UNIVERSIDADE FEDERAL DE SANTA CATARINA – UFSC
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA – INE
CURSO DE GRADUAÇÃO EM CIÊNCIAS DA COMPUTAÇÃO – CCO

Aplicativo Cliente-Servidor multicamadas para controle de uma rede de lojas via WEB utilizando Java

Autor:

Henrique Eduardo Machado de Oliveira

Orientador:

Prof. Dr. Leandro José Komosinski

Membros da Banca:

Profª. M. Maria Marta Leite

Prof. Dr. Vitório Bruno Mazzola

Prof. Alcides José Fernandes Andujar, M.Eng

Florianópolis, de Janeiro de 2003

Sumário

Lista de figuras

Resumo

Este trabalho apresenta o modelo de desenvolvimento Cliente-Servidor multicamadas para aplicações na Internet utilizando a tecnologia Java. Este estudo complementa a formação acadêmica do Curso de Ciência da Computação agregando um novo modelo de implementação de sistemas às já conhecidas.

Esta abordagem requer um estudo profundo da Linguagem Java e de todas as ferramentas necessárias à modelagem e desenvolvimento de sistemas.

Propõe-se uma aplicação utilizando o modelo Cliente-Servidor para a automatização de uma empresa de porte médio, com algumas filiais. A aplicação visa interligar a empresa e aperfeiçoar os processos de controle de estoque e cadastro de clientes à distância via Internet.

Palavras chave: Automação Empresarial, Administração à distância, Java, Javabeans

Abstract

This work studies the multi layer Client-Server model for Internet applications using the Java Technology. This study complements the Computer Science academic knowledgement adding a new model of system implementation to the ones already known.

This approach requires a deep study of the Java Language and all of the tools necessary for system modeling and system development.

A Client-Server model application is suggested to automate a medium-size company and its branches. The application's goal is to connect the company internally and to improve some stock control process and client register.

Keywords: Enterprise Automation, Long Distance Management, Java, Javabeans

Capítulo 1

1. Introdução

1.1. Cenário

Por muitos anos, o processamento de dados se consistiu de transações relativamente simples oriundas de aplicações que visavam automatizar tarefas repetitivas de um determinado negócio. Programas de processamento em lote ou de simples entrada de dados que rodam sem problemas em computadores de grande porte com processamento centralizado, manipulavam com facilidade essas tarefas. Contudo, na década de 80, o uso dos computadores mudou drasticamente. A tecnologia dos microcomputadores, redes e comunicações evoluíram. As pessoas tiveram acesso aos computadores pessoais e se acostumaram às suas interfaces amistosas, aceitando com mais dificuldade as interfaces e o tempo de resposta oferecidos pelos sistemas tradicionais com acesso por terminais de vídeo.

Com o crescimento das organizações surgiu a necessidade dos usuários finais acessarem dados de múltiplas bases de dados em diferentes locais, resultando em aumento nos custos de comunicação e decréscimo no tempo de resposta. O processamento centralizado em computadores de grande porte não se adapta a essas novas necessidades.

Assim surgiram novos modelos de processamento baseados na arquitetura cliente/servidor.

O modelo de processamento cliente/servidor tornou-se rapidamente uma das abordagens mais utilizadas na indústria da computação. A evolução da informática possibilitou a várias pessoas compartilharem recursos distribuídos como informações, periféricos, serviços e aplicações. O compartilhamento de recursos quando executado corporativamente, através da interação entre clientes(requisitores de recursos) e servidores(provedores de recursos), pode ser alcançado utilizando-se o modelo de processamento cliente/servidor.

1.2. Motivação

O modelo de aplicação Cliente-Servidor se encaixa perfeitamente em redes de computadores, onde um Servidor é responsável em prover serviços para outras máquinas da rede. Extrapolando este sistema para a Internet, o que temos é uma enorme rede mundial, interligando cidades e países, com grande potencial de desenvolvimento de serviços que podem ser utilizados por diversas máquinas em qualquer ponto do globo.

No âmbito empresarial, a corrida pela Internet no Brasil vem crescendo de forma rápida. As empresas estão se atualizando para acompanhar as mudanças e participar deste fenômeno que é a Internet. O primeiro passo para as empresas foi simplesmente aparecer na Internet, geralmente na forma de um site[1], apenas como uma vitrine virtual. O segundo passo foi oferecer serviços a clientes através do site na Internet. A compra e venda de produtos, e o atendimento bancário via Internet são algumas das inúmeras facilidades oferecidas aos clientes nos dias de hoje. O terceiro passo é utilizar toda essa infra-estrutura chamada Internet, que conecta internamente as empresas e também as conecta com fornecedores e clientes, para automatizar e agilizar os processos de controle da empresa, transações entre filiais, e relacionamentos com fornecedores e clientes.

Dentro do panorama apresentado, surge a necessidade de estudar o modelo de aplicações Cliente-Servidor, pois o mesmo melhor se adapta à realidade do mercado hoje na Internet. Uma visão mais aprofundada do que é visto no Curso de Ciência da Computação sobre esta modelagem vem complementar a ampla formação já recebida durante todo o curso.

1.3. Objetivos

1.3.1. Objetivo Geral

O objetivo principal deste trabalho é o estudo da arquitetura Cliente-Servidor multicamadas para o desenvolvimento de sistemas para a Internet, visando complementar a formação acadêmica do Curso de Ciência da Computação.

[1] Conjunto de páginas acessíveis via navegador, hospedadas em um servidor na Internet.

1.3.2. Objetivos Específicos

- Estudar as características da arquitetura Cliente-Servidor multicamadas.

- Aprofundar o conhecimento na Linguagem Java como base para a implementação de sistemas Cliente-Servidor.

- Apresentar uma solução para automatização de uma empresa de médio porte, utilizando as técnicas estudadas.

1.4. Estrutura do trabalho

Para atender os objetivos propostos, este trabalho está estruturado da seguinte forma:

O Capítulo 1 apresenta as motivações e a justificativa para o estudo e desenvolvimento do trabalho.

No Capítulo 2 é feito uma introdução aos Sistemas Distribuídos, inicialmente caracteriza-se a computação distribuída, descrevendo-se os fatores que motivaram o seu desenvolvimento, classificando-se o hardware e software distribuído e são apresentadas suas principais características

No Capítulo 3 é apresentada a arquitetura Cliente/Servidor, e são detalhadas as arquiteturas em duas e três camadas, explicitando as suas vantagens, desvantagens e as principais diferenças entre elas.

No Capítulo 4 é apresentada a Linguagem Java, suas características e quais as biliotecas disponíveis para o auxílio ao desenvolvimento de aplicações cliente/servidor

No Capítulo 5 são apresentados os problemas de administração de uma empresa de médio porte na execução de seus processos sem uma ferramenta adequada de controle e gerência. O objetivo é mostrar em que ponto se encontra a informatização e quais as dificuldades enfrentadas pelas empresas na área de tecnologia. Também será proposta uma solução para a empresa baseada no modelo Cliente-Servidor

No Capítulo 6 é apresentada a análise de requisitos e modelagem do sistema utilizando o padrão UML.

O Capítulo 7 mostra o funcionamento da aplicação, implementada baseada na arquitetura cliente/servidor multicamadas.

Finalmente no Capítulo 8 serão traçadas as dificuldades encontradas, sugestões para trabalhos futuros e a conclusão final.

Capítulo 2

2. Sistemas Distribuídos

Este capítulo apresenta uma introdução aos sistemas distribuídos. Inicialmente caracteriza-se a computação distribuída, descrevendo-se os fatores que motivaram o seu desenvolvimento, classificando-se o hardware e software distribuído e são apresentadas suas principais características. É dado ênfase ao modelo de comunicação interprocessos, ressaltando-se seus aspectos de programação. Finalmente, comenta-se aspectos de sincronização de processos.

2.1. Introdução

A demanda crescente de processamento tem motivado a evolução dos computadores, viabilizando implementações de aplicações que envolvem um intenso processamento e grandes volumes de dados [1]. Na busca pelo alto desempenho o desafio é aumentar o desempenho do processador. Este aumento pode ser obtido através de [6]:

- Aumento da velocidade do relógio. Esta alternativa envolve o desenvolvimento de tecnologia de confecção de circuitos integrados, trazendo entretanto, problemas, tal como, o aumento de temperatura;

- Melhorias na arquitetura. Este objetivo motivou o surgimento dos processadores RISC, vetoriais e super-escalares;

- Melhoria no acesso à memória. Esta melhoria pode ser conseguida, por exemplo, através da exploração da hierarquia de memória, utilizando registradores, memória cache e memória principal;

- Utilizar vários processadores, distribuindo, entre estes, o processamento do programa.

O uso de computadores está se deslocando do uso de sistemas centralizados para o uso de sistemas com características distribuídas, por razões como: custo/performance e velocidade entre outras, que detalharemos a seguir.

Um sistema distribuído pode ser conceituado, segundo Tanenbaum [12], como qualquer sistema que possua múltiplas UCPs interconectadas trabalhando em conjunto. Couloris [1] conceitua de forma mais restrita, como uma coleção de computadores autônomos conectados através de uma rede e equipados com algum *software* de sistema distribuído. O *software* de sistema distribuído permite que os computadores coordenem suas atividades e compartilhem os recursos do sistema como o *hardware*, o *software* e os dados. É importante que o sistema distribuído seja projetado de maneira que, embora composto por computadores geograficamente distribuídos, seja percebido como um único sistema integrado.

Quando usada apropriadamente, a computação distribuída pode oferecer ganhos significantes em relação aos sistemas centralizados, podemos citar como vantagens [1]:

- Performance através do processamento paralelo;
- Confiabilidade e acessibilidade através da replicação;
- Economia pela relação preço/performance em relação aos computadores de grande porte;
- Extensibilidade através da configuração e reconfiguração dinâmica;
- Relação custo/eficiência através do compartilhamento dos recursos do sistema operacional;
- Escalabilidade e portabilidade através da modularidade;
- Disponibilidade pelo fato da perda de uma máquina não provocar a parada do sistema.

O desenvolvimento de aplicações distribuídas, onde todos os componentes colaborem eficientemente, confiavelmente, com transparência, e com escalabilidade é uma tarefa complexa. Grande parte desta complexidade provem de limitações das técnicas e ferramentas usadas para desenvolver *software* de aplicações distribuídas.

Os sistemas distribuídos possuem algumas dificuldades como:

- Maior complexidade dos mecanismos de segurança;

- A rede pode saturar ou causar outros problemas.

Aplicações sequenciais em sistemas centralizados usam modelos tradicionais onde atividades e dados são manipulados isoladamente. Em sistemas distribuídos atividades e dados estão intrinsecamente distribuídos e compartilhados. Distribuição produz modelos mais complexos e aplicações mais difíceis de desenvolver. A meta dos pesquisadores, hoje, é prover um modelo de transparência em sistemas distribuídos análogo ao existente nos sistemas centralizados.

Sistemas operacionais distribuídos e plataformas estão sendo propostas oferecendo transparência no tocante à: falha, acesso, localização, concorrência, replicação, migração, performance e escala.

Algumas considerações devem ser feitas para delinearmos a natureza distribuída. O sistema distribuído tem características próprias, por este motivo devemos entender alguns fundamentos e características que o suporta.

2.1.1. Considerações de hardware

As máquinas com múltiplas UCP's são classificadas, levando-se em consideração o número de fluxos de instrução e o número de fluxos de dados, em:

SISD: Computadores monoprocessados, onde existe um único fluxo de instrução e um único fluxo de dados;

SIMD: Máquinas que aplicam uma única instrução a um conjunto de dados de forma paralela;

MIMD: Máquinas independentes, cada uma com seu próprio programa e com seus próprios dados.

As máquinas MIMD são divididas em duas categorias: Multiprocessadas, que compartilham memória , e multicomputadores, que não compartilham memória.

Baseado na arquitetura da rede de interconexão, esta categoria de máquina pode ser dividida em:

Baseada em barramento, onde existe um meio que conecta todas as máquinas;

E *comutada*, onde todas as máquinas estão ligadas entre si, havendo possibilidade de se modificar o perfil da conexão. A mensagem é enviada carregando a decisão de qual rota deve tomar ao chagar em um determinado ponto.

2.1.2. Considerações sobre software

Os sistemas operacionais não são claramente classificados, mas, ainda assim, é possível classificá-los. Tanenbaum [12] os classificou de duas maneiras:

Fortemente acoplados – Sistema estabelece uma estreita integração e compartilhamento de recursos caracterizando sistemas operacionais distribuídos. O ponto fundamental neste tipo de sistema é a transparência, que permite ao usuário o uso do mesmo sem a necessidade de saber qual máquina está sendo utilizada e onde residem seus arquivos. Exemplo: Sistemas multiprocessadores, que podem trocar dados na velocidade de suas memórias;

Fracamente acopladas – permitem que máquinas e usuários de um sistema distribuído sejam fundamentalmente independentes e ainda interagir de forma limitada, quando necessário. Exemplo: Um grupo de PCs numa LAN que compartilham recursos como impressoras e banco de dados, mas que cada um deles possuem suas próprias UCPs, memórias, discos e sistemas operacionais.

O conceito de fortemente acoplado tende a ser mais usado em sistemas paralelos, que trabalha em um único problema, e o conceito de fracamente acoplado serve de base na modelagem de sistemas distribuídos que trabalham em vários problemas independentes.

Reforçando a idéia de acoplamento discutida acima, onde vimos que sistemas fortemente acoplados aproxima-se de sistemas operacionais distribuídos e os fracamente

acoplados aproxima-se dos sistemas operacionais de redes, podemos verificar algumas diferenças chaves entre eles, tais como:

- O sistema operacional distribuído necessita que todos os nós rodem o mesmo sistema operacional, enquanto o sistema em rede pode trabalhar com vários sistemas operacionais;

- No sistema distribuído existe uma única fila de execução de processos, enquanto no sistema de rede podem existir várias filas de execução;

- O sistema distribuído parece um sistema monoprocessado virtual, enquanto o sistema de rede não possui este tipo de transparência.

2.1.3. Características dos Sistemas Distribuídos

O desenho de um sistema distribuído objetiva construir um sistema de alta performance, confiável, escalonável, consistente e seguro. Projetos de sistemas distribuídos devem corresponder esta expectativa e cobrir todos os requisitos necessários para que o sistema seja considerado um bom sistema.

As características principais dos sistemas distribuídos são:

- Compartilhamento de recursos
- Sistemas abertos
- Concorrência
- Escalabilidade
- Tolerância a falhas
- Transparência

Podemos também mencionar confiabilidade e performance.

Detalhando as características citadas.

- ***Compartilhamento de recursos***

É a característica principal dos sistemas distribuídos que afeta fortemente as arquiteturas de *software* que estão disponíveis nestes sistemas. Recursos podem ser itens de dados (arquivos, banco de dados, etc ...), *software* ou componentes de *hardware* (discos, impressoras, etc.). Os recursos no sistema distribuído são fisicamente encapsulados dentro de um computador e podem somente ser acessados por outro computador via comunicação.

Estes recursos para serem compartilhados, necessitam de um *software* que ofereça uma interface de comunicação disponibilizando o recurso para ser acessado, manipulado e atualizado com segurança e consistência. Este *software* é conhecido como gerente de recurso e é responsável por prover uma política de gerenciamento e um conjunto de métodos para acesso ao recurso. Para que isso seja possível é necessário:

- Um esquema de nomeação para cada classe de recurso;
- Habilitar acesso a recursos individuais a partir de qualquer local;
- Mapeamento de nomes de recursos com endereços de comunicação;
- Uma coordenação de acesso concorrente ao recurso de forma a evitar inconsistências.

Os usuários dos recursos acessam os mesmos comunicando-se com os gerenciadores de recursos. Através dessas perspectivas, define-se dois modelos de sistemas distribuídos :

Modelo cliente-servidor : nesse modelo, existem um conjunto de processos servidores, atuando como gerenciadores de recursos de um determinado tipo, e outro conjunto de processos clientes, que requerem serviços de algum recurso compartilhado. Nada, no entanto, impede que processos servidores necessitem também de serviços de recursos compartilhados. Portanto, alguns processos podem ser tanto clientes quanto servidores.

Nessa visão do modelo cliente-servidor, cada processo servidor deve ser visto como um provedor centralizado dos recursos que gerência. Como a centralização não é um conceito desejável em ambientes cliente-servidor, devemos separar os conceitos de *serviços* e *servidores*. Podemos considerar que os serviços podem ser oferecidos por diversos processos servidores rodando em diferentes computadores cooperativamente através de conexões de redes de computadores.

Modelo baseado em objeto : nesse modelo, cada recurso compartilhado é visto como um objeto, que devem ser unicamente identificados e trafegar livremente pela rede sem que percam sua identidade. Quando um programa requerer um serviço de algum recurso compartilhado, deve enviar uma mensagem ao objeto correspondente.

Assim como no modelo cliente-servidor, os objetos podem atuar tanto como usuários de recursos quanto como gerenciadores. Correspondentemente ao conceito de gerenciadores de recursos no modelo cliente-servidor, usa-se o termo *gerenciador de objetos* para especificar o conjunto de procedimentos e dados que caracterizam uma classe de objetos. Nesse modelo, um gerenciador de objeto de um determinado tipo de recurso necessariamente deve estar localizado onde o objeto estiver, para que o gerenciador possa acessar o estado do mesmo.

- ***Concorrência***

Sistemas distribuídos são compostos por muitos computadores, cada um deles possuindo um ou mais processadores, isto proporciona a execução de processos de maneira concorrente e paralela devido as atividades separadas dos usuários, a independência dos recursos e a localização de processos servidores em computadores separados. Por esta razão, acessos e atualizações concorrentes a recursos compartilhados devem ser sincronizados.

- ***Escalabilidade***

Dizemos que um sistema distribuído é escalonável, quando podemos aumentar o número de recursos compartilhados sem a necessidade de mudar os *software* de aplicação e sistemas.

Em sistemas centralizados os recursos de memória, processadores e canais de E/S são limitados. No sistema distribuído, estas limitações são minimizadas, já que potencialmente o sistema pode ser constituído de um número ilimitado de computadores.

- ***Tolerância a falhas***

Pelo fato de sistemas distribuídos possuírem múltiplas partes de *hardware* e de *software* funcionando em conjunto, as chances de alguma dessas partes falhar é bem maior do que ocorreria num sistema simples. Em contrapartida, uma falha em um nó não necessariamente compromete os demais. Para que um sistema seja confiável, ele deverá continuar funcionando corretamente mesmo com a presença de certos tipos de defeitos ou interferências indevidas.

Um sistema distribuído pode apresentar falhas de várias tipos, dentre as quais aquelas provenientes de defeitos nas linhas de comunicação que interligam os processadores e aquelas referentes a colapsos nos processadores.

O projeto de sistemas de computação tolerante à falhas é baseado em duas abordagens: Redundância de *hardware*, quando é usado componentes redundantes e redundância de *software*, que é um projeto de programas para recuperação em caso de falhas.

- ***Transparência***

A transparência são alguns aspectos do sistema distribuídos que não são percebidos pelo usuário (programador, desenvolvedor de sistemas, usuário ou programa de aplicação). A transparência oculta do usuário e dos programadores de aplicações alguns recursos, dando a entender que se trata de uma única coleção de recursos.

O número de transparências para sistemas distribuídos, ainda está sendo definido. É importante compreender que nem todas são apropriadas para todos os sistemas, ou estão disponíveis no mesmo nível de interface. Toda transparência tem um custo associado, e isto é extremamente importante para quem deseja implementar um sistema distribuído. As transparências, identificadas pelo ANSA Reference Manual e o ISO Reference Model for Open Distributed Processing (RM-ODP), são:

- Transparência de acesso – não existe diferença entre métodos de acesso local e remoto. Por exemplo: o acesso a uma impressora local deve ser idêntico ao acesso a uma impressora remota;

- Transparência de localização – os detalhes da topologia do sistema não deve ser uma preocupação do usuário. A localização de um objeto do sistema não é visível ao usuário ou programador;

- Transparência de concorrência – usuários e aplicações devem estar habilitados a acessar dados ou objetos compartilhados sem interferência de um no outro. Isto requer mecanismos complexos em sistemas distribuídos.

- Transparência de replicação – se um sistema prover replicação (por razões de disponibilidade ou performance) usuários ou programas aplicativos não devem tomar conhecimento;

- Transparência a falhas – se houver uma falha de *hardware* ou *software*, elas não devem ser percebidas pelo usuário. Isto pode ser difícil para sistemas distribuídos, visto que é possível uma falha de um sistema de comunicação, e isto pode não ser relatado. Contudo, essa transparência permite que os usuários e programas aplicativos terminem suas tarefas mesmo ocorrendo falhas de *hardware* ou *software*;

- Transparência de migração – se objetos (processos ou dados) migram (para prover melhor performance, ou confiabilidade, ou para ocultar diferenças entre máquinas), isto não deve afetar a operação de usuários ou sistemas aplicativos;

- Transparência de performance – permite que o sistema seja reconfigurado para melhorar performance caso a carga de trabalho varie. Isto pode requerer um complexo mecanismo de gerenciamento de recursos. E pode não ser aplicados nos casos em que os recursos são acessíveis somente via redes de baixa performance.

- Transparência de escalabilidade – permite que os sistemas e aplicações sejam expandidos em escala sem afetar a estrutura do sistema ou algoritmos das aplicações.

2.1.4. Comunicação entre Processos

Durante a execução do código, os processos precisam comunicar-se entre si. Esta comunicação pode ser entre os encadeamentos do mesmo processo ou entre encadeamentos de processos diferentes. A comunicação pode ser com memória compartilhada, se a comunicação for com encadeamentos do mesmo processo ou entre processos que executam numa única máquina, ou com memória distribuída, se a comunicação for entre encadeamentos de diferentes processos, que não compartilham memória ou que executem em diferentes nodos da rede.

2.1.5. Memória compartilhada

Os processos, ou encadeamentos, podem comunicar-se entre si diretamente através do compartilhamento de memória, podendo ler e escrever diretamente na área de memória compartilhada, quando estão executando numa única máquina [3].

Neste caso, o sistema operacional apenas proporciona uma área de memória que pode ser compartilhada por mais de um processo, ficando por conta do programador a implementação da comunicação e sincronização entre os processos.

2.1.6. Memória distribuída

A comunicação entre processos que não compartilham memória é feita através de mecanismos de comunicação implementados pelo sistema operacional. Diversas modelos de implementação dos mecanismos de comunicação entre processos podem ser encontrados, utilizando troca de mensagens:

2.1.6.1. Send e Receive

Na comunicação entre processos no sistema de mensagens existem duas operações básicas: **send**(*to*, *message)* e **receive**(*from*, *message*). Se dois processos precisam comunicar-se, o

processo que vai enviar a mensagem faz o **send**(*to*, *message)* e o outro processo faz o **receive**(*from*, *message*). Para que isso ocorra, deve haver um vínculo de ligação entre os processos. Esse elo de ligação pode ser implementado de diversas maneiras podendo ser através de uma área de memória compartilhada do computador ou através de uma rede entre computadores.

A comunicação entre dois processos pode ser direta ou indireta. Na comunicação direta, cada processo que deseja enviar ou receber uma mensagem precisa explicitamente nomear o receptor ou emissor da comunicação. Os processos precisam simplesmente conhecer a identificação do outro para a comunicação entre si, de modo que um vínculo de ligação é estabelecido automaticamente entre eles. Um vínculo é estabelecido somente entre dois processos. Na comunicação indireta, o envio e recepção de mensagens é feito através de *mailboxes*, também chamados de portas. Um *mailbox* pode ser abstratamente visto como um objeto no qual as mensagens são colocadas pelos processos e de onde podem ser removidas, possuindo uma identificação única. Os processos podem comunicar-se entre si se tiverem um *mailbox* compartilhado, estabelecendo um vínculo entre si. Um vínculo pode ser associado com mais de dois processos e dois processos podem ter vários vínculos entre si, cada um associado a um *mailbox*.

Uma fila é associada com cada mensagem. Os processos que enviam adicionam mensagens na fila e os processos que recebem retiram mensagens da fila. A comunicação através de mensagens pode ser síncrona ou assíncrona. Na forma de comunicação síncrona, os processos emissores e receptores sincronizam a cada mensagem. Nesse caso, tanto o **send** como o **receive** são operações bloqueantes, pois toda a vez que um **send** é emitido, o processo emissor fica bloqueado até que o **receive** correspondente seja utilizado. Da mesma forma, toda a vez que um **receive** é utilizado, o processo fica bloqueado até a chegada de uma mensagem. Na forma assíncrona de comunicação o uso da operação **send** é não-bloqueante em que o processo emissor continua a execução após seu uso e a operação **receive** pode ser bloqueante ou não bloqueante.

2.1.6.2. Cliente/Servidor

Esta forma de comunicação é desenvolvida para suportar as funções de troca de mensagens numa interação cliente-servidor, em que um grupo de processos cooperantes, chamados

servidores, oferecem serviços ao usuários, chamados clientes. Normalmente a comunicação requisição-resposta é síncrona, porque o processo cliente fica bloqueado até que a resposta chegue do servidor.

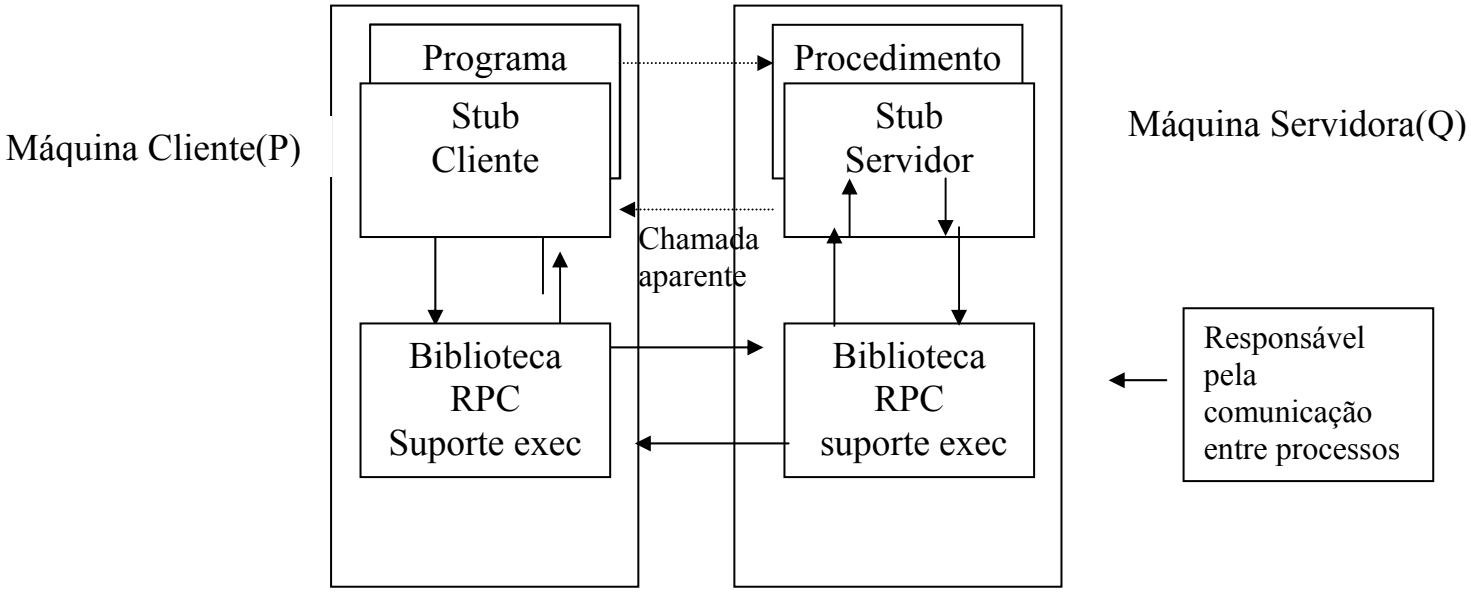
Uma comunicação assíncrona pode ser implementada no caso dos processos clientes continuarem executando após a requisição e necessitarem da resposta posteriormente.

2.1.6.3. RPC

As chamadas de procedimentos à distância (*Remote Procedure Call*) são um eficiente mecanismo de comunicação usados nos sistemas distribuídos e nos sistemas sem memória comum, como máquinas paralelas, organizados de acordo com o modelo cliente/servidor, segundo [9].

O modelo RPC é uma extensão distribuída do modelo procedural existente nas linguagens de programação tradicionais. A idéia principal é permitir a chamada de um procedimento em uma máquina por um processo sendo executado em outra máquina. Em um RPC, um programa em uma máquina P (o cliente), chama um procedimento em uma máquina Q (o servidor), enviando os parâmetros adequados. O cliente é suspenso e a execução do procedimento começa. Os resultados são enviados da máquina Q para máquina P e o cliente é acordado. Os componentes de um sistema RPC e o fluxo de comunicação entre eles está representados abaixo:

Figura 2.1 Sistema RPC e seus componentes



Toda função que pode ser chamada à distância é definida no lado do cliente por um procedimento chamada *stub cliente*. Esse procedimento recebe os parâmetros e os coloca em uma mensagem, adicionando a identificação do procedimento chamado e transmite essa mensagem ao processo servidor. O servidor, quando recebe a mensagem, extrai os parâmetros e chama o procedimento especificado na mensagem. No fim da execução do procedimento é realizada a operação inversa, colocando os resultados e enviando a mensagem de resposta ao processo cliente, pelo procedimento stub server. Finalmente, o procedimento stub client recebe os resultados, termina e o controle retorna ao cliente.

No nível RPC, um serviço pode ser visto como um módulo com uma interface que exporta um conjunto de procedimentos para operar em algum nível de abstração de dados ou recursos.

Os sistemas RPC podem ser divididos em duas classes:

- Mecanismo RPC é integrado com uma linguagem de programação particular. Como exemplo temos a linguagem Argus desenvolvida por Liskov no MIT;

- Uma linguagem de definição de interface é utilizada para se descrever as interfaces entre cliente e servidores. Exemplo: Sun RPC.

Resumindo, os passos de um procedimento RPC são (observar figura 2.1):

- O procedimento do cliente(programa chamador) chama o stub do cliente.

- O stub do cliente constrói uma mensagem e envia para a biblioteca RPC de suporte a execução.

- A biblioteca RPC estabelece a comunicação com a biblioteca RPC servidora.

- A biblioteca RPC servidora entrega a mensagem ao stub servidor.

- O stub servidor desempacota os parâmetros constantes da mensagem e chama o servidor.

- O servidor realiza o seu trabalho e retorna o resultado para o buffer dentro do stub servidor.

- O stub servidor empacota os resultados em uma mensagem e entrega a biblioteca RPC servidora.

- A biblioteca RPC estabelece a ligação com a biblioteca RPC cliente e passa-lhe a mensagem.

- A biblioteca RPC cliente entrega a mensagem ao stub cliente.

- O stub cliente desempacota a mensagem e entrega os resultados ao cliente.

O mecanismo RPC oferece aos programadores uma abstração similar à passagem de mensagens entre procedimentos, facilitando a programação. Também, esconde a maioria dos detalhes relativos à transferência dos dados. No entanto, algumas limitações são encontradas como:

- A natureza síncrona das RPCs restringe o grau de paralelismo que pode ser obtido em um sistema distribuído;

- RPCs lidam somente com questões relativas à comunicação e não tratam de conceitos de mais alto nível, como persistência e replicação, o que compromete sua ampla utilização em ambientes internet;

- RPCs impõem o modelo cliente/servidor às aplicações distribuídas;

- Utilizando RPCs, o usuário deve lidar com localização do alvo da chamada.

2.1.7. Sincronização de processos

Conforme visto, a concorrência entre processos é uma característica intrínseca dos sistemas distribuídos. Esta concorrência tem que ser saudável para os processos envolvidos, não

devendo haver nenhuma alteração no resultado final esperado. Para que isto seja garantido, é necessário que haja um sincronismo na execução destes processos.

Para escrever qualquer tipo de programa concorrente, é necessário sincronizar os diversos processos que compõem o sistema, principalmente para eliminar o problema do não-determinismo desse tipo de programas, garantindo a correta ordenação das atividades de leitura e gravação de dados na memória.

Uma seção crítica é uma seção de código que precisa estar habilitada para ser executada por completo, sem interrupção. Se um processo está na seção crítica e outro processo ganha o processador e ele também deseja entrar na seção, este último fica bloqueado aguardando a liberação por parte do primeiro para poder então entrar na seção crítica.

Todos os dados globais devem ser protegidos por variáveis de sincronização para evitar bugs e problemas nos programas concorrentes. Para isso, vários tipos de variáveis de sincronização foram desenvolvidas para atender diferetnes propósitos:

2.1.7.1.Mutexes

O *lock* de exclusão mútua é a mais simples e mais primitiva variável de sincronização. Ele provê uma simples e absoluta propriedade para a seção de código que está entre a chamada ***mutex_lock()*** e ***mutex_unlock()***. O primeiro processo ou encadeamento que chama *lock* adquire direito de prosseguir e os demais encadeamentos que chamarem *lock* ficarão bloqueados até que o primeiro faça uma chamada *unlock.*

O código abaixo ilustra o uso de mutex:

Encadeamento 1	**Encadeamento 2**
mutex_lock(&m); /*entra na região crítica	mutex_lock(&m); /*entra na região crítica
global++; /*incrementa variável	local = global; /*utiliza variável 'global'
mutex_unlock(&m)/*deixa região crítica;	mutex_unlock(&m); /*sai da região crítica

2.1.7.2. Semáforos

Um semáforo contador é uma variável que pode ser incrementada arbitrariamente, mas que só pode ser decrementada até zero. Quando o semáforo é criado, a variável de controle é inicializada com 1. Uma operação ***sema_post()*** (também chamada operação "V") incrementa o semáforo, se um ou mais processos estiverem bloqueados neste semáforo, impedidos de completarem uma operação *sema_wait*()(P). Um deles será escolhido pelo sistema, sendo-lhe então permitido completar a operação (P). A operação ***sema_wait()*** (também chamada operação "P") decrementa o semáforo, ela verifica se o valor do semáforo é maior que 0(zero). Se o valor for maior que zero então a operação sucede, senão o encadeamento que chamou é bloqueado até que um outro encadeamento incremente-o.

Exemplo de um semáforo:

Encadeamento 1

```
Repeat              /*inicia  operação de repetição
sema_wait(&s);  /*decrementa o contador
global++;           /*faz uma operação
sema_post(&s);  /*incrementa o contador
until 0;            /*condição de repetição
```

Encadeamento 2

```
Repeat      /*inicia repetição
sema_wait(&s);  /*decrementa contador
local = global;     /*usa variavel 'global'
sema_post(&s);  /*incrementa o contador
until 0;            /*condição de repetição
```

Capítulo 3

3. A arquitetura Cliente/Servidor

3.1. Origem e propósitos

De acordo com Sadoski [8] , o termo cliente/servidor foi usada pela primeira vez nos anos 80 em referência aos computadores pessoais(PCs) em uma rede. O modelo cliente/servidor atual começou a ganhar aceitação no final dos anos 80. A arquitetura cliente/servidor é versátil, baseada em mensagens e possui uma infraestrutura modular, características para melhorar a usabilidade, flexibilidade, interoperabilidade e escalabilidade, se compararmos com a computação centralizada dos mainframes de compartilhamento de tempo.

Um cliente é definido como um requisitor de serviços e um servidor é definido como o provedor de serviços. Uma única máquina pode ser um cliente e ao mesmo tempo um servidor, dependendo apenas da sua configuração de software.

Este capítulo fornece um resumo de algumas arquiteturas cliente/servidor comuns e para completar, também fornece um resumo das arquiteturas de mainframes e de compartilhamento de arquivos.

3.2. Detalhamento técnico

- *Arquitetura de Mainframe:*

Não é uma arquitetura cliente/servidor. Com arquiteturas de mainframes toda a inteligência está no computador principal. Usuários interagem com o mainframe através de um terminal que captura as teclas pressionadas e enviam esta informação ao computador central. Arquiteturas de mainframe não são presas a plataformas de hardware. A interação do usuário pode ser feita utilizando PCs ou estações Unix. A limitação desta arquitetura é que ela não suporta muito facilmente interfaces gráficas ou acesso a múltiplos bancos de dados, geograficamente distribuídos.

- *Arquitetura de compartilhamento de arquivos*

Também não é uma arquitetura cliente/servidor. As primeiras redes de PCs eram baseadas em compartilhamento de arquivos, o servidor fazia o download dos arquivos de uma localização compartilhada para seu ambiente. O processo do usuário então roda no ambiente do servidor. Arquiteturas de compartilhamento de arquivos funcionam se o volume de compartilhamento é baixo, e se as atualizações de conteúdos são poucas e o volume de dados a ser transferido é pequeno. Nos anos 90, esta arquitetura se defasou porque a capacidade de compartilhamento foi diminuindo uma vez que o número de usuários on-line foi crescendo (esta arquitetura só conseguia satisfazer apenas 12 usuários simultaneamente) e como as interfaces gráficas para usuários se tornaram populares, isto fez com que os mainframes e os terminais parecessem desatualizados. Os PCs agora estão sendo utilizados em arquiteturas cliente/servidores.

- *Arquitetura Cliente/servidor*

Como resultado de uma série de limitações da arquitetura de compartilhamento de arquivos, a arquitetura cliente/servidor emergiu. Esta abordagem introduziu o servidor de banco de dados como o substituto do servidor de arquivos. Utilizando sistemas de manutenção de banco de dados relacionais (DBMS – Database Management System), as consultas dos usuários poderiam ser respondidas diretamente. A arquitetura cliente/servidor reduziu o tráfego na rede pois respondia somente o resultado da consulta do usuário, ao invés de todo o download do arquivo compartilhado. Nas arquiteturas cliente/servidor, Chamadas de Procedimento Remoto (RPC – Remote Procedure Calls) ou chamadas SQL são tipicamente utilizadas para a comunicação entre o cliente e o servidor.

3.3. Arquitetura cliente/servidor 2 camadas

3.3.1. Visão geral

Com a arquitetura cliente/servidor 2 camadas [8], a interface do sistema é usualmente colocada na máquina do cliente e os serviços de banco de dados são usualmente colocados em um servidor que em geral é uma máquina mais potente, capaz de servir vários clientes.

O processamento é separado entre o cliente e o servidor de banco de dados. O servidor de banco da dados possui stored procedures(funções armazenadas) e triggers (gatilhos, que disparadores de funções). Existem uma variedade de produtos comerciais que fornecem ferramentas para simplificar o desenvolvimento de aplicações para a arquitetura cliente/servidor 2 camadas.

A arquitetura cliente/servidor com 2 camadas é uma boa solução para a computação distribuída quando os grupos de trabalho são na ordem de dezenas ou centenas de pessoas interagindo no sistema simultaneamente. Certamente há algumas limitações. Quando o número de usuários excede a casa das centenas, a performance tende a deteriorar. Esta limitação é resultado do servidor mantendo a conexão "viva" com cada cliente, mesmo quando nenhum trabalho está sendo feito. Uma segunda limitação da arquitetura em 2 camadas é que a implementação de serviços utilizando funções de um determinado fabricande de banco de dados tente a restringir a flexibilidade. Finalmente, as implementações de arquiteturas 2 camadas fornecem uma limitada flexibilidade em mover ou reparticionar funcionalidades do sistema de um servidor para outro, sem manualmente re-gerar todo o código da aplicação.

3.3.2. Detalhamento

A arquitetura cliente/servidor 2 camadas consiste em 3 componentes distribuídos em 2 camadas: cliente (requisita os serviços) e o servidor (provedor de serviços), os três componentes são:

1. Interface com o usuário (sessão, entrada de texto, mensagens)
2. Manutenção do processamento (desenvolvimento doprocesso, monitoramento do processo, serviços)
3. Manutenção da base de dados (base de dados)

As duas camadas alocam a interface com o usuário apenas no cliente. Coloca a manutenção da base de dados no servidor e divide o processamento entre o cliente e o servidor, criando assim duas camadas. A figura 3.1 nos mostra a arquitetura 2 camadas.

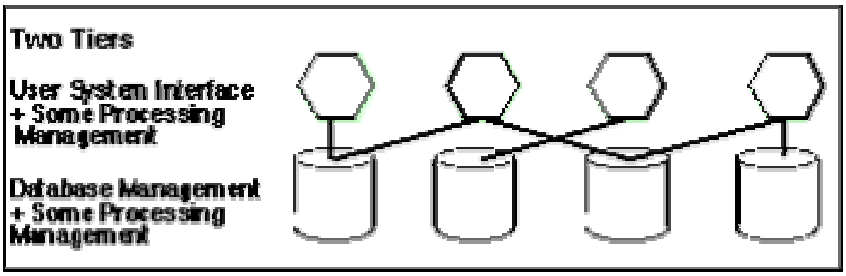


Figura 3.1: A arquitetura cliente/servidor 2 camadas [8]

Em geral, a interface com o usuário invoca serviços do servidor de banco de dados. Na maioria das aplicações 2 camadas, a maior parte da aplicação/processamento está no cliente. O servidor de banco da dados geralmente fornece o processamento apenas do acesso às informações do banco(geralmente implementadas na forma de *stored procedures*). Clientes se comunicam com o servidor através de consultas SQL. Deve ser notado que a conexão entre as camadas pode ser dinamicamente alteradas dependendo da requisição ou do serviço que o usuário estiver requisitando.

Comparando com a arquitetura de compartilhamento de arquivos, que também suporta sistemas distribuídos, a arquitetura cliente/servidor com 2 camadas melhora a flexibilidade e a escalabilidade pois disponibiliza duas camadas na rede de computadores. As duas camadas melhoram a usabilidade porque se torna mais fácil fornecer interfaces com o usuário customizadas.

Também é possível que um servidor funcione como um cliente para um outro servidor, em uma arquitetura cliente/servidor hierárquica. Isto é conhecido como arquitetura cliente/servidor em 2 camadas encadeada.

3.3.3. Considerações

A arquitetura de software em duas camadas é bastante usada em processamentos não críticos e onde a manutenção e o processamento do sistema não são complexos. Este design é usado frequentemente ondea quantidade de transações entre cliente e servidor é pequena. A arquitetura 2 camadas funciona bem para um ambiente em que as regras de negócio não mudem muito frequentemente, e quando o grupo de usuários esperado é de no máximo 100, como por exemplo em uma empresa de pequeno porte.

3.3.4. Custos e limitações

- **Escalabilidade:**

O design em duas camadas suporta bem até 100 usuários em uma rede. Aparentemente acima deste número de usuários, a capacidade do sistema é excedida. Isto acontece porque o servidor mantem a conexão com seus clientes até mesmo quando nenhum trabalho está sendo feito, desta forma saturando a conexão.

Implementando a lógica de negócios em *stored procedures* pode limitar a escalabilidade porque quanto mais lógica de negócios é movida para o servidor de banco de dados, maior processamento é requerido. Cada cliente executa uma parte do código da sua aplicação no servidor, e isto faz com que o número de usuários total que poderia ser atendido seja reduzido.

- **Interoperabilidade:**

A arquitetura em duas camadas limita a interoperabilidade porque usa *stored procedures* para implementar lógicas de processamento complexas, como funções para garantir a integridade do banco de dados, e geralmente essas funções são implementadas utilizando a linguagem proprietária do fabricante do banco de dados. Isto significa que mudar para outro banco, ou interoperar com mais de um tipo de banco dados, as funções terão que ser reescritas. Em geral as linguagens proprietárias dos bancos de dados para escrever *stored procedures* não são tão completas como as linguagens de programação comuns como C++ e Java, que fornece um ambiente robusto de programação, com ambiente para testes e correção de erros, controle de versão e uma extensa biblioteca de classes.

- **Administração e configuração do sistema:**

A arquitetura duas camadas pode ser difícil de administrar e manter pois como parte da aplicação reside no cliente, a cada atualização necessária tem que ser entregue, instalada e testada em cada cliente. A falta de padrão entre os sistemas e máquinas clientes e a falta de controle de suas configurações acarreta em um aumento significativo de trabalho.

3.3.5. Conclusões

Uma alternativa ao uso da arquitetura duas camadas é a arquitetura em três camadas, que será discutida a seguir.

Ao preparar uma aplicação em 2 camadas para uma possível migração para uma arquitetura em três camadas, os seguintes passos devem ser observados para que haja uma transição pouco custosa e com poucos riscos:

1. Eliminar a diversidade de aplicações assugurando-se de que haja uma biblioteca de classes multiplaforma(não dependende de hardware) e as ferramentas de desenvolvimento apropriadas.
2. Desenvolver serviços menores e mais específicos, e permitir o seu acesso através de interfaces bem claras.
3. Utilizar a Linguagem de Definição de Interfaces (IDL – Interface Definition Language) para modelar as interfaces dos serviços.
4. Implementar diferentes serviços em classes separadas, e não em um código fonte único.
5. Aumentar a flexibilidade e a funcionalidade distribuída inserindo elementos de serviço comuns em DLLs(Bibliotecas dinâmicas), deste modo o código não precisa ser compilado, e já está pronto para ser usado.

3.4. Arquitetura cliente/servidor três camadas

Ainda de acordo com Sadoski [8] , a arquitetura cliente/servidor em três camadas, também conhecida como arquitetura cliente/servidor multicamadas, surgiu para suprir as limitações que da arquitetura em duas camadas. Na arquitetura em três camadas, uma camada intermediária foi adicionada entre a camada cliente e o servidor. Existem uma variedade de formas diferentes de se implementar esta camada intermediária, como servidores de mensagens, servidores de aplicação e monitores de processamento de transições. A camada intermediária pode armazenar requisições de clientes em uma fila, então o cliente pode

requisitar seu pedido à camada intermediária e desconectar, pois a camada intermediária vai acessar o banco de dados e retornar a resposta ao cliente posteriormente. A camada intermediária também fornece serviços diferenciados dependendo da prioridade do trabalho, e agendamento de tarefas. A arquitetura em três camadas tem mostrado uma melhora de performance para grandes grupos de usuários, na casa dos milhares, e possui uma flexibilidade maior se comparado com a arquitetura duas camadas. Flexibilidade pode ser bem simples a ponto de particionar código da aplicação em módulos e colocá-los em diferentes computadores se tornar uma simples operação de "arrastar e soltar" com o mouse. Uma limitação da arquitetura em três camadas é que o desenvolvimento de aplicações neste modelo é mais difícil do que o desenvolvimento em duas camadas.

3.4.1. Detalhes técnicos

Uma arquitetura cliente/servidor distribuída, como mostra a figura 3.2, inclui a interface com o usuário em sua primeira camada, onde serviços ao usuário são disponibilizados.

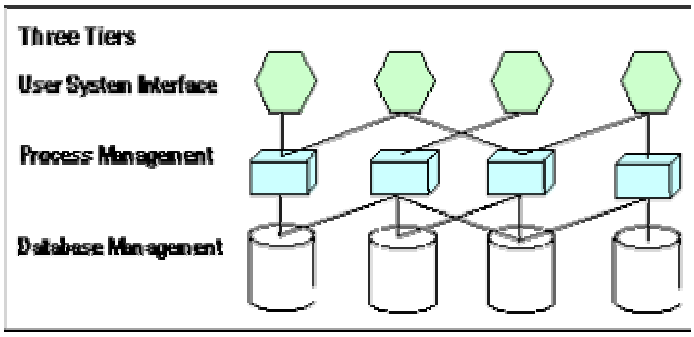


Figura 3.2: Arquitetura cliente/servidor 3 camadas [8]

A terceira camada provê os serviços de banco de dados e pode ser otimizado sem utilizar as linguagens proprietárias dos fabricantes. Um componente de manipula os dados garante que os dados estejam consistentes pelo ambiente distribuído, pelo uso de replicação, travamento de tabelas, e checagem de consistência. Note que as conexões entre camadas podem dinamicamente ser alteradas dependendo da requisição do usuário por informações ou serviços.

O servidor da camada intermediária, também chamado de servidor de aplicações, melhora a performance, flexibilidade, manutenibilidade, reusabilidade e escalabilidade ao centralizar a lógica do sistema. Lógica do sistema centralizada faz com que a administração do sistema e alterações se tornem fáceis de serem executadas, isto significa que uma mudança

no sistema, só será feita uma vez e colocada na camada intermediária, e o servidor de aplicações já disponibilizará esta nova funcionalidade através da rede. Em outras arquiteturas, uma mudança de uma função ou serviço fará com que sejam escritas em todas as aplicações clientes, quando se utilizando a arquitetura duas camadas.

A tabela abaixo resume as principais características de cada camada.

Camada	Tipo de Serviço	Característica	Responsável Por	Ferramentas
Serviços de interface com usuário	Aplicações cliente	Interface GUI	Apresentação e navegação	Linguagens de Quarta geração e aplicações desktop
Serviços de regras de negócios	Servidores de regras de negócio	Objetos de negócio	Políticas de negócio, regras e segurança	Linguagens de Quarta geração, Cobol, C
Serviços de banco de dados	Servidores de bancos de dados	Gerenciadores de dados	Integridade dos dados	Gerenciadores de banco de dados

Figura 3.3 Descrição das Camadas [6]

Algumas vezes a camada intermediária é dividida em uma ou mais unidades, com diferentes funções, nestes casos a arquitetura é também chamada de multicamadas. Este é o caso de algumas aplicações na Internet. Estas aplicações tipicamente possuem clientes "magros" escritos em HTML e servidores de aplicação escritos em C++ ou Java, a distância entre essas duas camadas é muito grande, então existe uma camada intermediária, que seria o servidor de páginas web, capaz de receber as requisições dos clientes da Internet e gerar o código HTML de volta, baseado nas respostas da camada de negócios. Esta camada adicional fornece um maior isolamento entre a interface da aplicação e a lógica da aplicação.

3.4.2. Conclusões

Construir uma aplicação em três camadas é um trabalho complexo. Ferramentas de desenvolvimento que suportam a arquitetura três camadas ainda não fornecem todas os serviços desejados para suporte ao desenvolvimento de aplicações distribuídas.

Um problema que pode surgir é que a separação da interface com o usuário, o processamento lógico da aplicação e a lógica de banco de dados nem sempre é tão óbvia. Alguns processos lógicos podem aparecem em todas as três camadas. A colocação de uma função em uma determinada camada pode ser feita baseada em alguns critérios como os seguintes:

- Facilidade de desenvolvimento e teste
- Facilidade na administração do serviço
- Escalabilidade dos servidores
- Performance, levando em conta processamento e tráfego na rede.

3.5. Diferenças entre duas e três camadas

Podemos fazer uma comparação entre as características das duas arquiteturas, lembrando que as características abaixo referem-se à aplicações que se encaixam nos pré-requisitos listados nas seções anteriores:

Característica	Duas Camadas	Três Camadas
Administração do sistema	Complexa (boa parte do gerenciamento no cliente)	Menos complexa (a aplicação pode ser gerenciada no servidor)
Encapsulamento de dados	Baixa (base de dados exposta)	Alta (o cliente invoca serviços ou métodos)
Performance	Média (muitas requisições SQL são enviadas através da rede)	Boa (apenas requisições de serviços são feitas entre cliente e servidor)
Escala	Baixa (gerenciamento limitado de links de comunicação com clientes)	Excelente (pode-se distribuir a carga de processamento entre diferentes servidores)
Reusabilidade	Baixa	Alta
Facilidade de desenvolvimento	Alta	Baixa (apesar de várias ferramentas de desenvolvimento disponíveis no mercado já oferecerem facilidades)

Figura 3.4 Duas Vs. Três Camadas [6]

Capítulo 4

4. A linguagem Java

4.1. Introdução

Java™ é uma linguagem orientada a objetos que foi desenvolvida por uma pequena equipe liderada por James Gosling na Sun Microsystems [10], seu desenvolvimento teve início em 1991. Inicialmente ela foi projetada para ser utilizada na programação de aparelhos como celulares e relógios, mas devido à explosão da Internet começou em 1995, ficou claro que Java era uma linguagem de programação ideal para aplicações na Internet.

Java era capaz de cobrir diversos tópicos importantes em software distribuído, tais como *interoperabilidade*, *portabilidade* e *integridade*. Quando inseridos em uma página web, programas Java são chamados de "applets". Os Applets em conjunto com JavaBeans™ provém ao desenvolvedor a flexibilidade de criar uma interface para o usuário mais sofisticada em uma página da web. Applets Java fornecem conteúdo executável, como janelas que respondem a eventos do usuário, como cliques do mouse, e uma interface gráfica com o usuário mais amigável e cheia de recursos, provida por classes da linguagem Java que dão suporte a uma enorme gama de aplicações.

Os applets Java são uma parte dominante no lado cliente de aplicações na Internet , porém no lado do servidor, onde roda o código que gera o conteúdo HTML para o cliente, a sua performance foi considerada fraca se comparado a linguagens como C++ ou linguagens de script como PERL. Contudo a situação está mudando com o lançamento do Java 2 Enterprise Editiom™(J2EE). J2EE é uma nova plataforma Java desenvolvida especificamente para atender às necessidades de um servidor de aplicações de grande porte. J2EE provê escalabilidade, interoperabilidade, segurança e confiabilidade

4.2. Detalhes técnicos

Java é uma linguagem de programação de alto nível, similar ao Smalltalk e similar em sintaxe à C e C++. Porém Java é muito menos complexo do que C++. Java é uma linguagem orientada a objetos, com tipagem forte, multi processada e robusta.

Java provém um coletor de lixo interno, suporta apenas herança simples e não utiliza ponteiros, portanto eliminando as três maiores fontes de erros da maioria dos programas escritos em C++. Como a sintaxe de Java é bem parecida com as já famosas C e C++, a linguagem se torna familiar à maioria dos desenvolvedores.

Programas Java começam como códigos fonte, que são compilador para o chamado "bytecode"(que é uma linguagem intermediária para a Máquina Virtual Java) e armazenados em um servidor ou em um computador local em arquivos ".class". Para executar um programa Java, o usuário chama a Máquina Virtual Java (JVM) que executa o bytecode Java. Diferentemente de outras linguagens de programação, o bytecode Java não é dependente de plataforma, nem é nativo de nenhum processador em especial, é uma abordagem "escreva uma vez, rode em qualquer lugar". Esta abordagem que provê neutralidade de plataforma significa que Java é uma linguagem portável nata.

O ambiente Java também provê uma extensa biblioteca de classes que provêm acesso aos mais variados tipos de sistemas operacionais. Atualmente a maioria dos navegadores web possuem uma Máquina Virtual Java pré-instalada, incluindo Netscape Navigator, Microsoft Internet Exploer e o Sun HotJava Browser. As plataformas como Microsoft Windows, MacOS, OS/2 Warp e Sun Solaris também possuem uma Máquina Virtual Java para executar códigos Java em aplicações no micro. Existe uma nova geração dos chamados *"Network Computers"* que já executam código java diretamente em seus processadores. A Sun está extendendo a disponibilidade da Máquina Virtual Java para que os programas Java possam ser escritos para aparelhos como pagers, celulares, telefones e televisões.

A plataforma Java provê portabilidade, segurança e confiabilidade natas, incluindo uma forte proteção de memória, encriptação de dados e assinaturas, e verificação de código durante a execução do programa. Java é designada para permitir que applets sejam baixados e executados sem que vírus ou código fonte alterado sejam inseridos. Isto foi feito colocando-se

restrições muito rígidos na funcionalidade dos applets, para prevenir ações maliciosas. Por exemplo: Applets não podem nem ler nem escrever no disco rígido local do usuário.

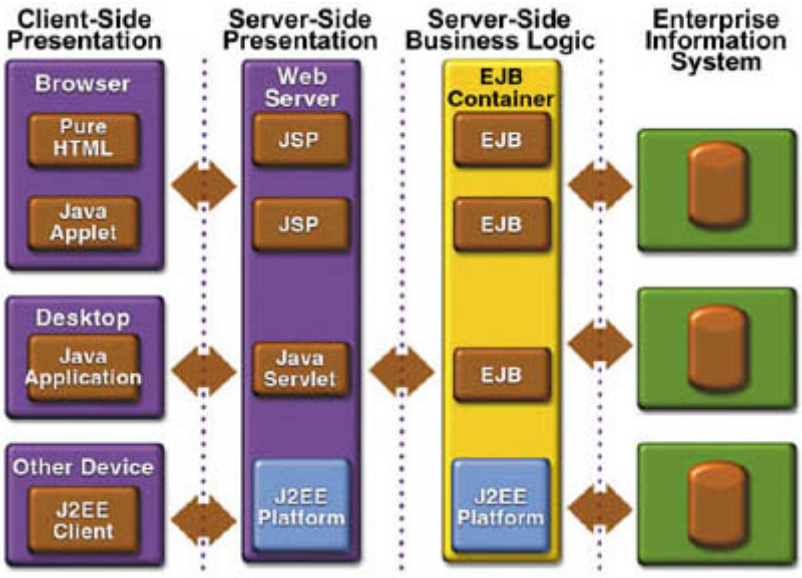


Figura 4.1 : Arquitetura J2EE [10]

Infelizmente, enquanto o modelo Java é teoricamente seguro, as várias "*implementações"* da Máquina Virtual Java continuam a mostrar sinais de fraqueza. A exploração das brechas de segurança ainda é alarmantemente comum. As ações de um applet são restritas ao seu "cantinho", o local onde o browser web aloca espaço em memória para o applet e em qual ele pode fazer o que quiser. Mas um applet Java não pode ler ou alterar nenhum dado fora do seu "cantinho". Consequentemente usuários podem criar e executar códigos Java não muito confiáveis sem comprometer seus sistemas computacionais. Aplicações Java também são proibidas de fazer conexões de rede para computadores em uma Intranet corporativa, então códigos maliciosos não podem explorar as brechas de seguranças que ainda não tenham sido descobertas.

Os applets não são suficientes para contruir sistemas computacionais de grande porte. Os applets como padrão entregam grande parte da funcionalidade aos clientes remotos, mas as aplicações de grande porte precisam de muito mais do que isso, precisam de escalabilidade e controle de transações. Para atender à estes requisitos a Sun desenvolveu a Plataforma Java™ 2, Enterprise Edition (J2EE). J2EE é um padrão de classes Java que definem uma arquitetura multi-camadas que se encaixa no desenvolvimento, preparação e manutenção de aplicações de grande porte escritas em Java. J2EE possui uma completa funcionalidade uma vez que é

possível desenvolver uma enorme classe de aplicações de grande porte usando apenas as classes J2EE. A figura 4.1 ilustra a arquitetura de uma aplicação J2EE.

Clientes remotos são implementados como uma combinação de páginas HTML e applets. A camada intermediária é separada em dois, os Enterprise JavaBeans framework™(EJB) que contém os Enterprise Beans, que são unidades reusáveis que contém lógica de negócios com controle de transações, e o Web Server que contém páginas JSP e Servlets, que são entidades de software que provém serviços em resposta às requisições http. A camada de persistência pode ser implementada com qualquer banco de dados comercial.

4.3. Considerações ao uso

Java especifica um conjunto de classes e uma extensão de classes que cobrem a maioria das funcionalidades requeridas por desenvolvedores. As principais classes são:

- Tipos básicos da linguagem
- Entrada e saída de dados
- Entrada e saída de dados em redes
- Utilitários como zip e classes de segurança
- Classes para o desenvolvimento de interfaces gráficas

Performance: A performance é uma consideração importante quando da decisão de adotar Java. Na maioria dos casos, o código "interpretado" de Java é muito mais lento do que o código compilado em C ou C++(aproximadamente 10 a 15 vezes mais rápido). Entretanto as mais recentes versões dos populares browsers web e os ambientes de desenvolvimento Java provêm o compilador Just In Time(JIT) que produz código binário nativo para determinado processador, e isto favorece a briga com seu rival otimizado C++. A plataforma Java 2 também fornece o Java HotSpot™ Performance Engine que combina funcionalidade de uma JIT com otimizações em tempo de execução que aumentam ainda mais a performance de Java. Para aplicações em tempo real, as implicações que o coletor de lixo interno do Java devem ser considerados. O coletor de lixo pode tornar difícil a aplicação conseguir fornecer as respostas e resultádos no tempo hábil requerido pela aplicação.

Migrando para a linguagem Java: Alguns itens devem ser considerados ao migrar de C ou C++ para Java, incluindo os seguintes:

- Java é completamente orientada a objetos, portanto tudo precisa ser feito via invocação de métodos
- Java não possui ponteiros ou tipos definidos pelo usuário
- Java suporta multi tarefa e coletor de lixo

4.4. Maturidade

Java foi disponibilizado para o público em geral em Maio de 1995, e tem passado por uma rápida transição sem precedentes. Todos os browsers web da atualidade fornecem suporte ao Java através de Máquinas Virtuais Java como parte integrante de seus produtos. Existem diversos livros disponíveis que descrevem as funcionalidades e todos os aspectos da programação Java. O uso comercial de Java também cresceu muito rapidamente em um período curto de tempo. A Sun disponibiliza em seu site uma série de casos de sucesso de usuários que utilizaram Java em seus websites..

Existem na atualidade diversos ambientes de desenvolvimento que suportam a linguagem Java, incluindo IBM Visual Age for Java, Symantec's Visual Café, Microsoft J++ e o próprio Sun Java Development Kit (JSDK). A maioria destes produtos fornece editores integrados, debugadores, compiladores JIT, ferramentas de teste, e mais uma série de ferramentas comuns ao desenvolvimento.

Capítulo 5

5. Estudo de Caso

5.1. Visão Geral

O surgimento da Internet veio trazer uma nova forma de fazer negócios. A compra e venda de produtos, o relacionamento entre fornecedores e clientes, o auto-atendimento em bancos estão mais ágeis e rápidos na era da Internet.

Na realidade brasileira, grande parte das empresas já está aparecendo na Internet na forma de um site, com páginas que mostram quem elas são e que produtos têm para oferecer. O segundo passo é utilizar este site na Internet para vender produtos e oferecer serviços aos clientes (B2C[2]) e também para outras empresas (B2B[3]). Os bancos estão neste patamar de desenvolvimento, bem como empresas fornecedoras de peças para montadoras de automóveis, grandes supermercados, e uma grande parte do comércio.

Esta estrutura que a Internet nos proporciona, pode ser utilizada também para automatizar muitos processos dentro das empresas, diminuindo assim custos e agilizando sua produção, o atendimento ao cliente e garantindo um controle melhor dos processos.

5.2. Problemática

No escopo deste trabalho um estudo de caso será apresentado baseado na empresa Planet Cap Bordados, que atua no varejo vendendo bonés e camisetas bordadas, e oferecendo serviços de bordados em peças diversas. A empresa também oferece serviços de punching[4], e possui quatro lojas em diferentes localidades.

[2] B2C – Business-to-Consumer: Negócios entre a empresa e seus clientes.
[3] B2B – Business-to-Business: Negócios entre empresas.
[4] O termo punching é usado no ramo de bordados como o ato de transformar/programar uma logomarca em pontos de bordado

A empresa possui 200 tipos de itens em estoque somando aproximadamente 6.000 peças em um escritório central e possui estoque aproximado de 800 peças em cada uma das 4 lojas. O problema é que não há um controle automático deste estoque e todas as modificações no estoque são feitas de modo manual utilizando uma planilha do Excel.

O processo de controle de estoque é feito da seguinte maneira: A cada produto vendido, uma comanda é preenchida com seu respectivo código. No final do dia, os códigos dos produtos vendidos naquele dia são passados por email para o escritório central onde serão atualizados manualmente em uma planilha.

Neste processo, muitos erros podem acontecer: erros no momento em que foi escrito o código do produto na comanda, erros quando foram digitados os códigos dos produtos no email para o escritório no final do dia e erros no processo de atualizar a planilha no escritório no dia seguinte. Todos estes erros são passíveis de ocorrer pois todos estes processos são feitos manualmente. Um inventário periódico é feito a cada mês para apurar as diferenças, e corrigir a planilha do estoque.

A empresa não possui um cadastro de clientes, conseqüentemente não tem base de dados suficiente para analisar o perfil do seu consumidor e praticar promoções específicas. Sem o cadastro não há possibilidade de se obter as informações relacionadas a um cliente quando o mesmo retorna a uma das lojas para executar um mesmo serviço já realizado antes, economizando tempo pois as informações dos serviços realizados pelo cliente como valores cobrados, quais peças foram vendidas, já estariam armazenadas.

A empresa gostaria de entrar em contato com os clientes, ou parte deles, para praticar o pós-venda e medir o grau de satisfação do cliente, perante o serviço que lhe foi prestado. Esta prática também fica impossibilitada uma vez que as vendas e serviços não estão relacionados diretamente com o cliente.

O controle da frente de caixa das lojas também é problemático, ele é feito da seguinte forma: a cada produto vendido, o dinheiro é colocado no caixa, e após a comanda ser feita, seu código é anotado em uma lista de vendas do dia. Assim todas as comandas são anotadas durante o dia na lista de controle. No final do dia, os valores contidos na lista de controle são somados e confrontados com o valor em dinheiro encontrado no caixa. Após a verificação de

que o caixa está correto, então o processo de digitar os códigos dos produtos vendidos no dia em um email para ser enviado ao escritório é iniciado.

5.3. Soluções comerciais

As soluções comerciais que existem para a solução deste problema, geralmente tratam de uma forma genérica a maioria das funcionalidades, como: controle de estoque, controle de caixa e cadastro de clientes. Em geral soluções prontas não possuem uma característica importante que é a personalização do programa, de forma a atender a todos os requisitos diferentemente para cada empresa.

Uma outra característica de programas de controle de lojas, é que a maioria dos programas hoje são aplicações Windows© ou até mesmo aplicações mais antigas, ainda na plataforma MS-DOS©, que rodam em uma máquina local apenas, geralmente não possuindo mecanismos de integração com outras lojas e com um escritório central.

Outra importante funcionalidade procurada é o funcionamento da aplicação na Internet e a operação on-line com mais de uma loja em um mesmo banco de dados, para que os dados estejam sempre atualizados, e disponíveis entre as diferentes localidades.

5.4. Proposta para solução

A solução proposta é a utilização do modelo Cliente-Servidor multicamadas no ambiente da Internet para a solução do problema. De acordo com Komosinski [4] um sistema desse tipo de baseia em um servidor na Internet, contendo a aplicação que controla todos as funcionalidades necessárias, como os controles de estoque e caixa, e cadastro de clientes.

5.5. Metodologia

A arquitetura Cliente-Servidor consiste em um servidor central contendo a aplicação que será acessada via Internet pelas diversas máquinas das lojas que também estarão conectadas à Internet.

A aplicação ficará acessível na Internet na forma de um site e as lojas entrarão no site através de um endereço de páginas de Internet comum, e terão acesso ao sistema mediante uma senha. Após a entrada do nome de usuário e senha, a aplicação ficará disponível à loja e as transações poderão ser feitar normalmente.

A aplicação será desenvolvida utilizando o modelo MVC [11] (Model View Controller) que separa a aplicação em três componentes distintos, o modelo, a interface e o controlador.

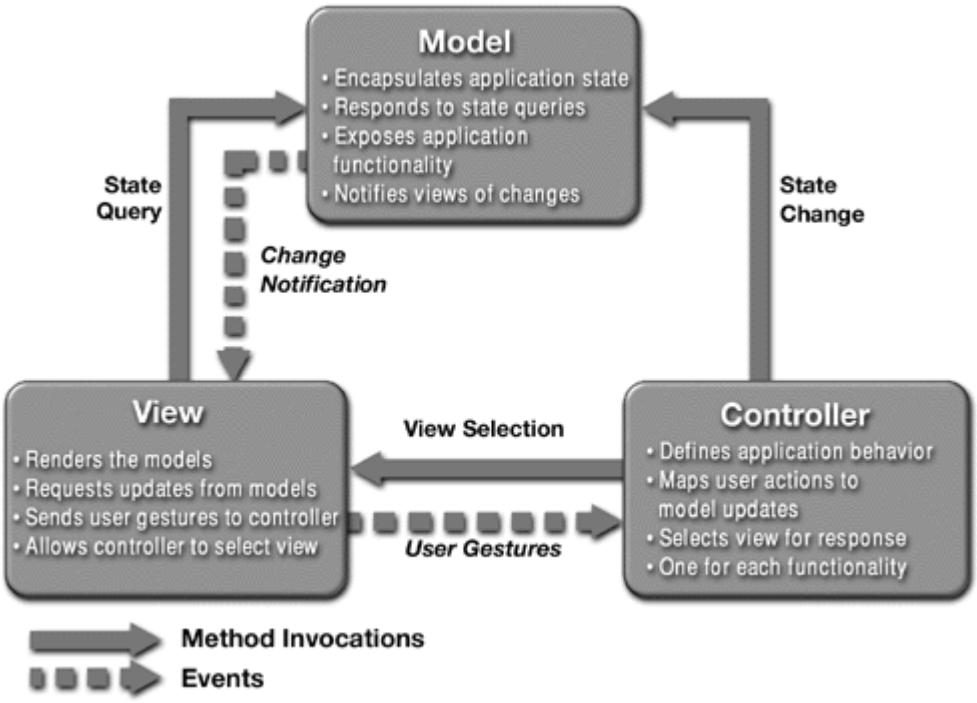


Figura 5.1: Modelo MVC [11]

Cada um destes componentes possuem responsabilidades distintas, como visto na figura 5.1:

- O modelo representa os dados e a lógica da aplicação. O modelo em geral é a aplicação em si, que transforma os dados da forma que o usuário requisitou, e possui uma interface que pode ser acessada pelo controlador.

- A interface transforma os dados do modelo. Ela acessa as informações do modelo e a mostra ao usuário da forma que ela tem q ser apresentada. A interface se modifica, quando os dados do modelo se modificam. A interface também repassa as requisições do usuário para o controlador.

- O controlador define o comportamento da aplicação. Ele despacha as requisições dos usuários e controla a apresentação dos dados. O controlador interpreta as entradas do usuário e mapeia estas entradas para o modelo, e seleciona qual será a próxima interface a ser mostrada ao usuário.

5.6. Vantagens

O modelo Cliente-Servidor traz várias vantagens neste problema:

Quando uma atualização da aplicação é feita, automaticamente as lojas estarão acessando a versão atualizada da aplicação, ao contrário de aplicações que funcionam diretamente no micro local, onde a troca da aplicação toda se faz necessária, máquina a máquina.

O estoque é atualizado diretamente no servidor e consequentemente está sempre atualizado. Qualquer alteração de preços, nomes e características dos produtos que se altere, imediatamente já estarão sendo vistas pelas lojas.

Toda a aplicação e o banco de dados está em um servidor central, isso facilita a manutenção e o backup de todas as informações.

5.7. Detalhamento do sistema

5.7.1. Ferramentas

O sistema será desenvolvido utilizando a linguagem Java, comentada anteriormente no Capítulo 4.

O servidor de aplicação Java será o Tomcat[5], servidor de páginas html e JSP[6].

O servidor de aplicação Tomcat é na verdade um framework[7] que facilita o desenvolvimento da aplicação. O Tomcat faz o papel do servidor de aplicações presente na camada intermediária das três camadas, e já está pronto, bastanto apenas desenvolvermos a aplicação que rodará nele.

Da mesma forma o cliente, no caso o browser, também já está desenvolvido. Pode ser o Internet Explorer, Netscape Navigator ou qualquer navegador que tenha capacidade de processar páginas html[8]. O protocolo de comunicação entre o servidor de aplicações e o cliente é o protocolo http, padrão na Internet para a transferências de páginas html entre servidores e browsers.

Assim como não há a necessidade de desenvolver o servidor de aplicação, responsável por tratar as requisições dos clientes e servir as páginas da aplicação, e nem precisamos desenvolver o cliente, responsável por conectar no servidor e requisitar as informações, podemos então focar os esforços diretamente no desenvolvimento da aplicação em si, sem se preocupar com detalhes da implementação do cliente ou do servidor de aplicações, tampouco com o protocolo de comunicação entre eles.

A figura 5.2, de acordo com Komisiski [4], mostra uma visão do problema de forma simplificada, separando dois módulos distintos: um módulo que trata do problema e possui as

[5] Tomcat faz parte do projeto Apache Jakarta e pode ser encontrado em http://jakarta.apache.org/tomcat/
[6] Java Server Pages, tecnologia Java para gerar páginas html dinamicamente.
[7] Conjunto de classes e aplicativos já prontos, que são utilizados como suporte ao desenvolvimento de sistemas
[8] Páginas html são as páginas que vemos na Internet. HTML é a linguagem na qual elas são escritas.

regras de negócio, e o outro módulo que faz a apresentação das informações ao usuário(Interface[9]). Os módulos estão envoltos pelo que seria a aplicação.

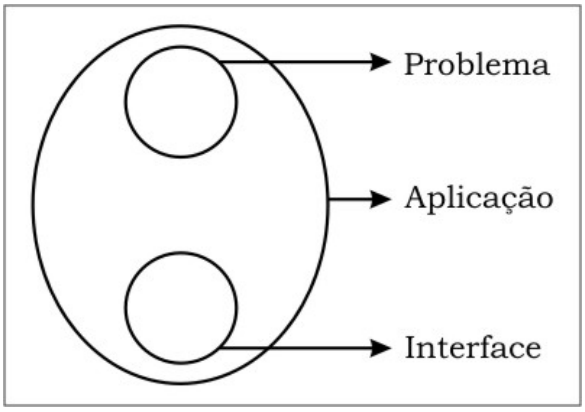


Figura 5.2 Visão do problema

A figura 5.3 mostra como é utilizado o modelo Cliente-Servidor nesta visão do problema. Nesta figura são mostradas as aplicações utilizadas como cliente e como servidor. Pode-se observar que o foco pode ser dado ao desenvolvimento da aplicação e na apresentação dos dados ao usuário(Interface).

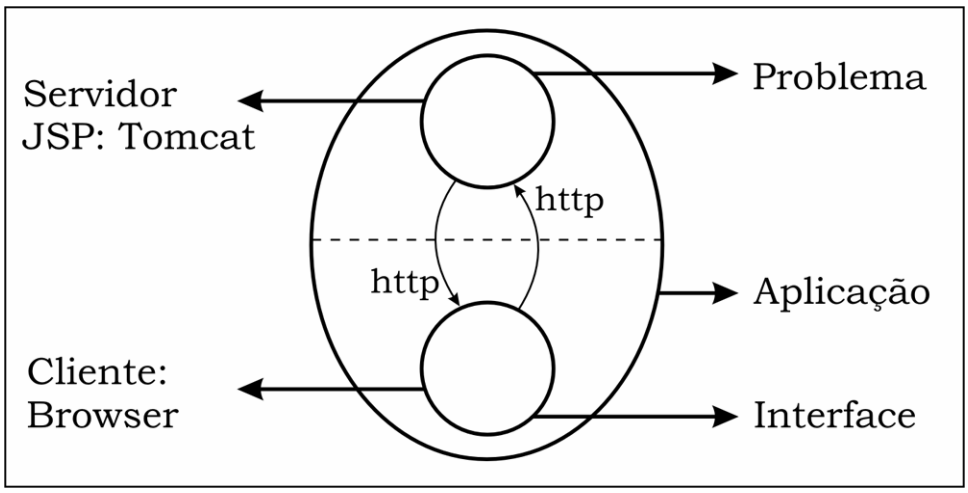


Figura 5.3 Visão do problema utilizando arquitetura Cliente-Servidor

[9] Interface é o termo utilizado para designar a parte de apresentação de um programa. É onde o usuário vê as informações e entra com os dados.

O sistema será desenvolvido utilizando a ferramenta NetBeans[10], ferramenta livre construída sobre a plataforma Java e que possui várias facilidades para a implementação de sistemas como: ferramentas de implementação, depuração e testes da aplicação.

A modelagem UML do sistema será feita utilizando a ferramenta ArgoUML, ferramenta livre construída também sobre a plataforma Java, e de acordo com Larman [5].

O banco de dados a ser utilizado é o MySQL, banco de dados de uso livre encontrado em www.mysql.com

O servidor de aplicações Tomcat e o banco de dados MySQL foram instalados sobre o sistema operacional linux Red Hat 7.2 no LAMI – Laboratório de Aprendizagem Mediada pela Informática (www.lami.inf.ufsc.br)

O protótipo da aplicação pode ser encontrado em: www.lami.inf.ufsc.br/webone

[10] Netbeans é uma ferramenta livre para desenvolvimento de aplicações Java. http://www.netbeans.org

Capítulo 6

6. Modelagem do Sistema

6.1. Casos de uso

Os casos de uso encontrados para a empresa Planet Cap Bordados é mostrado no diagrama a seguir:

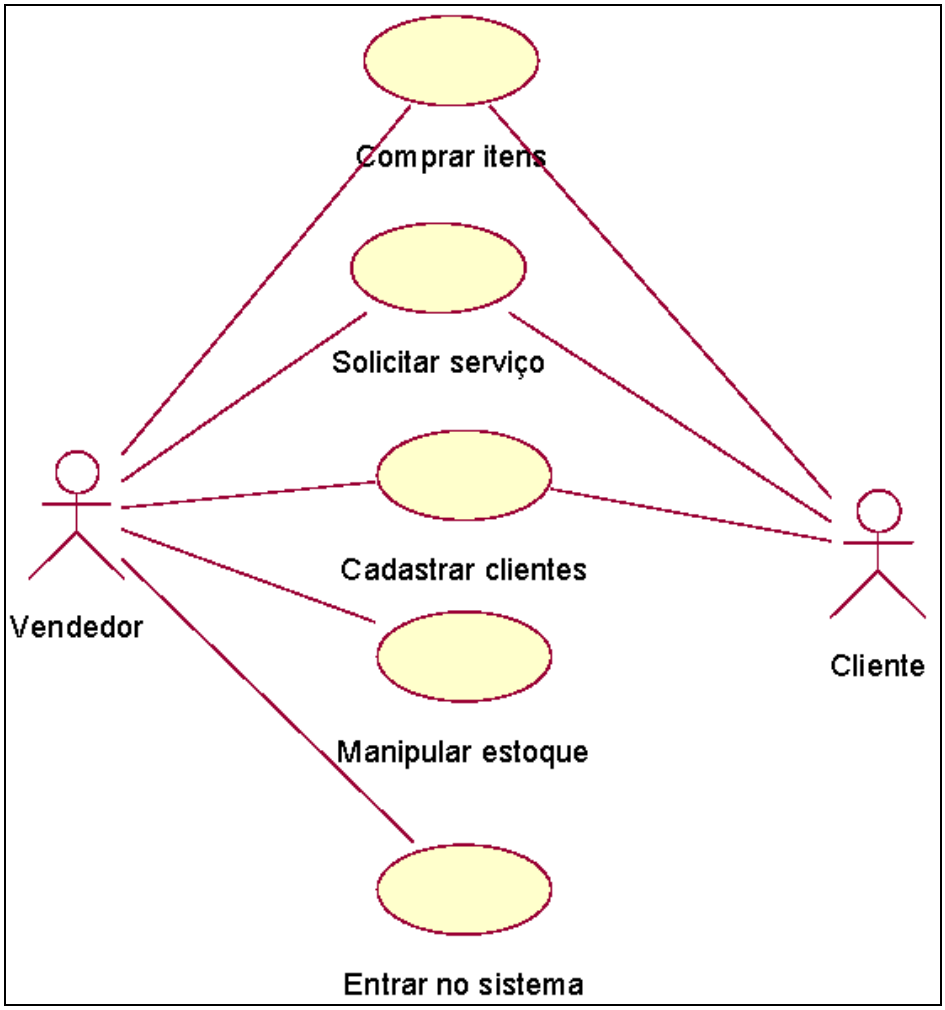


Figura 6.1 Diagrama de Casos de Uso do sistema

As tabelas a seguir mostram a concepção dos Casos de Uso e posteriormente os Casos de Uso expandido com a seqüência típica de eventos. Os Casos de Uso são sequência típicas de eventos do mundo real, e detalhes de interação com o sistema serão mostrados nos Casos de Uso expandidos.

Nome Caso de Uso:	**Comprar Itens**
Objetivo:	Comprar Itens que a loja oferece
Descrição:	O cliente chega na loja, escolhe um produto e decide por levá-lo. O vendedor então verifica o código do produto, seu valor e efetua a venda em dinheiro, cartão ou cheque.
Atores:	
Cliente	É o cliente da loja
Vendedor	É a pessoa responsável pelo atendimento ao cliente e operação do sistema.

Nome Caso de Uso:	**Solicitar Serviço**
Objetivo:	Solicitar um dos serviços que a loja oferece
Descrição:	O cliente chega na loja, e escolhe um serviço, que pode ser uma personalização em bordado em alguma peça do cliente ou a transformação da logomarca da empresa do cliente em bordado. O vendedor dá o preço do serviço em questão baseado no tipo do pedido do cliente e preenche um pedido com todas as informações do serviço prestado.
Atores:	
Cliente	É o cliente da loja interessado no serviço
Vendedor	É a pessoa responsável pelo atendimento ao cliente e operação do sistema.

Nome Caso de Uso:	**Cadastrar Cliente**
Objetivo:	Cadastrar o cliente na base de dados da loja
Descrição:	O vendedor cadastra o cliente na base de dados da loja.
Atores:	
Cliente	É o cliente da loja.
Vendedor	É a pessoa responsável pelo atendimento ao cliente e operação do sistema.

Nome Caso de Uso:	**Manipular estoque**
Objetivo:	Manutenção do estoque
Descrição:	O vendedor recebe mercadorias ou transfere mercadorias para outras lojas, atualizando no sistema a quantidade de produtos recebidas ou enviadas
Atores:	
Vendedor	É a pessoa responsável pelo atendimento ao cliente e operação do sistema.

Nome Caso de Uso:	**Entrar no sistema**
Objetivo:	Entrar no sistema
Descrição:	O vendedor entra no sistema através da Internet, e entra com o nome da loja e a senha.
Atores:	
Vendedor	É a pessoa responsável pelo atendimento ao cliente e operação do sistema.

6.2. Casos de Uso Expandido

Os casos de uso apresentados na seção anterior agora são apresentados em detalhes, com seus fluxos típicos de eventos e interações com o sistema.

Nome Caso de Uso:	**Comprar Itens**
Objetivo:	Comprar Itens que a loja oferece
Descrição:	O cliente chega na loja, escolhe um produto e decide por levá-lo. O vendedor então verifica o código do produto, seu valor e efetua a venda em dinheiro, cartão ou cheque.
Fluxo Típico	A venda ao cliente é efetuada pelo vendedor Vendedor seleciona opçao de nova venda no sistema Sistema solicita qual codigo do produto Vendedor entra com o codigo do produto e o sistema apresenta o valor do mesmo, e pede o valor do serviço, no caso se o cliente vai bordar a peça ou não. Sistema solicita o codigo do cliente seja informado caso o cliente já possua cadastro, caso contrario a opcao de cadastrar o cliente é mostrada. Sistema calcula os valores totais da venda e dá a opção de imprimir uma nota. Vendedor encerra a venda e o sistema salva a venda guardando a data e hora da venda Automaticamente o sistema já faz a baixa do produto vendido, baseado no código do mesmo.
Atores:	
Cliente	É o cliente da loja
Vendedor	É a pessoa responsável pelo atendimento ao cliente e operação do sistema.

Nome Caso de Uso:	**Solicitar Serviço**
Objetivo:	Solicitar um dos serviços que a loja oferece
Descrição:	O cliente chega na loja, e escolhe um serviço, que pode ser uma personalização em bordado em alguma peça do cliente ou a transformação da logomarca da empresa do cliente em bordado. O vendedor dá o preço do serviço em questão baseado no tipo do pedido do cliente e preenche um pedido com todas as informações do serviço prestado.
Fluxo Típico:	O cliente chega na loja, escolhe um produto e decide por bordar o produto. O cliente escolhe qual bordado será feito na peça O vendedor é capaz de informar o preço do bordado escolhido baseado no número de pontos do mesmo. Serviço é efetuado e vendedor entra com o valor do serviço e o tipo da peça que foi bordada no sistema Cliente paga o serviço.
Atores:	
Cliente	É o cliente da loja interessado no serviço
Vendedor	É a pessoa responsável pelo atendimento ao cliente e operação do sistema.

Nome Caso de Uso:	**Entrar no sistema**
Objetivo:	Entrar no sistema
Descrição:	O vendedor entra no sistema através da Internet, e entra com o nome da loja e a senha.
Fluxo Típico:	Vendedor acessa atraves da Internet, o servidor de aplicações onde está rodando a aplicação É apresentada a tela principal, e é requisitada a entrada de um nome e uma senha. Vendedor entra com o nome a senha, e o sistema processa a requisição A entrada é permitida depois da verificação do nome e da senha do usuário.
Atores:	
Vendedor	É a pessoa responsável pelo atendimento ao cliente e operação do sistema.

Nome Caso de Uso:	**Manipular estoque**
Objetivo:	Manutenção do estoque
Descrição:	O vendedor recebe mercadorias ou transfere mercadorias para outras lojas, atualizando no sistema a quantidade de produtos recebidas ou enviadas
Fluxo Típico:	O escritório central separa mercadorias para enviar as lojas O escritório anota quais foram as mercadorias separadas e enviadas, para uma posterior conferência. O escritório faz a modificação no estoque em uma planilha do Excel. A loja recebe as mercadorias e envia um email com os códigos e quantidades recebidas para o escritorio central O escritorio recebe o email, confere com o que foi anotado Caso haja diferença na transferência de mercadorias, é feita a correção na planilha.
Atores:	
Vendedor	É a pessoa responsável pelo atendimento ao cliente e operação do sistema.

Capítulo 7

7. A aplicação

7.1. Estrutura

A aplicação foi montada baseada no modelo MVC [11] (model view controller), ou seja, existe um controlador que controla as entradas do usuário e baseada nessas entradas chama as funções que existem no modelo. Pode se ter uma idéia do funcionamento pela figura 7.1:

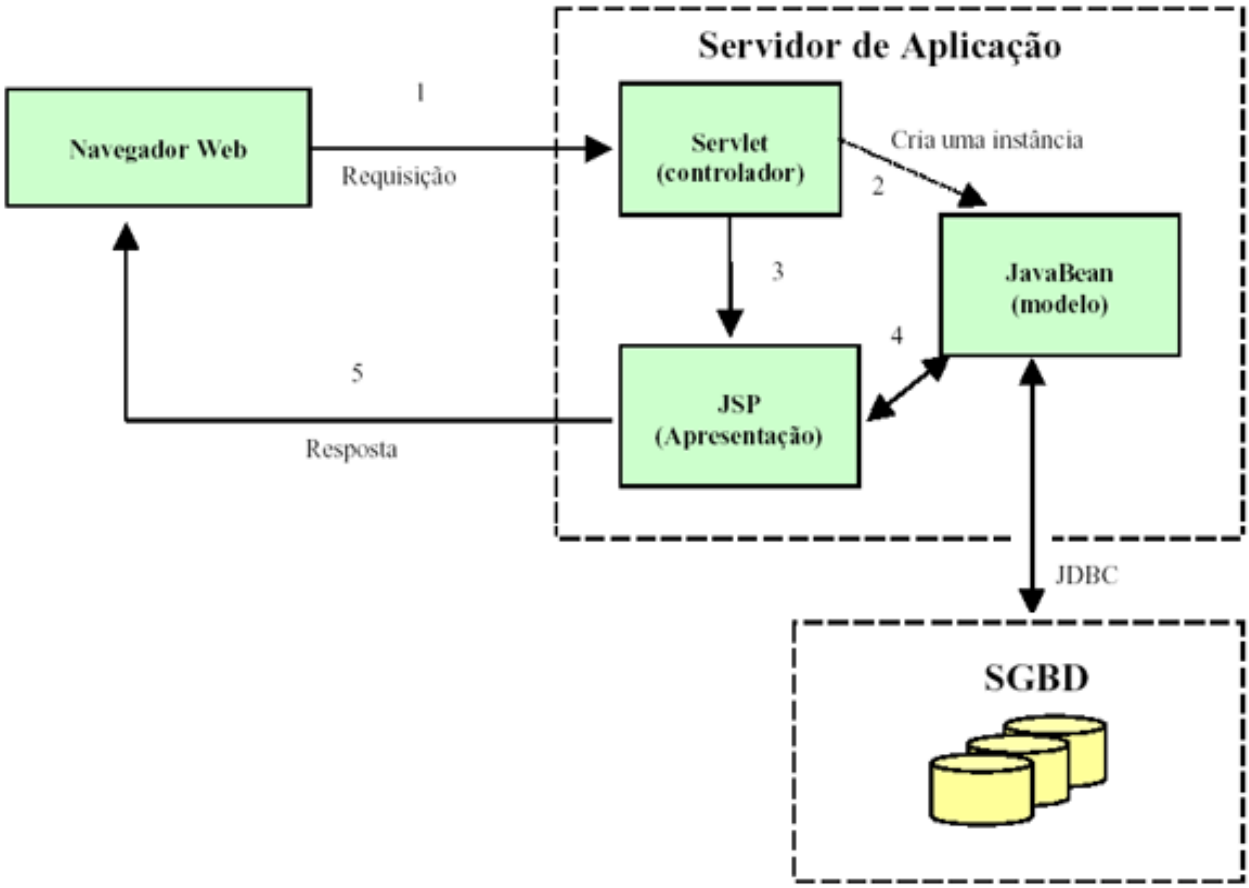


Figura 7.1: A arquitetura da aplicação. [11]

O servidor de aplicações foi o Tomcat e o cliente web pode ser qualquer browser web existente. O canco de dados utilizado foi o MySQL.

7.2. Estrutura de classes

A classe Controlador.java é um JavaBean que faz o papel do controlador, ele recebe as requisições enviadas pelo browser do cliente e atua na Aplicação.java, onde estão implementadas as interações com o banco de dados e o fluxo normal do sistema.

Existem ainda classes que modelam a Venda, Item de Venda, Cliente e Transferencia de produtos do estoque. Estas classes encapsulam os objetos e fornecem interfaces públicas para a manutenção de seus dados.

Como a aplicação foi inteiramente desenvolvida utilizando o paradigma da orientação a objetos, então todas as requisições, e serviços são baseados em objetos. Desta forma quando cadastramos um cliente, uma instância da classe Cliente é enviada à Aplicação, e dentro da classe aplicação o cliente então é armazenado no banco de dados.

Esta forma de implementação torna o sistema mais genérico, pois posso alterar os atributos da classe cliente, e a forma como suas funções são programadas, e a forma como a aplicação armazenará o cliente será a mesma, pois estou passando o objeto Cliente.

Esta técnica de programação é mais complexa e mais demorada mas provê uma reusabilidade de componentes maior, uma vez que esta classe Cliente pode ser utilizada em qualquer outra aplicação.

7.3. Layout da aplicação

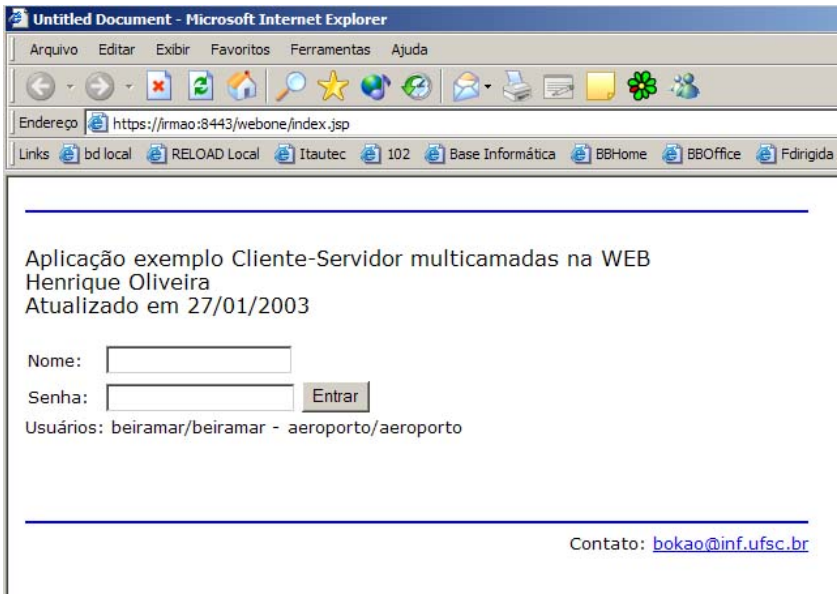


Figura 7.2: Tela inicial de entrada

O usuário entra com o nome e a senha, e esta requisição é enviada então ao Controlador, que chama a função de validação de login na Aplicação.

A aplicação então verifica no banco de dados se o login está correto e libera ou não o acesso às páginas internas.

As opções do sistema são mostradas na tela seguinte. É possível efetuar uma venda, cadastrar um cliente, fazer movimentações com o estoque e tirar alguns relatórios. A figura 7.3 mostra as opções do sistema.

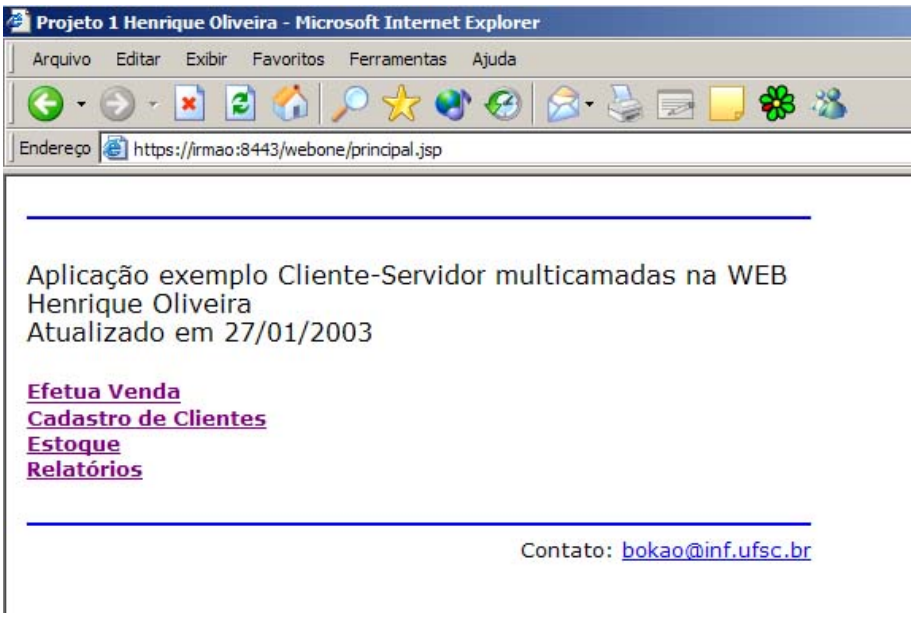


Figura 7.3: Opções do sistema

A figura 7.4 nos mostra a tela onde as vendas são efetuadas. O procedimento da venda foi implementado na forma de um "carrinho de supermercado", uma forma tradicional na Internet que consiste em adicionar itens ao carrinho, e no final da venda, os valores dos produtos que estão no carrinho são somadas e é informado o valor total.

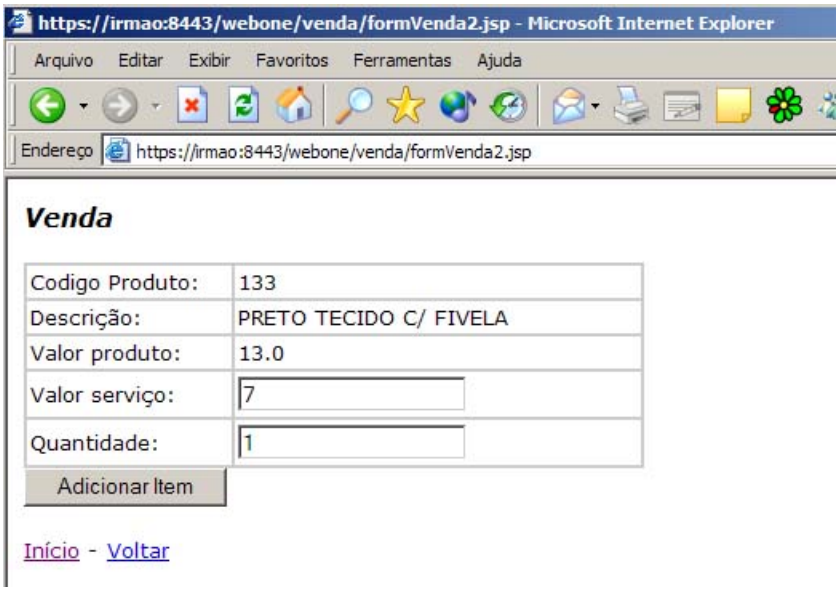


Figura 7.4: Efetuando a venda de produtos

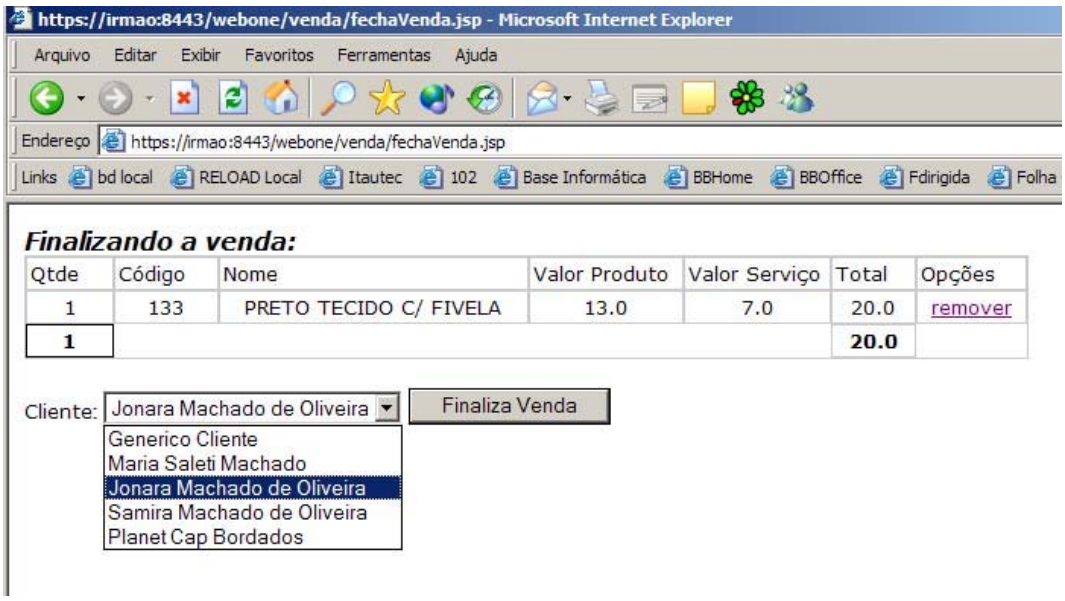


Figura 7.5: Finalizando a venda

Tem-se a opção de adicionar mais itens à venda atual, e ao concluir a venda, a opção de vincular um cliente à venda é solicitada, como mostra a figura 7.5.

Na opção de cadastro de clientes, pode-se listar todos os clientes, pesquisar os clientes por qualquer campo de seu cadastro e adicionar um novo cliente. A figura 7.6 mostra as opções da seção Cadastro de Clientes.

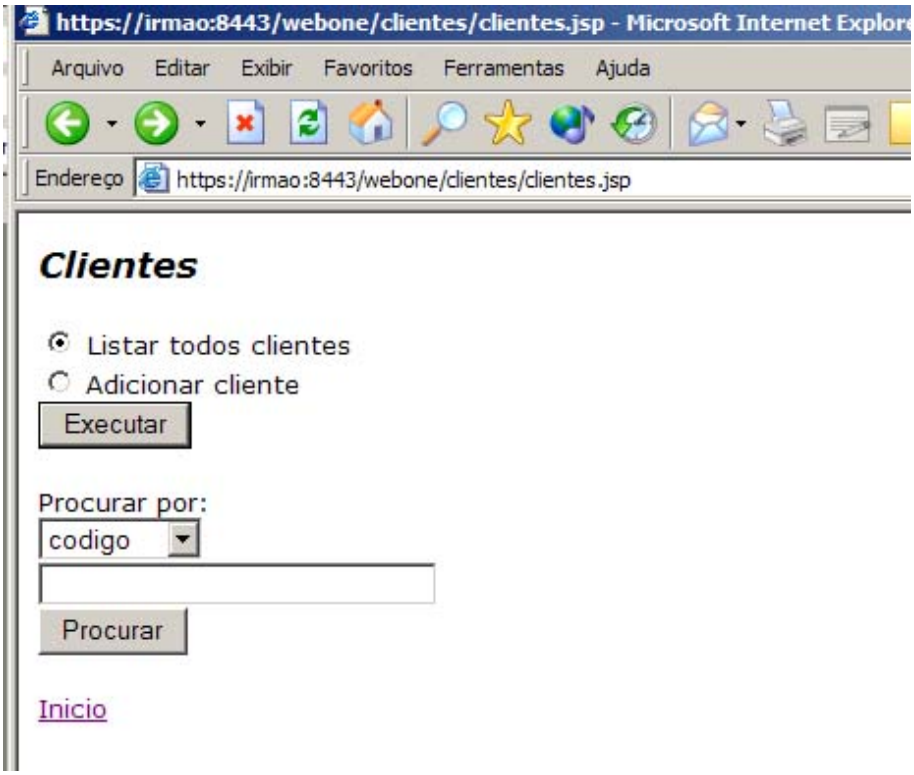


Figura 7.6: Opções do Cadastro de Clientes

Na listagem de clientes, pode-se automaticamente adicionar, remover ou ver mais informações sobre respectivo cliente, bastanto clicar na opção ao lado do nome, como vemos na figura 7.7.

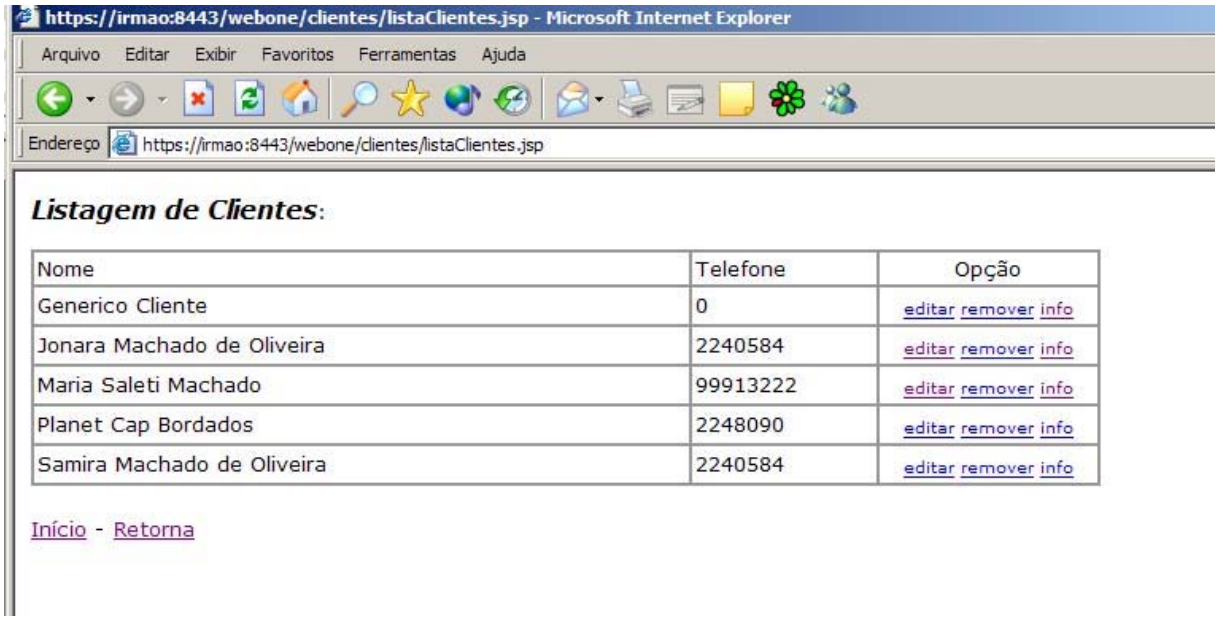


Figura 7.7: Listagem de clientes.

Ao clicar sobre a opção *info,* as informações detalhadas do cliente é mostrada. A figura 7.8 exemplifica:

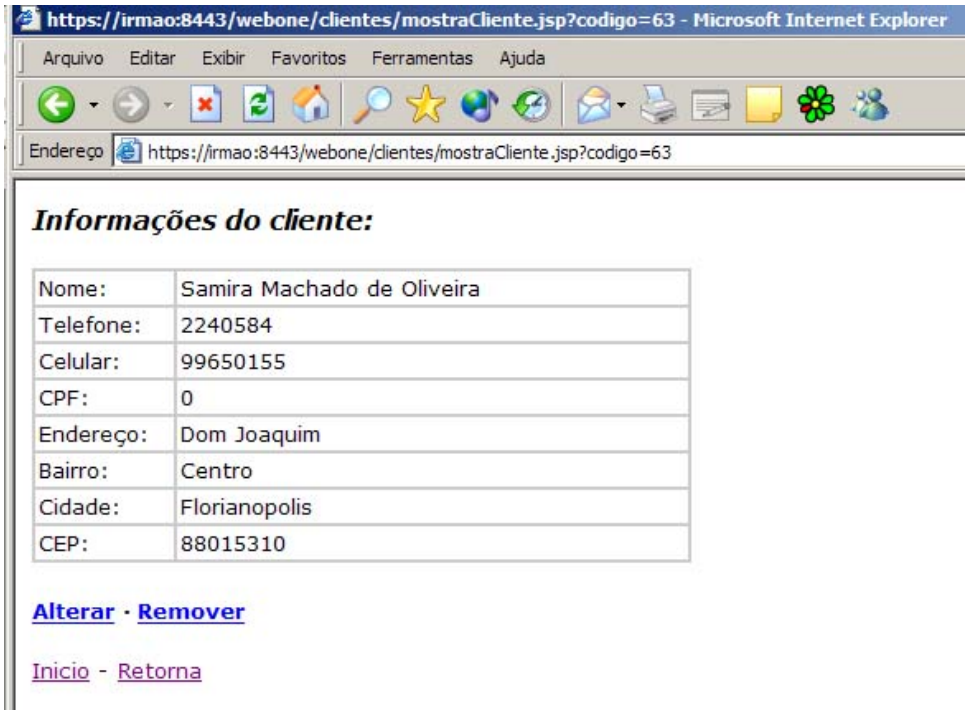
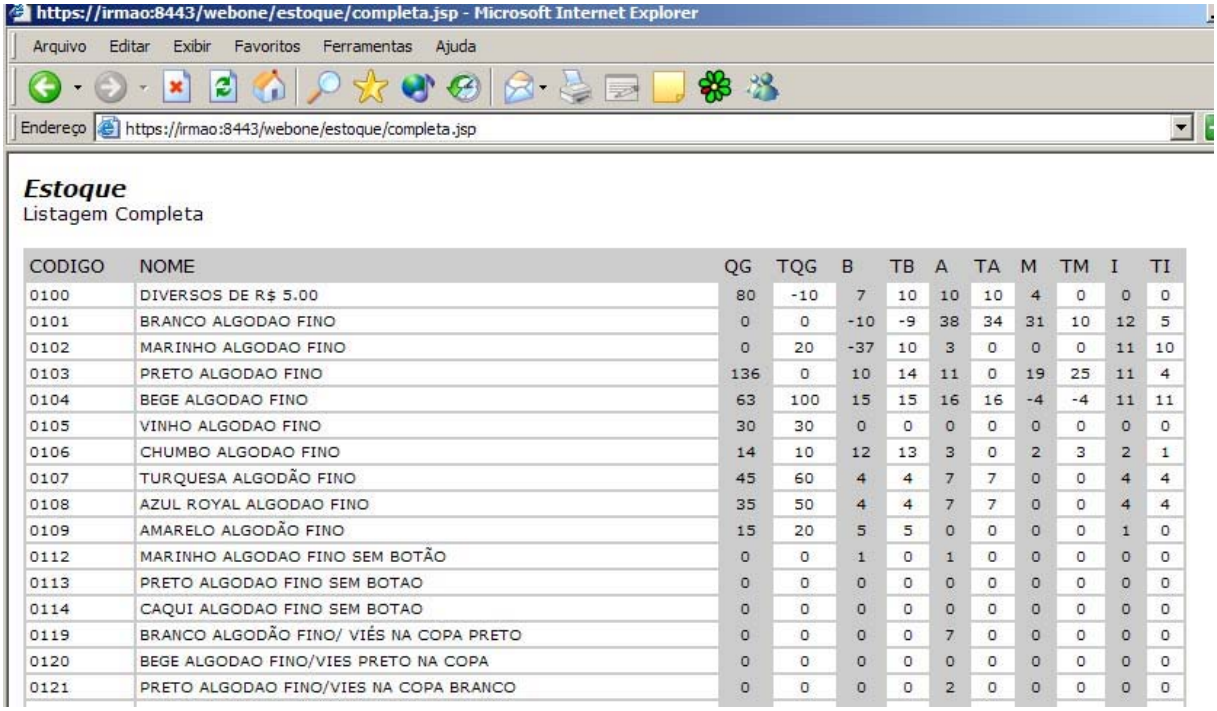


Figura 7.8: Informações detalhadas do cliente.

Na opção de movimentação de estoques, a opção de listagem completa retorna ao usuário toda a listagem dos produtos, com código, descrição do mesmo e a quantidade presente em cada localidade, como visto na figura 7.9:



CODIGO	NOME	QG	TQG	B	TB	A	TA	M	TM	I	TI
0100	DIVERSOS DE R$ 5.00	80	-10	7	10	10	10	4	0	0	0
0101	BRANCO ALGODAO FINO	0	0	-10	-9	38	34	31	10	12	5
0102	MARINHO ALGODAO FINO	0	20	-37	10	3	0	0	0	11	10
0103	PRETO ALGODAO FINO	136	0	10	14	11	0	19	25	11	4
0104	BEGE ALGODAO FINO	63	100	15	15	16	16	-4	-4	11	11
0105	VINHO ALGODAO FINO	30	30	0	0	0	0	0	0	0	0
0106	CHUMBO ALGODAO FINO	14	10	12	13	3	0	2	3	2	1
0107	TURQUESA ALGODÃO FINO	45	60	4	4	7	7	0	0	4	4
0108	AZUL ROYAL ALGODAO FINO	35	50	4	4	7	7	0	0	4	4
0109	AMARELO ALGODÃO FINO	15	20	5	5	0	0	0	0	1	0
0112	MARINHO ALGODAO FINO SEM BOTÃO	0	0	1	0	1	0	0	0	0	0
0113	PRETO ALGODAO FINO SEM BOTAO	0	0	0	0	0	0	0	0	0	0
0114	CAQUI ALGODAO FINO SEM BOTAO	0	0	0	0	0	0	0	0	0	0
0119	BRANCO ALGODÃO FINO/ VIÉS NA COPA PRETO	0	0	0	0	7	0	0	0	0	0
0120	BEGE ALGODAO FINO/VIES PRETO NA COPA	0	0	0	0	0	0	0	0	0	0
0121	PRETO ALGODAO FINO/VIES NA COPA BRANCO	0	0	0	0	2	0	0	0	0	0

Figura 7.9: Listagem completa do estoque

As tranferências entre lojas, e do escritório central para as lojas é mostrado na figura 7.10, onde vários produtos estão sendo transferidos de uma loja para outra.

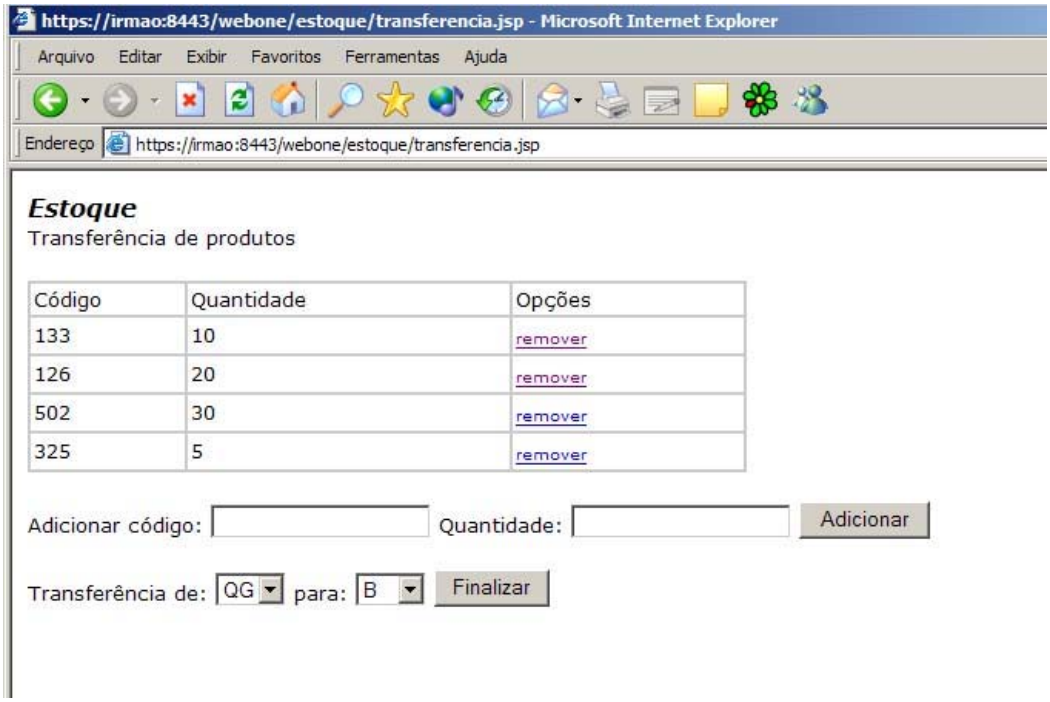


Figura 7.10: Produtos sendo transferidos de uma localidade para outra.

Finalmente na seção de relatórios, os usuários podem tirar relatórios de vendas separados pro loja ou por produto, e ainda selecionar o mês desejado do relatório, a figura 7.11 mostra um relatório de uma das lojas. O relátorio contém o código do produto vendido, valores praticados, código do cliente, data e quantidade de itens daquela venda.

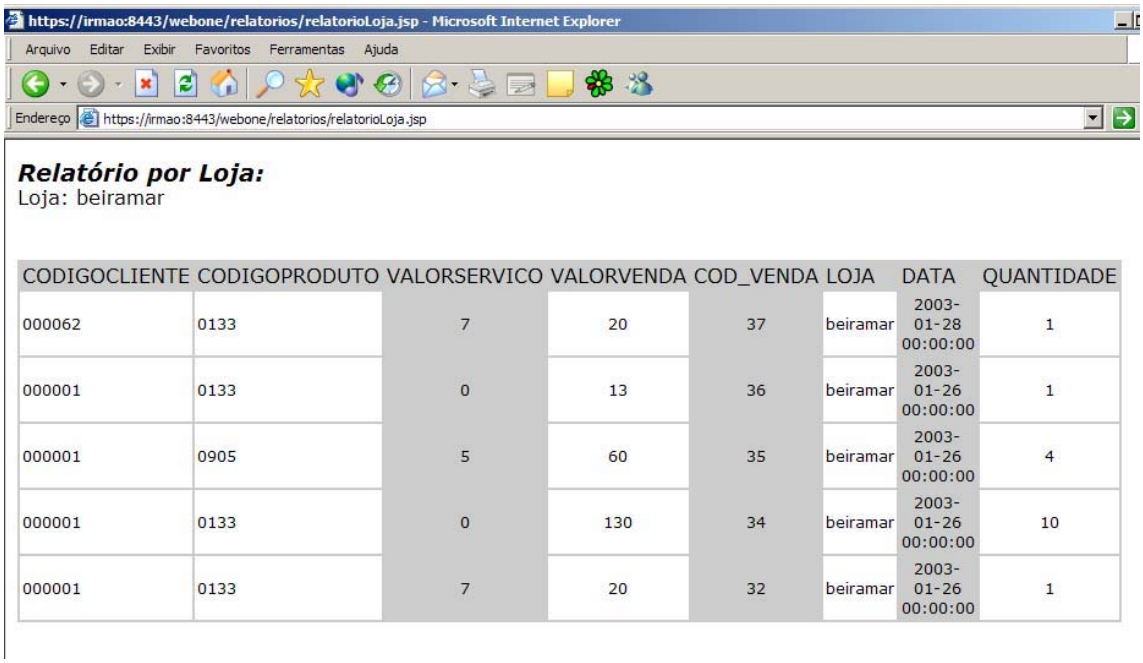


Figura 7.11: Relatório de vendas por loja

7.4. Considerações sobre o sistema

O sistema já está em funcionamento no escritório central, e está sendo utilizado na manutenção do estoque. Claramente o processo de atualização de estoque, movimentação de produtos entre lojas ficou mais simples e rápido de se executar, diminuindo as margens de erro que uma simples planilha do Excel causava.

O sistema de vendas, onde as lojas estarão efetuando suas vendas on-line depende apenas da infraestrutura de rede de cada loja, com serviços de internet a cabo, como ADSL entrar em funcionamento. Assim que estiverem conectadas on-line à Internet, o sistema de vendas já estará sendo modificado para ser executado diretamente na Internet.

A expectativa do sistema é facilitar, e agilizar os processos, e isto vem acontecendo. Diversos módulos ainda precisam ser desenvolvidos, como controle contábil, e controle de recursos humanos, mas ao que estre trabalho se propôs a fazer, que era automatizar os processos de venda e estoque foram cumpridos com êxito.

O processo de implantação em cada loja é de aproximadamente 10 dias, e o sistema é colocado em funcionamento em paralelo com o sistema atual, para fins de verificação da confiabilidade do sistema.

Capítulo 8

8. Considerações finais

8.1. Dificuldades encontradas

Uma das maiores dificuldades encontradas foi o fato do sistema ter sido totalmente construído sobre a tecnologia Java, e a pessoa que aqui lhes escreve não conhecia a linguagem.

Me propus este desafio para aprender uma nova tecnologia, e novas arquiteturas de desenvolvimento de sistemas não vistas por mim durante a graduação, e tomei um bom tempo do projeto apenas para dominar a tecnologia e suas ferramentas de desenvolvimento.

Outra dificuldade foi na questão da modelagem, dúvidas sobre a responsabilidade de cada classe, como exemplo: “a quem pertence a responsabilidade de atualizar o cliente no banco de dados? pertence à aplicação pois ela conhece o cliente? Ou pertence ao próprio cliente, pois só ele melhor conhece seus atributos, assim passaríamos a conexão do banco de dados ao cliente, e ele mesmo se armazenaria”. Então questões de modelagem das classes, procurando modelar de uma forma que facilitasse a reusabilidade e a flexibilidade de alterações tomaram um bom tempo.

E finalmente a questão da persistência dos objetos no banco de dados foi um caso um pouco complicado, pois o mapeamento dos objetos em tabelas relacionais a princípio não é uma tarefa corriqueira. Objetos tiveram que ser divididos em uma ou duas tabelas para serem armazenados.

8.2. Trabalhos futuros

Desenvolvimento de um sistema que permita transações automática entre a loja e o fornecedor, através da web. Tornando a reposição do estoque da empresa automático por exemplo. À medida que as peças em estoque vão baixando, e ao chegar em um nível limite, automaticamente a aplicação já se conectaria ao fornecedor e solicitaria os produtos. A

tecnologia utilizada e que está surgindo com futuro promissor para a troca desse tipo de informações através da internet são os Webservices, utilizando XML, e acredito que cabe um bom trabalho sobre isto.

Desenvolvimento do módulo financeiro, para acompanhamento da saúde financeira da empresa, e integração com o módulo de estoque e clientes já existentes.

8.3. Conclusão final

O modelo de processamento cliente/servidor não é o final da evolução do ambiente de computação e a solução de todos os problemas, mas existem vários motivos para utilizá-lo, sendo especialmente adequado para ambientes de sistemas distribuídos na Internet.

Aplicações cliente/servidor podem ser desenvolvidas utilizando-se diversas técnicas de programação. O trabalho apresentou de forma sucinta as diversas formas de organização das camadas que constituem as aplicações cliente/servidor.

O objetivo principal deste trabalho é explorar as diversas técnicas de desenvolvimento de aplicações servidoras mostrando suas principais vantagens e empregar estas técnicas diretamente na resolução de um problema, no caso a implantação de um sistema de controle de estoque de uma rede de lojas.

A implementação utilizou do modelo Model-View-Controler, que consiste em separar na aplicação, os módulos responsáveis pelo modelo, pela apresentação e pelo controle de fluxo das informações, e tem se tornado um padrão nas aplicações para Internet pelos benefícios que traz ao desenvolvimento.

O sistema desenvolvido atende aos requisitos propostos e já está em funcionamento, e os objetivos de aprendizado de novas tecnologias para desenvolvimento na Internet utilizando Java foram alcançados.

Bibliografia

[1] COULOURIS, George ; DOLLIMORE, Jean ; KINDBERG, Tim,***Distributed Systems Concepts and Design***, Addison Wesley, 1994

[2] DEITEL, H. M. & DEITEL, P.J. ***Java, Como programar***. trad. Edson Furmankiewicz. Porto Alegre: Bookman, 2001.

[3] FOWLKES, Edward; SHIDA, Takashi, ***Multithreaded Programming***, 1996

[4] KOMOSINSKI, Leandro J. ***Aplicações Cliente-Servidor via Web usando Java***. Florianópolis, 2002.

[5] LARMAN, Craig. ***Utilizando UML e Padrões, Uma introdução à Análise e ao Projeto Orientado a Objetos***. trad. Luiz A. Meirelles Salgado. Porto Alegre: Bookman, 2000.

[6] PIMENTEL, Walter M. ***Aplicações Cliente/Servidor em Múltiplas Camadas***, 1998

[7] PIRES, Paulo de Figueiredo, ***Arquitetura Cliente/Servidor – Uma Chave para o Projeto de Sistemas Distribuídos***

[8] SADOSKI, Darleen. ***Client/Server Architectures – An Overview***. http://www.sei.cmu.edu/str/descriptions/clientserver_body.html, January 1997.

[9] SILVA, ***Francisco José da Silva, Ambiente de Apoio ao Desenvolvimento de Aplicações Distribuídas***, UFMA, 1997

[10] SUN, JavaTM 2 Platform, ***Enterprise Edition (J2EE) Overview*** http://java.sun.com/j2ee/overview2.html

[11] SUN, ***J2EE Architecture Approaches, Model-View-Controller Architecture*** http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/app-arch/app-arch2.html#1105854

[12] TANEMBAUM, Andrew S.; ***Sistemas Operacionais Modernos***,1992

[13] VINOSKI, Steve; SCHMIDT, Douglas,***Comparing Alternative Client-side Distributed Programming Techniques*** (Column 3), 1995

Anexo 1

Artigo sobre trabalho

Aplicativo Cliente Servidor Multicamadas para controle de uma rede de lojas via WEB utilizando Java

Henrique Eduardo M. de Oliveira[1]

[1,] Ciências da Computação, 2003
Departamento de Informática e Estatística – INE
Universidade Federal de Santa Catarina (UFSC), Brasil, 88040-970
Fone: (048) 231-9739 Fax: (048) 231-9770
bokao@inf.ufsc.br

Resumo:

Este trabalho apresenta o modelo de desenvolvimento Cliente-Servidor multicamadas para aplicações na Internet utilizando a tecnologia Java. Este estudo complementa a formação acadêmica do Curso de Ciência da Computação agregando um novo modelo de implementação de sistemas às já conhecidas.

Esta abordagem requer um estudo profundo da Linguagem Java e de todas as ferramentas necessárias à modelagem e desenvolvimento de sistemas.

Propõe-se uma aplicação utilizando o modelo Cliente-Servidor para a automatização de uma empresa de porte médio, com algumas filiais. A aplicação visa interligar a empresa e aperfeiçoar os processos de controle de estoque e cadastro de clientes à distância via Internet.

Palavras chave: *Automação Empresarial, Administração à distância, Java, Javabeans*

Abstract:

This work studies the multi layer Client-Server model for Internet applications using the Java Technology. This study complements the Computer Science academic knowledgement adding a new model of system implementation to the ones already known.

This approach requires a deep study of the Java Language and all of the tools necessary for system modeling and system development.

A Client-Server model application is suggested to automate a medium-size company and its branches. The application's goal is to connect the company internally and to improve some stock control process and client register.

Keywords: *Enterprise Automation, Long Distance Management, Java, Javabeans*

Introdução

O modelo de aplicação Cliente-Servidor se encaixa perfeitamente em redes de computadores, onde um Servidor é responsável em prover serviços para outras máquinas da rede. Extrapolando este sistema para a Internet, o que temos é uma enorme rede mundial, interligando cidades e países, com grande potencial de desenvolvimento de serviços que podem ser utilizados por diversas máquinas em qualquer ponto do globo.

No âmbito empresarial, a corrida pela Internet no Brasil vem crescendo de forma rápida. As empresas estão se atualizando para acompanhar as mudanças e participar deste fenômeno que é a Internet. O primeiro passo para as empresas foi simplesmente aparecer na Internet, geralmente na forma de um site[11], apenas como uma vitrine virtual. O segundo passo foi oferecer serviços a clientes através do site na Internet. A compra e venda de produtos, e o atendimento bancário via Internet são algumas das inúmeras facilidades oferecidas aos clientes nos dias de hoje. O terceiro passo é utilizar toda essa infra-estrutura chamada Internet, que conecta internamente as empresas e também as conecta com fornecedores e clientes, para automatizar e agilizar os processos de controle da empresa, transações entre filiais, e relacionamentos com fornecedores e clientes.

Dentro do panorama apresentado,

[11] Conjunto de páginas acessíveis via navegador, hospedadas em um servidor na Internet.

surge a necessidade de estudar o modelo de aplicações Cliente-Servidor, pois o mesmo melhor se adapta à realidade do mercado hoje na Internet. Uma visão mais aprofundada do que é visto no Curso de Ciência da Computação sobre esta modelagem vem complementar a ampla formação já recebida durante todo o curso

Arquitetura Cliente/Servidor duas camadas

Com a arquitetura cliente/servidor 2 camadas [8], a interface do sistema é usualmente colocada na máquina do cliente e os serviços de banco de dados são usualmente colocados em um servidor que em geral é uma máquina mais potente, capaz de servir vários clientes.

O processamento é separado entre o cliente e o servidor de banco de dados. O servidor de banco da dados possui stored procedures(funções armazenadas) e triggers (gatilhos, que disparadores de funções). Existem uma variedade de produtos comerciais que fornecem ferramentas para simplificar o desenvolvimento de aplicações para a arquitetura cliente/servidor 2 camadas.

A arquitetura cliente/servidor com 2 camadas é uma boa solução para a computação distribuída quando os grupos de trabalho são na ordem de dezenas ou centenas de pessoas interagindo no sistema simultaneamente. Certamente há algumas limitações. Quando o número de usuários excede a casa das centenas, a performance tende a deteriorar. Esta limitação é resultado do servidor mantendo a conexão "viva" com cada cliente, mesmo quando nenhum trabalho está sendo feito. Uma segunda limitação da arquitetura em 2 camadas é que a implementação de serviços utilizando funções de um determinado fabricande de banco de dados tente a restringir a flexibilidade. Finalmente, as implementações de arquiteturas 2 camadas fornecem uma limitada flexibilidade em mover ou reparticionar funcionalidades do sistema de um servidor para outro, sem manualmente re-gerar todo o código da aplicação.

- **Considerações**

A arquitetura de software em duas camadas é bastante usada em processamentos não críticos e onde a manutenção e o processamento do sistema não são complexos. Este design é usado frequentemente ondea quantidade de transações entre cliente e servidor é pequena. A arquitetura 2 camadas funciona bem para um ambiente em que as regras de negócio não mudem muito frequentemente, e quando o grupo de

usuários esperado é de no máximo 100, como por exemplo em uma empresa de pequeno porte.

Arquitetura Cliente/Servidor três camadas

Ainda de acordo com Sadoski [8] , a arquitetura cliente/servidor em três camadas, também conhecida como arquitetura cliente/servidor multicamadas, surgiu para suprir as limitações que da arquitetura em duas camadas. Na arquitetura em três camadas, uma camada intermediária foi adicionada entre a camada cliente e o servidor. Existem uma variedade de formas diferentes de se implementar esta camada intermediária, como servidores de mensagens, servidores de aplicação e monitores de processamento de transições. A camada intermediária pode armazenar requisições de clientes em uma fila, então o cliente pode requisitar seu pedido à camada intermediária e desconectar, pois a camada intermediária vai acessar o banco de dados e retornar a resposta ao cliente posteriormente. A camada intermediária também fornece serviços diferenciados dependendo da prioridade do trabalho, e agendamento de tarefas. A arquitetura em três camadas tem mostrado uma melhora de performance para grandes grupos de usuários, na casa dos milhares, e possui uma flexibilidade maior se comparado com a arquitetura duas camadas. Flexibilidade pode ser bem simples a ponto de particionar código da aplicação em módulos e colocá-los em diferentes computadores se tornar uma simples operação de “arrastar e soltar” com o mouse. Uma limitação da arquitetura em três camadas é que o desenvolvimento de aplicações neste modelo é mais difícil do que o desenvolvimento em duas camadas.

- **Considerações**

Construir uma aplicação em três camadas é um trabalho complexo. Ferramentas de desenvolvimento que suportam a arquitetura três camadas ainda não fornecem todas os serviços desejados para suporte ao desenvolvimento de aplicações distribuídas.

Um problema que pode surgir é que a separação da interface com o usuário, o processamento lógico da aplicação e a lógica de banco de dados nem sempre é tão óbvia. Alguns processos lógicos podem aparecem em todas as três camadas. A colocação de uma função em uma determinada camada pode ser feita baseada em alguns critérios como os seguintes:

- Facilidade de desenvolvimento e teste

- Facilidade na administração do serviço
- Escalabilidade dos servidores
- Performance, levando em conta processamento e tráfego na rede.

Estudo de caso

No escopo deste trabalho um estudo de caso será apresentado baseado na empresa Planet Cap Bordados, que atua no varejo vendendo bonés e camisetas bordadas, e oferecendo serviços de bordados em peças diversas. A empresa também oferece serviços de punching[12], e possui quatro lojas em diferentes localidades.

A empresa possui 200 tipos de itens em estoque somando aproximadamente 6.000 peças em um escritório central e possui estoque aproximado de 800 peças em cada uma das 4 lojas. O problema é que não há um controle automático deste estoque e todas as modificações no estoque são feitas de modo manual utilizando uma planilha do Excel.

O processo de controle de estoque é feito da seguinte maneira: A cada produto vendido, uma comanda é preenchida com seu respectivo código. No final do dia, os códigos dos produtos vendidos naquele dia são passados por email para o escritório central onde serão atualizados manualmente em uma planilha.

Neste processo, muitos erros podem acontecer: erros no momento em que foi escrito o código do produto na comanda, erros quando foram digitados os códigos dos produtos no email para o escritório no final do dia e erros no processo de atualizar a planilha no escritório no dia seguinte. Todos estes erros são passíveis de ocorrer pois todos estes processos são feitos manualmente. Um inventário periódico é feito a cada mês para apurar as diferenças, e corrigir a planilha do estoque.

A empresa não possui um cadastro de clientes, conseqüentemente não tem base de dados suficiente para analisar o perfil do seu consumidor e praticar promoções específicas. Sem o cadastro não há possibilidade de se obter as informações relacionadas a um cliente quando o mesmo retorna a uma das lojas para executar um mesmo serviço já realizado antes, economizando tempo pois as informações dos serviços realizados pelo cliente como valores cobrados, quais peças foram vendidas, já estariam armazenadas.

[12] O termo punching é usado no ramo de bordados como o ato de transformar/programar uma logomarca em pontos de bordado

A empresa gostaria de entrar em contato com os clientes, ou parte deles, para praticar o pós-venda e medir o grau de satisfação do cliente, perante o serviço que lhe foi prestado. Esta prática também fica impossibilitada uma vez que as vendas e serviços não estão relacionados diretamente com o cliente.

O controle da frente de caixa das lojas também é problemático, ele é feito da seguinte forma: a cada produto vendido, o dinheiro é colocado no caixa, e após a comanda ser feita, seu código é anotado em uma lista de vendas do dia. Assim todas as comandas são anotadas durante o dia na lista de controle. No final do dia, os valores contidos na lista de controle são somados e confrontados com o valor em dinheiro encontrado no caixa. Após a verificação de que o caixa está correto, então o processo de digitar os códigos dos produtos vendidos no dia em um email para ser enviado ao escritório é iniciado.

Solução Proposta

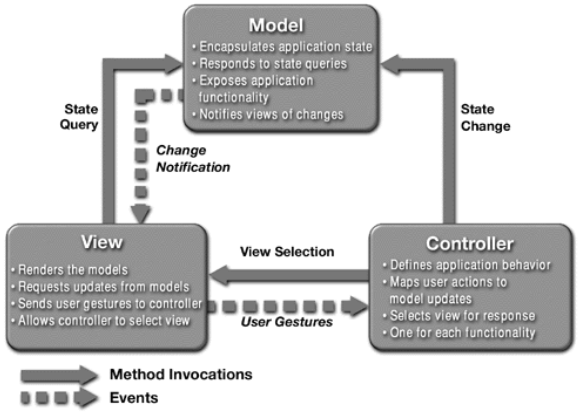
A solução proposta é a utilização do modelo Cliente-Servidor multicamadas no ambiente da Internet para a solução do problema. De acordo com Komosinski [4] um sistema desse tipo de baseia em um servidor na Internet, contendo a aplicação que controla todos as funcionalidades necessárias, como os controles de estoque e caixa, e cadastro de clientes.

Metodologia

A arquitetura Cliente-Servidor consiste em um servidor central contendo a aplicação que será acessada via Internet pelas diversas máquinas das lojas que também estarão conectadas à Internet.

A aplicação ficará acessível na Internet na forma de um site e as lojas entrarão no site através de um endereço de páginas de Internet comum, e terão acesso ao sistema mediante uma senha. Após a entrada do nome de usuário e senha, a aplicação ficará disponível à loja e as transações poderão ser feitar normalmente.

A aplicação será desenvolvida utilizando o modelo MVC [11] (Model View Controller) que separa a aplicação em três componentes distintos, o modelo, a interface e o controlador.



Cada um destes componentes possuem responsabilidades distintas, como visto na figura 5.1:

- O modelo representa os dados e a lógica da aplicação. O modelo em geral é a aplicação em si, que transforma os dados da forma que o usuário requisitou, e possui uma interface que pode ser acessada pelo controlador.

- A interface transforma os dados do modelo. Ela acessa as informações do modelo e a mostra ao usuário da forma que ela tem q ser apresentada. A interface se modifica, quando os dados do modelo se modificam. A interface também repassa as requisições do usuário para o controlador.

- O controlador define o comportamento da aplicação. Ele despacha as requisições dos usuários e controla a apresentação dos dados. O controlador interpreta as entradas do usuário e mapeia estas entradas para o modelo, e seleciona qual será a próxima interface a ser mostrada ao usuário.

Vantagens

O modelo Cliente-Servidor traz várias vantagens neste problema:

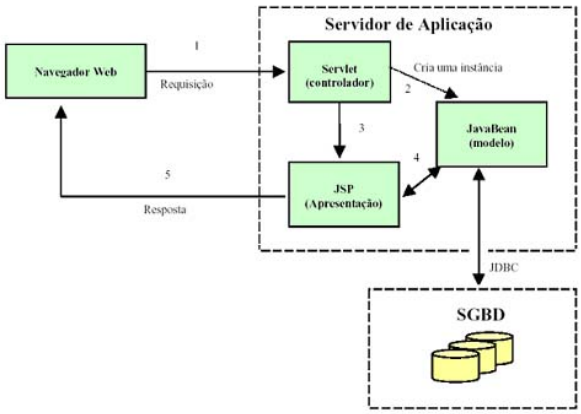
Quando uma atualização da aplicação é feita, automaticamente as lojas estarão acessando a versão atualizada da aplicação, ao contrário de aplicações que funcionam diretamente no micro local, onde a troca da aplicação toda se faz necessária, máquina a máquina.

O estoque é atualizado diretamente no servidor e consequentemente está sempre atualizado. Qualquer alteração de preços, nomes e características dos produtos que se altere, imediatamente já estarão sendo vistas pelas lojas.

Toda a aplicação e o banco de dados está em um servidor central, isso facilita a manutenção e o backup de todas as informações.

Estrutura da Aplicação

A aplicação foi montada baseada no modelo MVC [11] (model view controller), ou seja, existe um controlador que controla as entradas do usuário e baseada nessas entradas chama as funções que existem no modelo. Pode se ter uma idéia do funcionamento pela figura 7.1:



A arquitetura da aplicação. [11]

O servidor de aplicações foi o Tomcat e o cliente web pode ser qualquer browser web existente. O canco de dados utilizado foi o MySQL.

Estrutura de classes

A classe Controlador.java é um JavaBean que faz o papel do controlador, ele recebe as requisições enviadas pelo browser do cliente e atua na Aplicação.java, onde estão implementadas as interações com o banco de dados e o fluxo normal do sistema.

Existem ainda classes que modelam a Venda, Item de Venda, Cliente e Transferencia de produtos do estoque. Estas classes encapsulam os objetos e fornecem interfaces públicas para a manutenção de seus dados.

Como a aplicação foi inteiramente desenvolvida utilizando o paradigma da orientação a objetos, então todas as requisições, e serviços são baseados em objetos. Desta forma quando cadastramos um cliente, uma instância da classe Cliente é enviada à Aplicação, e dentro da classe aplicação o cliente então é armazenado no banco de dados.

Esta forma de implementação torna o sistema mais genérico, pois posso alterar os atributos da classe cliente, e a forma como suas funções são programadas, e a forma como a aplicação armazenará o cliente será a mesma, pois estou passando o objeto Cliente.

Esta técnica de programação é mais complexa e mais demorada mas provê uma reusabilidade de componentes maior, uma vez que esta classe Cliente pode ser utilizada em qualquer outra aplicação.

Considerações sobre o sistema

O sistema já está em funcionamento no escritório central, e está sendo utilizado na manutenção do estoque. Claramente o processo de atualização de estoque, movimentação de produtos entre lojas ficou mais simples e rápido de se executar, diminuindo as margens de erro que uma simples planilha do Excel causava.

O sistema de vendas, onde as lojas estarão efetuando suas vendas on-line depende apenas da infraestrutura de rede de cada loja, com serviços de internet a cabo, como ADSL entrar em funcionamento. Assim que estiverem conectadas on-line à Internet, o sistema de vendas já estará sendo modificado para ser executado diretamente na Internet.

A expectativa do sistema é facilitar, e agilizar os processos, e isto vem acontecendo. Diversos módulos ainda precisam ser desenvolvidos, como controle contábil, e controle de recursos humanos, mas ao que estre trabalho se propôs a fazer, que era automatizar os processos de venda e estoque foram cumpridos com êxito.

O processo de implantação em cada loja é de aproximadamente 10 dias, e o sistema é colocado em funcionamento em paralelo com o sistema atual, para fins de verificação da confiabilidade do sistema.

Considerações finais

Uma das maiores dificuldades encontradas foi o fato do sistema ter sido totalmente construído sobre a tecnologia Java, e a pessoa que aqui lhes escreve não conhecia a linguagem.

Me propus este desafio para aprender uma nova tecnologia, e novas arquiteturas de desenvolvimento de sistemas não vistas por mim durante a graduação, e tomei um bom tempo do projeto apenas para dominar a tecnologia e suas ferramentas de desenvolvimento.

Outra dificuldade foi na questão da modelagem, dúvidas sobre a responsabilidade de cada classe, como exemplo: “a quem pertence a responsabilidade de atualizar o cliente no banco de dados? pertence à aplicação pois ela conhece o cliente? Ou pertence ao próprio cliente, pois só ele melhor conhece seus atributos, assim passaríamos a conexão do banco de dados ao cliente, e ele mesmo se armazenaria”. Então questões de modelagem das classes, procurando modelar de uma forma que facilitasse a reusabilidade e a flexibilidade de alterações tomaram um bom tempo.

E finalmente a questão da persistência dos objetos no banco de dados foi um caso um pouco complicado, pois o mapeamento dos objetos em tabelas relacionais a princípio não é uma tarefa corriqueira. Objetos tiveram que ser

divididos em uma ou duas tabelas para serem armazenados.

Trabalhos futuros

Desenvolvimento de um sistema que permita transações automática entre a loja e o fornecedor, através da web. Tornando a reposição do estoque da empresa automático por exemplo. À medida que as peças em estoque vão baixando, e ao chegar em um nível limite, automaticamente a aplicação já se conectaria ao fornecedor e solicitaria os produtos. A tecnologia utilizada e que está surgindo com futuro promissor para a troca desse tipo de informações através da internet são os Webservices, utilizando XML, e acredito que cabe um bom trabalho sobre isto.

Desenvolvimento do módulo financeiro, para acompanhamento da saúde financeira da empresa, e integração com o módulo de estoque e clientes já existentes.

Conclusão final

O modelo de processamento cliente/servidor não é o final da evolução do ambiente de computação e a solução de todos os problemas, mas existem vários motivos para utilizá-lo, sendo especialmente adequado para ambientes de sistemas distribuídos na Internet.

Aplicações cliente/servidor podem ser desenvolvidas utilizando-se diversas técnicas de programação. O trabalho apresentou de forma sucinta as diversas formas de organização das camadas que constituem as aplicações cliente/servidor.

O objetivo principal deste trabalho é explorar as diversas técnicas de desenvolvimento de aplicações servidoras mostrando suas principais vantagens e empregar estas técnicas diretamente na resolução de um problema, no caso a implantação de um sistema de controle de estoque de uma rede de lojas.

A implementação utilizou do modelo Model-View-Controler, que consiste em separar na aplicação, os módulos responsáveis pelo modelo, pela apresentação e pelo controle de fluxo das informações, e tem se tornado um padrão nas aplicações para Internet pelos benefícios que traz ao desenvolvimento.

O sistema desenvolvido atende aos requisitos propostos e já está em funcionamento, e os objetivos de aprendizado de novas tecnologias para desenvolvimento na Internet utilizando Java foram alcançados.

Bibliografia

[1] COULOURIS, George ; DOLLIMORE, Jean ; KINDBERG, Tim,***Distributed Systems Concepts and Design***, Addison Wesley, 1994

[2] DEITEL, H. M. & DEITEL, P.J. ***Java, Como programar***. trad. Edson Furmankiewicz. Porto Alegre: Bookman, 2001.

[3] FOWLKES, Edward; SHIDA, Takashi, ***Multithreaded Programming***, 1996

[4] KOMOSINSKI, Leandro J. ***Aplicações Cliente-Servidor via Web usando Java***. Florianópolis, 2002.

[5] LARMAN, Craig. ***Utilizando UML e Padrões, Uma introdução à Análise e ao Projeto Orientado a Objetos***. trad. Luiz A. Meirelles Salgado. Porto Alegre: Bookman, 2000.

[6] PIMENTEL, Walter M. ***Aplicações Cliente/Servidor em Múltiplas Camadas***, 1998

[7] PIRES, Paulo de Figueiredo, ***Arquitetura Cliente/Servidor – Uma Chave para o Projeto de Sistemas Distribuídos***

[8] SADOSKI, Darleen. ***Client/Server Architectures – An Overview***. http://www.sei.cmu.edu/str/descriptions/clientserver_body.html, January 1997.

[9] SILVA, ***Francisco José da Silva, Ambiente de Apoio ao Desenvolvimento de Aplicações Distribuídas***, UFMA, 1997

[10] SUN, JavaTM 2 Platform, ***Enterprise Edition (J2EE) Overview*** http://java.sun.com/j2ee/overview2.html

[11] SUN, ***J2EE Architecture Approaches, Model-View-Controller Architecture*** http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/app-arch/app-arch2.html#1105854

[12] TANEMBAUM, Andrew S.; ***Sistemas Operacionais Modernos***,1992

[13] VINOSKI, Steve; SCHMIDT, Douglas,Comparing Alternative Client-side Distributed Programming Techniques (Column 3), 1995

Anexo 2

Código Fonte

Seguem os códigos da aplicação, também disponíveis em

http://www.inf.ufsc.br/~bokao/projeto.html

Código dos Javabeans, pertencentes ao “modelo” da aplicação:

Controlador.java

```
package webone.beans.ctrl;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;
import java.sql.*;
import webone.beans.app.*;

public class Controlador{

       private Aplicacao aplicacao;
       private boolean redirecionar;
       private String proxPagina;

       public Controlador(){
              redirecionar = false;
              proxPagina = null;
       }

    public void processaOpcao(HttpServletRequest request, HttpSession session)
throws Exception{
       redirecionar = false;
           if (request.getAttribute("acessoIlegal") != null) {
              proxPagina = "loginInvalido.jsp";
                redirecionar = true;

         } else {
              String opcao = request.getParameter("opcao");
              if (opcao == null) {
                     proxPagina = "index.jsp";
                     redirecionar = true;
               } else if(opcao.equals("venda")){
                     processaOpcaoVenda(request,session);
               } else if(opcao.equals("clientes")){
                     processaOpcaoClientes(request,session);
               } else if(opcao.equals("estoque")){
                     processaOpcaoEstoque(request,session);
               } else if(opcao.equals("relatorios")){
                            processaOpcaoRelatorios(request,session);
               } else if(opcao.equals("login")){
                            processeLogin(request,session);
              } else {
                     proxPagina = "index.jsp";
                     redirecionar = true;
              }
       }
    }
```

```
      private void processaOpcaoRelatorios(HttpServletRequest request, HttpSession
session)throws Exception{
            String subOpcao = request.getParameter("subOpcao");
      if (subOpcao == null){
                  proxPagina = "relatorios/relatorios.jsp";
            redirecionar = true;
          } else if (subOpcao.equals("porLoja")){
            proxPagina = "relatorios/relatorio1.jsp";
            redirecionar = true;
          } else if (subOpcao.equals("selLoja")){
            String loja = (String)request.getParameter("loja");
            proxPagina = "relatorios/relatorioLoja.jsp?loja="+loja;
            redirecionar = true;
          }
       }

      private void processaOpcaoEstoque(HttpServletRequest request, HttpSession
session)throws Exception{
            String subOpcao = request.getParameter("subOpcao");
      if (subOpcao == null){
                  proxPagina = "estoque/estoque.jsp";
            redirecionar = true;
          } else if(subOpcao.equals("completa")){
                  proxPagina = "estoque/completa.jsp";
            redirecionar = true;
          } else if(subOpcao.equals("entrada")){
                  proxPagina = "estoque/entrada.jsp";
            redirecionar = true;
          } else if(subOpcao.equals("fimEntrada")){
                  aplicacao.processeEntrada();
                  proxPagina = "estoque/estoque.jsp";
            redirecionar = true;
          } else if(subOpcao.equals("remover")){
                  int codigo = Integer.parseInt(request.getParameter("cod"));
                  aplicacao.getEntrada().removeProduto(codigo);
                  proxPagina = "estoque/entrada.jsp";
            redirecionar = true;
          } else if(subOpcao.equals("adicionar")){
                  EntradaProdutos entrada = aplicacao.getEntrada();
                  int codigo = Integer.parseInt(request.getParameter("codigo"));
                  int qtde = Integer.parseInt(request.getParameter("qtde"));
                  entrada.addProduto(codigo,qtde);
                  proxPagina = "estoque/entrada.jsp";
            redirecionar = true;
          }
          //funcoes referentes à tranferencia de produtos
          else if(subOpcao.equals("transferencia")){
            proxPagina = "estoque/transferencia.jsp";
            redirecionar = true;
          } else if(subOpcao.equals("transAdicionar")){
            Transferencia transferencia = aplicacao.getTransferencia();
                  int codigo = Integer.parseInt(request.getParameter("codigo"));
                  int qtde = Integer.parseInt(request.getParameter("qtde"));
                  transferencia.addProduto(codigo,qtde);
                  proxPagina = "estoque/transferencia.jsp";
            redirecionar = true;
          } else if(subOpcao.equals("transRemover")){
                  int codigo = Integer.parseInt(request.getParameter("cod"));
                  aplicacao.getTransferencia().removeProduto(codigo);
            proxPagina = "estoque/transferencia.jsp";
            redirecionar = true;
          } else if(subOpcao.equals("transFinaliza")){
            Transferencia trans = aplicacao.getTransferencia();
                  trans.setDeLoja(request.getParameter("deLoja"));
                  trans.setParaLoja(request.getParameter("paraLoja"));
                  aplicacao.processaTransferencia();
            proxPagina = "estoque/estoque.jsp";
            redirecionar = true;
```

```
            }
        }

    private void processeLogin(HttpServletRequest request, HttpSession
session)throws Exception{
            session.removeAttribute("usuarioAtual");
        String nome = request.getParameter("nome");
        String senha = request.getParameter("senha");

            if (aplicacao.valideLogin(nome,senha)) {
            session.setAttribute("usuarioAtual", nome);
           proxPagina = "principal.jsp";
           redirecionar = true;
         } else {
           proxPagina = "loginInvalido.jsp";
           redirecionar = true;
         }
        }

        private void processaOpcaoClientes(HttpServletRequest request, HttpSession
session)throws Exception{
        String subOpcao = request.getParameter("subOpcao");
        if (subOpcao == null){
            proxPagina = "clientes/clientes.jsp";
              redirecionar = true;
           } else if(subOpcao.equals("listar")) {
            proxPagina = "clientes/listaClientes.jsp";
            redirecionar = true;
           } else if(subOpcao.equals("procura")) {
            String tipo =  request.getParameter("tipo");
            String procura = request.getParameter("procura");
            proxPagina =
"clientes/listaClientes.jsp?procura="+procura+"&tipo="+tipo;
            redirecionar = true;
           } else if(subOpcao.equals("editar")) {
            proxPagina =
"clientes/editaCliente.jsp?codigo="+request.getParameter("select");
            redirecionar = true;
           } else if(subOpcao.equals("mostra")) {
            proxPagina =
"clientes/mostraCliente.jsp?codigo="+request.getParameter("cod");
            redirecionar = true;
           } else if(subOpcao.equals("remover")) {

        aplicacao.removeCliente(Integer.parseInt(request.getParameter("select")));
            proxPagina = "clientes/listaClientes.jsp";
            redirecionar = true;
           } else if(subOpcao.equals("adicionar")) {
            proxPagina = "clientes/adicionarClientes.jsp";
            redirecionar = true;
         } else if(subOpcao.equals("alterar")){
                    Cliente cliente = new Cliente();

        cliente.setCodigo(Integer.parseInt(request.getParameter("codigo")));
                            cliente.setNome(request.getParameter("nome"));

        cliente.setTelefone(Integer.parseInt(request.getParameter("telefone")));

        cliente.setCelular(Integer.parseInt(request.getParameter("celular")));

        cliente.setCpf(Long.parseLong(request.getParameter("cpf")));
                            cliente.setEndereco(request.getParameter("endereco"));
                            cliente.setBairro(request.getParameter("bairro"));
                            cliente.setCidade(request.getParameter("cidade"));

        cliente.setCep(Integer.parseInt(request.getParameter("cep")));
                            aplicacao.atualizaCliente(cliente);
                    proxPagina = "clientes/listaClientes.jsp";
```

```
                redirecionar = true;
         } else if (subOpcao.equals("fimAdicionar")){
                Cliente cliente = new Cliente();
                      cliente.setNome(request.getParameter("nome"));

       cliente.setTelefone(Integer.parseInt(request.getParameter("telefone")));

       cliente.setCelular(Integer.parseInt(request.getParameter("celular")));

       cliente.setCpf(Long.parseLong(request.getParameter("cpf")));
                      cliente.setEndereco(request.getParameter("endereco"));
                      cliente.setBairro(request.getParameter("bairro"));
                      cliente.setCidade(request.getParameter("cidade"));

       cliente.setCep(Integer.parseInt(request.getParameter("cep")));
                      aplicacao.cadastraCliente(cliente);
                proxPagina = "clientes/listaClientes.jsp";
                redirecionar = true;
         }
       }

    private void processaOpcaoVenda(HttpServletRequest request, HttpSession
session)throws Exception{
       String subOpcao = request.getParameter("subOpcao");
       if (subOpcao == null){
              proxPagina = "venda/formVenda.jsp";
                redirecionar = true;
       } else if (subOpcao.equals("selProduto")){
              Venda venda = aplicacao.getVenda();
                      ItemVenda item = new ItemVenda();

                      int codigo = Integer.parseInt(request.getParameter("codigo") );
              item.setCodigoProduto(codigo);
              ResultSet rs = aplicacao.processeQuery("select * from bones where
codigo = "+codigo);
                      if(rs.next()){
                            item.setNome(rs.getString(2));
                            item.setValorProduto(rs.getFloat(4));
                      }
              venda.setItem(item);
                      proxPagina = "venda/formVenda2.jsp";
                      redirecionar = true;
       } else if (subOpcao.equals("adicionar")){
                      int quantidade =
Integer.parseInt(request.getParameter("quantidade"));
                      float valorServico =
Float.parseFloat(request.getParameter("valorservico"));
                      ItemVenda item =
(ItemVenda)aplicacao.getVenda().getListaItens().lastElement();
                      item.setQuantidade(quantidade);
                      item.setValorServico(valorServico);
                      proxPagina = "venda/verItens.jsp";
                      redirecionar = true;
       } else if (subOpcao.equals("preFechar")){
                      proxPagina = "venda/fechaVenda.jsp";
                      redirecionar = true;
       } else if (subOpcao.equals("remover")){
                      int codigo = Integer.parseInt(request.getParameter("item"));
                      int qtde = Integer.parseInt(request.getParameter("qtde"));
                      Venda venda = aplicacao.getVenda();
                      venda.removeItem(codigo,qtde);
                      proxPagina = "venda/verItens.jsp";
                      redirecionar = true;
       } else if (subOpcao.equals("fechar")){
              Venda venda = aplicacao.getVenda();

       venda.setCodCliente(Integer.parseInt(request.getParameter("select")));
                      String loja = (String) session.getAttribute("usuarioAtual");
```

```
                        venda.setLoja(loja);
                        aplicacao.processaVenda();
                        proxPagina = "venda/formVenda.jsp";
                        redirecionar = true;
                }
        }


        public void setAplicacao(Aplicacao aplicacao) {
                this.aplicacao = aplicacao;
        }
        public Aplicacao getAplicacao() {
                return (this.aplicacao);
        }

        public void setRedirecionar(boolean redirecionar) {
                this.redirecionar = redirecionar;
        }
        public boolean getRedirecionar() {
                return (this.redirecionar);
        }

        public void setProxPagina(String proxPagina) {
                this.proxPagina = proxPagina;
        }
        public String getProxPagina() {
                return (this.proxPagina);
        }
}
```

Aplicacao.java

```
package webone.beans.app;
import java.sql.*;
import java.util.Vector;

public class Aplicacao {

        private BancoDados banco;
        private Venda venda;
        private EntradaProdutos entrada;
        private Transferencia transferencia;
        private Relatorio relatorio;

        public Aplicacao()throws Exception{
                banco = new BancoDados();
        }

        public boolean valideLogin(String nome, String senha) throws Exception{
                String consulta = new String("select * from usuarios where nome =
'"+nome+"' && senha = '"+senha+"'");
                ResultSet resultado = banco.processeQuery(consulta);
                if(resultado.next()){
                        if ( nome.equals(resultado.getString(1)) &&
senha.equals(resultado.getString(2) ) ) {
                                return true;
                        }
                }
                return false;
        }

        public Venda getVenda(){
                if (venda == null){
                        venda = new Venda();
                }
```

```
		return venda;
	}

	public Connection getConexao(){
		return banco.getConexao();
	}
	public Relatorio getRelatorio(){
		if (relatorio == null){
			relatorio = new Relatorio();
		}
		return relatorio;
	}

	public EntradaProdutos getEntrada(){
		if (entrada == null){
			entrada = new EntradaProdutos();
		}
		return entrada;
	}

	public Transferencia getTransferencia(){
		if (transferencia == null){
			transferencia = new Transferencia();
		}
		return transferencia;
	}

	public ResultSet processeQuery(String query) throws Exception{
		return banco.processeQuery(query);
	}

	public String getNomeBone(int codigo) throws Exception{
		String consulta = new String("select nome from bones where codigo =
"+codigo);
		ResultSet resultado = banco.processeQuery(consulta);
		resultado.first();
		String nome = resultado.getString(1);
		return nome;
	}


	/**
	* retorna os nomes das colunas de uma determinada tabela
	*/
	public Vector getNomeColunas(String tabela)throws Exception{
		Vector listaNomes = new Vector();
		String consulta = new String("select * from clientes");
		ResultSet resultado = banco.processeQuery(consulta);
		ResultSetMetaData metadata = resultado.getMetaData();
		for (int indice = 1; indice <= metadata.getColumnCount(); indice++){
			listaNomes.add(new String(metadata.getColumnName(indice)));
		}
		return listaNomes;
	}

	/**
	* Metodos para o tratamento do cliente
	*
	*/
	public Vector getClientes() throws Exception{
		String consulta = new String("select * from clientes order by nome");
		ResultSet resultado = banco.processeQuery(consulta);
		return clienteParaVector(resultado);
	}

	private Vector clienteParaVector(ResultSet resultado)throws Exception{
		Vector lista = new Vector();
		while(resultado.next()){
```

```
                Cliente cli = new Cliente();
                cli.setCodigo(resultado.getInt(1));
                cli.setNome(resultado.getString(2));
                cli.setTelefone(resultado.getInt(3));
                cli.setCelular(resultado.getInt(4));
                cli.setCpf(resultado.getLong(5));
                cli.setEndereco(resultado.getString(6));
                cli.setBairro(resultado.getString(7));
                cli.setCidade(resultado.getString(8));
                cli.setCep(resultado.getInt(9));
                lista.add(cli);
            }
            return lista;
      }

      public Cliente getCliente(int cod) throws Exception{
            String consulta = new String("select * from clientes where codigo =
"+cod);
            ResultSet resultado = banco.processeQuery(consulta);
            resultado.first();
            Cliente cli = new Cliente();
                cli.setCodigo(resultado.getInt(1));
                cli.setNome(resultado.getString(2));
                cli.setTelefone(resultado.getInt(3));
                cli.setCelular(resultado.getInt(4));
                cli.setCpf(resultado.getLong(5));
                cli.setEndereco(resultado.getString(6));
                cli.setBairro(resultado.getString(7));
                cli.setCidade(resultado.getString(8));
                cli.setCep(resultado.getInt(9));
            return cli;
      }

      public Cliente getClienteNome(String nome)throws Exception{
            String consulta = new String("select * from clientes where nome =
'+nome+' order by name");
            ResultSet resultado = banco.processeQuery(consulta);
            resultado.first();
            Cliente cli = new Cliente();
                cli.setCodigo(resultado.getInt(1));
                cli.setNome(resultado.getString(2));
                cli.setTelefone(resultado.getInt(3));
                cli.setCelular(resultado.getInt(4));
                cli.setCpf(resultado.getLong(5));
                cli.setEndereco(resultado.getString(6));
                cli.setBairro(resultado.getString(7));
                cli.setCidade(resultado.getString(8));
                cli.setCep(resultado.getInt(9));
            return cli;
      }


      public void cadastraCliente(Cliente cliente) throws Exception{
            banco.cadastraCliente(cliente);

        //pega o ultimo codigo inserido e seta para o cliente.
            String consulta = new String("select LAST_INSERT_ID()");
            ResultSet resultado = banco.processeQuery(consulta);
            resultado.first();
            cliente.setCodigo(resultado.getInt(1));
      }

      public void atualizaCliente(Cliente cliente) throws Exception{
            banco.atualizaCliente(cliente);
      }

      public void removeCliente(Cliente cliente) throws Exception{
```

```
            String consulta = new String("delete from clientes where codigo =
"+cliente.getCodigo());
            banco.processeQuery(consulta);
      }

      public void removeCliente(int codigo) throws Exception{
            String consulta = new String("delete from clientes where codigo =
"+codigo);
            banco.processeQuery(consulta);
      }

      public Vector procuraCliente(String procura,String tipo) throws Exception{
            String consulta = new String("select * from clientes where "+tipo+"
like '%"+procura+"%' order by nome");
            return clienteParaVector( banco.processeQuery(consulta) );
      }


      /**********************************************************************
      étodos para cuidar das transferencias de produtos
      ************************************/

      public ResultSet getEstoque() throws Exception{
            String consulta = new String("select bones.codigo,nome,estoque_.* from
bones, estoque_ where bones.codigo = estoque_.codigo order by bones.codigo");
          return banco.processeQuery(consulta);
      }

      public Vector getNomeLojas() throws Exception{
            String consulta = new String("select * from estoque_ where codigo =
100");
          ResultSet resultado = banco.processeQuery(consulta);
          ResultSetMetaData rsmeta = resultado.getMetaData();
          Vector nomes = new Vector();
          for(int i = 2 ; i <= rsmeta.getColumnCount() ; i++){
            nomes.add(rsmeta.getColumnName(i));
          }
          return nomes;
      }

      public void processeEntrada() throws Exception {
            int cod,qtde;
            int i = 0;
            int listaProdutos[][] = entrada.getListaProdutos();
            while(listaProdutos[i][0] != 0){
                  cod = listaProdutos[i][0];
                  qtde = listaProdutos[i][1];

                  //arruma coluna do QG
                  String query = new String("update estoque_ set qg = qg +
"+qtde+" where codigo = "+cod);
                  banco.processeQuery(query);

                  //arruma a transferencia
                  query = new String("update estoque_ set tqg = tqg + "+qtde+"
where codigo = "+cod);
                  banco.processeQuery(query);

                  //armazena na tabela entradaEstoque
                  Date datasql = new Date(entrada.getData().getTime());

                  query = new String("insert into entradaEstoque values('"+
                              datasql+"',"+cod+",'"+qtde+"')");
                  banco.processeQuery(query);
                  i++;
            }
            entrada.inicializa();
      }
```

```
	public boolean processaTransferencia()throws Exception{
	    String aux = new String();
	    String deLoja = transferencia.getDeLoja().toLowerCase();
	      String paraLoja = transferencia.getParaLoja().toLowerCase();
	      int listaProdutos[][] = transferencia.getListaProdutos();
	      int cod,qtde;


	      int i = 0;
	      while(listaProdutos[i][0] != 0){
	            cod = listaProdutos[i][0];
	            qtde = listaProdutos[i][1];

	      //baixa da loja inicial
	            aux = "update estoque_ set "+deLoja+" = "+deLoja+" - "+qtde+"
where codigo = "+cod;
	            banco.processeQuery(aux);

	            // baixa da transferencia mensal da loja inicial
	            // se for na loja QG então não mexe
	            if (!deLoja.equals("qg")){
	                  aux = "update estoque_ set t"+deLoja+" = t"+deLoja+" -
"+qtde+" where codigo = "+cod;
	                  banco.processeQuery(aux);
	            }

	            //adiciona na loja final
	            aux = "update estoque_ set "+paraLoja+" = "+paraLoja+" +
"+qtde+" where codigo = "+cod;
	            banco.processeQuery(aux);


	            //adiciona na transferencia mensal da loja final
	            // se for na loja QG então não mexe
	            if (!paraLoja.equals("qg")){
	                  aux = "update estoque_ set t"+paraLoja+" = t"+paraLoja+"
+ "+qtde+" where codigo = "+cod;
	                  banco.processeQuery(aux);
	            }
	      i++;
	      }
	      transferencia.inicializa();
	      return true;
	}


	public void processaVenda() throws Exception{

	      for (int i=0 ; i < venda.getListaItens().size(); i++){
	            ItemVenda item = (ItemVenda)venda.getListaItens().elementAt(i);

	banco.cadastraItemVenda(item,venda.getCodCliente(),venda.getLoja(),venda.getD
ata());
	            // atualiza o estoque da venda.
	            processaEstoqueVenda(item,venda.getLoja());
	      }
	      venda.novaVenda();
	}

	private void processaEstoqueVenda(ItemVenda item,String loja) throws
Exception{
	      int codigo = item.getCodigoProduto();
	      int quantidade = item.getQuantidade();
	      loja = loja.substring(0,1);
```

```
            String query  = new String("update estoque_ set "+loja+" = "+loja+" -
"+quantidade+" where codigo = "+codigo);
            banco.processeQuery(query);
      }

}
```

BancoDados.java

```
package webone.beans.app;

import java.sql.*;
import java.util.*;

public class BancoDados{


      private Connection conexao;
      private ResultSet resultado = null;
      private PreparedStatement query = null;


      public BancoDados() throws Exception{
            Class.forName("org.gjt.mm.mysql.Driver");
          conexao =
DriverManager.getConnection("jdbc:mysql://localhost/bone?user=henrique&password=200
2henrique&autoReconnect=true");

      }

      public Connection getConexao(){
            return conexao;
      }

      public synchronized ResultSet processeQuery(String query) throws Exception{
            PreparedStatement stt = conexao.prepareStatement(query);
            return stt.executeQuery();
      }

      public synchronized void cadastraCliente(Cliente cliente) throws Exception{
            query = conexao.prepareStatement("insert into clientes values
('',?,?,?,?,?,?,?,?)");
            query.setString(1,cliente.getNome());
            query.setInt(2,cliente.getTelefone());
            query.setInt(3,cliente.getCelular());
            query.setLong(4,cliente.getCpf());
            query.setString(5,cliente.getEndereco());
            query.setString(6,cliente.getBairro());
            query.setString(7,cliente.getCidade());
            query.setInt(8,cliente.getCep());
            query.execute();

      }

      public synchronized void atualizaCliente(Cliente cliente) throws Exception{
            query = conexao.prepareStatement("update clientes set nome = ?,"+
                  "telefone = ?,celular = ?,cpf = ?,endereco = ?,bairro = ?,"+
                  "cidade = ?,cep = ? where codigo = ?");
            query.setString(1,cliente.getNome());
            query.setInt(2,cliente.getTelefone());
            query.setInt(3,cliente.getCelular());
            query.setLong(4,cliente.getCpf());
            query.setString(5,cliente.getEndereco());
            query.setString(6,cliente.getBairro());
            query.setString(7,cliente.getCidade());
            query.setInt(8,cliente.getCep());
            query.setInt(9,cliente.getCodigo());
            query.execute();
```

```
	}

	public synchronized void cadastraItemVenda(ItemVenda item,int
codCliente,String loja,java.util.Date data)throws Exception{
		PreparedStatement query = conexao.prepareStatement("insert into venda
values (?,?,?,?,?,'',?,?,?)");

		query.setInt(1,codCliente);
		query.setInt(2,item.getCodigoProduto());
		query.setFloat(3,item.getValorProduto());
		query.setFloat(4,item.getValorServico());
		query.setFloat(5,item.getValorTotal());
		query.setString(6,loja);
		query.setDate(7,new java.sql.Date(data.getTime()));
		query.setInt(8,item.getQuantidade());
		query.execute();

	}

}
```

Cliente.java

```
package webone.beans.app;

public class Cliente {

	private int codigo;
	private String nome,bairro,cidade,endereco;
	private int telefone,celular,cep;
	private long cpf;

	public Cliente(){
	}

	public void setCodigo(int codigo) {
		this.codigo = codigo;
	}

	public void setNome(String nome) {
		this.nome = nome;
	}

	public void setTelefone(int telefone) {
		this.telefone = telefone;
	}
	public int getCodigo() {
		return (this.codigo);
	}

	public String getNome() {
		return (this.nome);
	}

	public int getTelefone() {
		return (this.telefone);
	}


	public void setBairro(String bairro) {
		this.bairro = bairro;
	}

	public void setCidade(String cidade) {
		this.cidade = cidade;
	}
```

```
	public void setEndereco(String endereco) {
		this.endereco = endereco;
	}

	public void setCelular(int celular) {
		this.celular = celular;
	}

	public void setCep(int cep) {
		this.cep = cep;
	}
	public String getBairro() {
		return (this.bairro);
	}

	public String getCidade() {
		return (this.cidade);
	}

	public String getEndereco() {
		return (this.endereco);
	}

	public int getCelular() {
		return (this.celular);
	}

	public int getCep() {
		return (this.cep);
	}


	public void setCpf(long cpf) {
		this.cpf = cpf;
	}
	public long getCpf() {
		return (this.cpf);
	}


}
```

EntradaProdutos.java

```
/*
	Classe Entrada

*/
package webone.beans.app;

import java.util.Vector;

public class EntradaProdutos {

	java.util.Date data;
	int listaProdutos[][];

	public EntradaProdutos(){
		inicializa();
	}

	public void inicializa(){
		listaProdutos = new int[100][2];
		data = new java.util.Date();
```

```
	}

	public void addProduto(int codigo,int qtde){
		int count = 0;
		while (listaProdutos[count][0] != 0){
			count++;
		}
		listaProdutos[count][0] = codigo;
		listaProdutos[count][1] = qtde;
	}

	public void removeProduto(int codigo){
		int i = codigo;
		while(listaProdutos[i][0] != 0){
			listaProdutos[i][0] = listaProdutos[i+1][0];
			listaProdutos[i][1] = listaProdutos[i+1][0];
			i++;
		}
	}

	public void setData(java.util.Date data) {
		this.data = data;
	}

	public void setListaProdutos(int[][] listaProdutos) {
		this.listaProdutos = listaProdutos;
	}

	public java.util.Date getData() {
		return this.data;
	}

	public int[][] getListaProdutos() {
		return this.listaProdutos;
	}

}
```

ItemVenda.java

```
package webone.beans.app;

public class ItemVenda{

	private int quantidade;
	private int codigoProduto;
	private float valorProduto;
	private float valorServico;
	private int codigoInterno;
	public String nome;

	public ItemVenda(){
		inicializa();
	}

	public void inicializa(){
		quantidade = 0;
		codigoProduto = 0 ;
		valorProduto = 0;
		valorServico = 0;
		codigoInterno = 0;
		nome = new String();
	}

	public void setQuantidade(int quantidade) {
		this.quantidade = quantidade;
	}
```

```
	public void setCodigoProduto(int codigoProduto) throws Exception{
		this.codigoProduto = codigoProduto;
	}

	public void setCodigoInterno(int codigoInterno) {
		this.codigoInterno = codigoInterno;
	}
	public int getQuantidade() {
		return (this.quantidade);
	}

	public int getCodigoProduto() {
		return (this.codigoProduto);
	}

	public int getCodigoInterno() {
		return (this.codigoInterno);
	}


	public void setValorProduto(float valorProduto) {
		this.valorProduto = valorProduto;
	}
	public float getValorProduto() {
		return (this.valorProduto);
	}

	public void setValorServico(float valorServico) {
		this.valorServico = valorServico;
	}
	public float getValorServico() {
		return (this.valorServico);
	}

	public void setNome(String nome) {
		this.nome = nome;
	}
	public String getNome() {
		return (this.nome);
	}

	public float getValorTotal(){
		float valorTotal;
		valorTotal = (valorProduto+valorServico)*this.quantidade;
		return valorTotal;
	}


}
```

Transferencia.java

```
/*
	Classe Transferencia

	Modela uma transferencia de produtos entre lojas
	Guarda informacoes de quais produtos foram de uma loja pra outra,
	e guarda quais foram as lojas envolvidas na tranferencia.
*/
package webone.beans.app;

import java.util.Date;
import java.util.Vector;
```

```
public class Transferencia {

	String deLoja;
	String paraLoja;
	Date data;
	int listaProdutos[][];

	public Transferencia(){
		inicializa();
	}

	public void inicializa(){
		deLoja = new String();
		paraLoja = new String();
		data = new Date();
		listaProdutos = new int[100][2];
	}

	public void addProduto(int codigo,int qtde){
		int count = 0;
		while (listaProdutos[count][0] != 0){
			count++;
		}
		listaProdutos[count][0] = codigo;
		listaProdutos[count][1] = qtde;
	}

	public void removeProduto(int codigo){
		int i = codigo;
		while(listaProdutos[i][0] != 0){
			listaProdutos[i][0] = listaProdutos[i+1][0];
			listaProdutos[i][1] = listaProdutos[i+1][0];
			i++;
		}
	}

	public void setDeLoja(String deLoja) {
		this.deLoja = deLoja;
	}

	public void setParaLoja(String paraLoja) {
		this.paraLoja = paraLoja;
	}

	public void setData(Date data) {
		this.data = data;
	}

	public void setListaProdutos(int[][] listaProdutos) {
		this.listaProdutos = listaProdutos;
	}
	public String getDeLoja() {
		return this.deLoja;
	}

	public String getParaLoja() {
		return this.paraLoja;
	}

	public Date getData() {
		return this.data;
	}

	public int[][] getListaProdutos() {
		return this.listaProdutos;
	}
```

```
}
```

Usuario.java

```
package webone.beans.app;


public class Usuario{

	private String nome;
	private String senha;

	public Usuario(){
	}

	public Usuario(String nome, String senha){
		this.nome = nome;
		this.senha = senha;

	}

	public void setNome(String nome) {
		this.nome = nome;
	}

	public void setSenha(String senha) {
		this.senha = senha;
	}
	public String getNome() {
		return (this.nome);
	}

	public String getSenha() {
		return (this.senha);
	}


}
```

Venda.java

```
package webone.beans.app;

import java.util.Vector;

public class Venda {

	private Vector listaItens;
	private float valorTotal;
	private int codCliente;
	private String loja;
	private java.util.Date data;

	public Venda(){
		novaVenda();
	}

	public void novaVenda(){
		listaItens = new Vector();
		valorTotal = 0;
		codCliente = 0;
		loja = new String();
		data = new java.util.Date();
```

```
	}

	public void setItem(ItemVenda item){
		listaItens.add(item);
	}

	public String getLoja(){
		return loja;
	}

	public void setLoja(String temp){
		loja = temp;
	}


	public void setListaItens(Vector listaItens) {
		this.listaItens = listaItens;
	}

	public void setValorTotal(float valorTotal) {
		this.valorTotal = valorTotal;
	}

	public void setCodCliente(int codCliente) {
		this.codCliente = codCliente;
	}

	public Vector getListaItens() {
		return (this.listaItens);
	}

	public float getValorTotal() {
		for (int i=0 ; i < this.listaItens.size(); i++){
			ItemVenda item = (ItemVenda)listaItens.elementAt(i);
			valorTotal = item.getValorTotal();
		}
		return (this.valorTotal);
	}

	public int getCodCliente() {
		return (this.codCliente);
	}

	public void adicionaItem(ItemVenda item){
		this.valorTotal += item.getValorProduto();
		this.listaItens.add(item);
	}

	public void removeItem(int codigo,int qtde){
		for (int i=0 ; i < this.listaItens.size(); i++){
			ItemVenda item = (ItemVenda)listaItens.elementAt(i);
			if ( item.getQuantidade() == qtde && item.getCodigoProduto() ==
codigo){
				listaItens.removeElementAt(i);
			}
		}
	}


	public void setData(java.util.Date data) {
		this.data = data;
	}
	public java.util.Date getData() {
		return (this.data);
	}


}
```