%  if (cartao.getEstado()) { %>
    seu cartão foi aprovado.
<% } else { %>
    seu cartao está fora de validade.
<% } %>

</FONT>
</BODY>
</HTML>
```

Referências Bibliográficas

[ANG00] ANGELOV C., August J., *Designing a Flexible Services Based Architecture for Internet Applications*, OOPSLA, 2000.

[ASP02] Empresa Microsoft, *Active Server Pages*. Endereço:

http://msdn.microsoft.com/asp.

[BEA02] Sun Microsystems, *JavaBeans Technology Home Page*. Endereço:

http://java.sun.com/beans.

[BOO94] BOOCH G. *Object oriented analysis and design with applications*, Addison-Wesley, 1994.

[BRA99] BRAGA R., Germano F., Masiero P., *A Pattern Language for Business Resource Management*, Pattern Languages of Program Conference (PloP), 1999.

[CAR99] CARDELLI L., *Abstractions for Mobilde Computation*, 1999. Endereço:

http://www.research.digital.com/SRC/personal/Lucca_Cardelli/.

[CAS99] CASTRO C. , *Uma proposta baseada em padrões de design para o desenvolvimento de sistemas cooperativos em ambiente aberto*, manuscrito de dissertação de mestrado, UFSC, 1999.

[DAS02] DASTANI M., Jacobs N., Jonker C., Treur J., *Modelling User Preferences and Mediating Agents in Eletronic Commerce*, 2002.

[DAV00] DAVIS T., Walker C., *Take Control of the Servlet Environment: Invisibly extend the functionality of the servlet API*, 2000. Endereço:

http://developer.java.sun.com/developer/technicalArticles/Servlets/ServletControl/.

[DEI01] DEITEL H.M. *Java: como programar*. Terceira Edição. Bookman, 2001.

[ECK00] ECKEL B., *Thinking in Java*, 2000. Endereço:

http://www.BruceEckel.com.

[FOW00] FOWLER M., *UML Essential : um breve guia para a linguagem padrão de modelagem de objetos*. Segunda Edição. Bookman, 2000.

[FOW97] FOWLER M., Scott K*, UML Distilled*, Addison-Wesley, 1997.

[GAM94] GAMMA E., Helm R., Johnson R., Vlissides J., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1994.

[GAR01] GARLAN D., *Software Architecture*, "Encyclopedia of Software Engineering", John Wiley & Sons, 2001.

[GAR94] GARLAN D., SHAW M., *An Introduction to Software Architecture*, 1994, CMU-Software Engineering Institute Technical Report, 1994.

[HOA74] HOARE C., "*Monitors: an operating system structuring concept*", Communication of the ACM, vol. 17, num. 10, oct 1974.

[HTM02] *Html Specifications*. Endereço: http://www.w3.org/MarkUp/html-spec/.

[HUN00] HUNTER J., *The problems with JSP*, 2000. Endereço:

http://www.servlets.com/soapbox/problems-jsp.html.

[JAV02] *JAVA Tutorial*, http://java.sun.com, 2002.

[JES00] JESUS J. , *Let´s Brew Some Java*, 2000.

[JSP01] *JavaServer Pages Specification v 1.2*, 2001. Endereço:

http://java.sun.com/products/jsp.

[LAR00] LARMANN C., *Utilizando UML e Padrões: uma introdução à análise e ao projeto orientados a objetos*. Bookman, 2000.

[LAV02] LAVANDOWSKA L., *JSP Architectures: an explanation and comparison of the methodologies commonly known as "Model I" and "Model II"*. Endereço:

http://www.brainopolis.com/jsp/book/jspBook_Architectures.html.

[LEA00] LEA D., *Concurrent Programing in Java*. Second Edition. Addison-Wesley, 2000.

[LEW00] LEWIS J., Loftus W., *Java software solutions: foundations of program design*. Second Edition. Addison-Wesley, 2000.

[LUB98] LUBLING O., Malave L., *Developing Scalable, Reliable, Business Applications with Servlets*, 1998. Endereço:

http://developer.java.sun.com/developer/technicalArticles/Servlets/Razor/index.html.

[MAR02] MARQUES C., *Um Sistema de Bibliotecas Utilizando Agentes Móveis*, manuscrito de dissertação de mestrado, USFC, 2002.

[MED99] MEDVIDOVIC N., Rosenblum D., *Assessing the suitability of a standard design method for modeling software architectures*, Proceedings of the First Working IFIP Conference on Software Architecture (WICSAI), 1999.

[NUG98] NUGENT J., *Introduction to CGI*, http://tech.irt.org/articles/js073, 1998.

[OZS99] OZSU M.T., Valduriez P., *Principles of Distributed Database Systems*. Prentice Hall, 1999.

[PET02] PETERSEN S., Rao J., Tveit A., *Virtual Enterprises: Challenges in Selecting and Integrating Computational and Human Resources,* 2002.

[PHP02] The PHP Group. Endereço: http://www.php.net.

[SCH96] SCHNEIDER G., Winters J.P., *Applaying Use Cases: a practical guide*, Addison-Wesley, 1999.

[SER01] *Java Servlet Specification v 2.3*, 2001. Endereço:

http://java.sun.com/products/servlets.

[SES00] SESHADRI G., *Advanced form processing using JSP*, 2000. Endereço: http://www.javaworld.com/javaworld/jw-03-2000/jw-0331-ssj-forms_p.html.

[SES01] SESHADRI G., *JavaServer Pages Fundamentals*, 2000. Endereço:

http://developer.java.sun.com/developer/onlineTraining/JSPIntro/contents.html.

[SES99] SESHADRI G., *Understanding JavaServer Pages Model 2 Architecture*, 1999.

Endereço: http://www.javaworld.com/javaworld/jw-12-1999/jw-12-ssj-jspmvc_p.html.

[SOA97] SOARES L., Lemos G., Colcher S., *Redes de Computadores: Das LAN´s, MANs e WANs às Redes ATM*, 2ª Edição. Ed. Campus, Rio de Janeiro, 1997.

[SUN02] Empresa Sun Microsystems. Endereço: http://www.sun.com.

[TAN97] TANENBAUM A. S., *Redes de Computadores***.** Tradução da Terceira Edição**.** Editora Campus, 1997.

[TAN99] TANENBAUM A. S., *Sistemas Operacionais Modernos***.** LTC-Livros Técnicos e Científicos Editora S.A.**,** 1999.

[TRU00] NSCA, *A Beginner´s Guide to HTML*, 1997. Endereço: http://www.ncsa.uiuc.edu/General/Internet/WWW/HTMLPrimer.html.

[WES98] WESTLAND J. C., Au G., *A comparison of shopping experiences across three competing digital retailing interfaces*, International Journal of Eletronic Commerce 2(2): 57-69, 1998.

[WOO00] WOOLDRIDGE M., Jennings N., Zambonelli F., Omicini A., *Agent-Oriented software Engineering for Internet Applications*, publicado como Cap. 13 do livro Coordination of Internet Agents: Models,Technologies and Applications, Springer, 2000.

[WOO95] WOOLDRIDGE M., Jennings N., *Intelligent agents: Theory and practice*. The Knowlodge Engineering Review, 10(2):115-152, 1995.

[ZWA98] ZWASS V., *Structure and Macro Level impacts of Eletronic Commerce: from technological infrastructure to eletronic marketplaces,* "Emerging Information Technologies", Thousand Oaks-CA, 1998.