

Gustavo P. Santos

0520016

**APLICAÇÃO DO PADRÃO DE PROJETO
MVC COM JSF**

Jaguariúna

2008

Gustavo P. Santos

0520016

**APLICAÇÃO DO PADRÃO DE PROJETO
MVC COM JSF**

Monografia apresentada à disciplina Trabalho de Graduação III, do Curso de Ciência da Computação da Faculdade de Jaguariúna, sob a orientação do Prof. Ms. Peter Jandl Jr., como exigência parcial para conclusão do curso de graduação.

Jaguariúna

2008

SANTOS, Gustavo P. **APLICAÇÃO DO PADRÃO DE PROJETO MVC COM JSF**,
Monografia defendida e aprovada na FAJ em 11 de Dezembro de 2008 pela banca
examinadora constituída pelos professores:

Prof. Ms. Peter Jndl Jr – FAJ Orientador

Prof.

Prof.

A Deus, meus pais, irmãos, namorada e professores

O meu sincero obrigado pela força, colaboração e compreensão nos momentos mais difíceis de minha vida, me auxiliando e me orientando para o rumo certo.

AGRADECIMENTOS

Ao Coordenador / Orientador / Mestre / Professor Peter Jandl Jr., manifesto minha enorme gratidão, não só pelo trabalho em si, mas pela orientação nestes 4 anos juntos, me orientando e me direcionando, fica aqui meu imenso afeto e agradecimento, a uma memória viva da Computação.

Em especial aos Professores, Prof. Ademário Jr. pela lógica brilhante demonstrada em sala de aula, Prof. Sílvio Petroli Neto pelas aulas e colaboração em todos os momentos, um amigo pra vida toda , Prof. Luciano Calderoni pelas aulas simples e objetivas, Miro pelos espetáculos de matemática, Carlos Viviani pelas aulas, conselhos e amizade demonstrada em momentos difíceis, Ricardo pelas aulas e orientações nos últimos anos , e a todos os professores que não citei dedico a vocês meu enorme afeto e carinho.

A minha namorada que amo tanto, Deus me presenteou por diversas vezes durante os 4 anos desta faculdade, mas de todos presentes que ganhei nos últimos 4 anos nada se compara ao brilho e companheirismo desta enorme mulher e companheira, a você meu eterno amor por tudo que era, sou e espero ser ao seu lado.

Aos meus pais e irmãos e família, pelo enorme carinho, compreensão e amor durante este período de minha vida, dedico a vocês este trabalho inteiro, e todos estes 4 anos de faculdade, por tudo que sempre passamos e passaremos juntos e unidos para sempre, a vocês meu tudo....

Aos amigos, Juliano Peruffo, Fabio Q. Santana, por tudo que vivemos juntos e nos ajudamos, por todas risadas, conquistas e derrotas vividas nestes 4 anos, a vocês minha enorme fidelidade e a expectativa que continuemos assim com um laço forte de amizade e nos encontrando sempre.

Enfim a todos que por algum motivo cruzaram por minha vida e me iluminaram durante esta fase tão importante de minha vida, desejo a todos vocês felicidade, saúde, paz e muito amor, espero que todos aqui citados e não citados, levem um pedacinho de mim, pois com certeza estou levando muito de vocês....

Quase todos os homens são capazes de suportar adversidades, mas se quiser pôr à prova o caráter de um homem, dê-lhe poder.

(Abraham Lincoln)

SANTOS, Gustavo P. **APLICAÇÃO DO PADRÃO DE PROJETO MVC COM JSF**, 2008. Monografia (Bacharelado em Ciência da Computação) – Curso de Ciência da Computação da Faculdade de Jaguariúna, Jaguariúna.

RESUMO

Hoje em dia, com o grande crescimento e avanço da tecnologia, os softwares estão cada vez mais presentes nas grandes empresas, e, relacionado com este crescimento, também é elevado o número de projetos inacabados, gerados ou pela falta de documentação ou de alguma ferramenta que auxilie no desenvolvimento.

Com o advento da Internet e sua rápida expansão surgiram os softwares voltados para web, que proporcionam o fácil acesso ao mesmo sistema em qualquer lugar em que ela esteja, não apenas para os usuários finais, mas também por grandes corporações, onde o tempo é dinheiro e qualquer forma de integrar um processo de maneira rápida e eficiente é que faz dela uma empresa diferenciada.

Diante dos fatores apresentados, este trabalho tem por objetivo mostrar os padrões de projetos MVC (Model View Controller) e o framework JSF 1.2. Envolvendo todos estes conceitos é possível ter um software com qualidade, e que atenda todas as necessidades do mundo corporativo no tocante ao tempo reduzindo, com isso, os custos de desenvolvimento.

Palavras-chave: PADROES DE PROJETO, MVC, JSF 1.2.

SUMÁRIO

LISTA DE FIGURAS.....	xii
LISTA DE TABELAS.....	xiii
1 INTRODUÇÃO	1
2 PADRÕES DE PROJETO	2
2.1 Definições	2
2.2 Características.....	3
2.3 Descrição de Padrões de Projeto.....	3
2.4 Classificação GOF	6
2.4.1 Padrões de criação	6
2.4.2 Padrões estruturais	7
2.4.3 Padrões comportamentais	7
3 MODELO MVC (MODEL VIEW CONTROLLER).....	8
3.1 Camadas.....	9
3.1.1 Aplicação de uma camada.....	9
3.1.2 Aplicação de duas camadas.....	9
3.1.3 Aplicação de três camadas	10
3.2 O Padrão MVC.....	11
3.2.1 Objetivos.....	12
3.2.2 Características	12
3.3 Camadas no MVC.....	13
3.3.1 Camada de Apresentação.....	13
3.3.2 Camada Lógica da Aplicação	13
3.3.3 Camada de Controle da Aplicação.....	13
3.4 Vantagens e desvantagens do MVC	14
3.4.1 Vantagens.....	14
3.4.2 Desvantagens	14
3.5 Frameworks MVC.....	15
4 O FRAMEWORK JSF (<i>Java Server Faces</i>)	16
4.1 MVC no JSF.....	16
4.1.1 Controle:.....	16
4.1.2 Modelo	17
4.1.3 Visualização.....	17

4.2	Ciclo de vida do JSF	18
4.3	Características.....	19
4.4	Tags do JSF	20
5	PROTOTIPO MVC.....	22
5.1	Descrição do ambiente desenvolvimento	22
5.2	Descrição do Protótipo	23
5.2.1	Fluxo de Navegação.....	23
5.2.2	Diagrama de Classes.....	25
5.2.3	Área Pública.....	26
5.2.4	Área Cliente	27
5.2.5	Área Funcionários.....	27
5.3	Desenvolvimento do Protótipo.....	28
5.3.1	MVC no Protótipo.....	28
5.3.2	Desenvolvimento da Área Pública.....	30
5.3.3	Área Cliente	34
5.3.4	Área Funcionário	38
6	CONCLUSÕES.....	41
7	REFERÊNCIAS BIBLIOGRÁFICAS	42

LISTA DE FIGURAS

Figura 1	Aplicação de uma camada.....	9
Figura 2	Aplicação com duas camadas.....	10
Figura 3	Aplicação MVC de três camadas.....	11
Figura 4	Arquitetura do padrão MVC.....	14
Figura 5	Tecnologia JSF para UIs.....	17
Figura 6	Ciclo de vida do JSF.....	19
Figura 7	Fluxo de navegação do protótipo desenvolvido.	24
Figura 8	Fragmento do arquivo de configuração faces-config.xml.....	25
Figura 9	Diagrama de classes do protótipo desenvolvido.....	26
Figura 10	MVC no framework JSF 1.2.....	29
Figura 11	Desenvolvimento da pagina inicial.....	30
Figura 12	Desenvolvimento da pagina inicial.....	31
Figura 13	Desenvolvimento da pagina inicial.....	32
Figura 14	Desenvolvimento da pagina inicial.....	33
Figura 15	Desenvolvimento área cliente.....	34
Figura 16	Desenvolvimento área cliente.....	35
Figura 17	Desenvolvimento área cliente.....	36
Figura 18	Operação de insert.....	37
Figura 19	Bean X paginas.....	37

Figura 20 Desenvolvimento área funcionário.....	38
Figura 21 Pagina x estilo	39
Figura 22 Pagina consulta x tags tabela.....	39

LISTA DE TABELAS

Tabela 1	Fundamentais JSF.....	21
Tabela 2	Tags html do JSF.....	21
Tabela 3	Atributos Básicos Comum a Quase Todos Componentes.....	22

1 INTRODUÇÃO

A realização de um software nem sempre é fácil, pois de acordo com o grau de complexidade do software, é também o grau de soluções possíveis para um mesmo projeto, e por diversas vezes, o projeto se encontra em uma encruzilhada para saber qual solução adotar, Knuth¹ (1968) diria “a otimização prematura é a raiz de todo mal”.

Em quase todos os problemas ou na sua grande maioria o seu entendimento não tem uma definição clara, muitas vezes nem mesmo tem uma documentação e quando tem a mesma geralmente esta desatualizada ou incompleta.

Nos dias de hoje fica difícil falar em desenvolvimento de software sem falar de OO (Orientação a Objeto), que pode agregar a qualquer software em desenvolvimento, confiabilidade, robustez, distributividade, armazenabilidade, extensibilidade e reusabilidade, mas apenas o fato de utilizar a orientação a objeto não garante todas as qualidades acima descritas.

Os padrões de projeto (ou *design patterns*) vem despertando o interesse de todas as áreas de desenvolvimento, por sua reusabilidade, documentação de problemas e suas soluções, o que garantem uma melhor qualidade do software final, tanto em custos, como em desempenho.

Este trabalho mostra uma idéia básica dos principais padrões de projeto, mas focando no uso do padrão de projeto MVC para desenvolvimento de aplicações web, com a utilização do framework JSF (*Java Server Faces*).

Um padrão de projeto sistematicamente nomeia, motiva e explica um projeto genérico, que endereça um problema de projeto recorrente em sistemas orientados a objetos. Ele descreve o problema, a solução, quando é aplicável e quais as consequências de seu uso.

¹ Donald E. Knuth é um dos maiores cientistas da computação de que se tem notícia, tendo colaborado em diversas áreas, entre elas, programação, algoritmos e sistemas operacionais.

2 PADRÕES DE PROJETO

2.1 *Definições*

Uma solução padrão para um problema comum de programação, uma técnica capaz de tornar o código mais flexível ao fazer com que o código satisfaça certos critérios, um projeto ou uma estrutura de implementação que satisfaz com sucesso um propósito específico, um idioma de programação em alto nível, uma maneira mais prática de se descrever certos aspectos da organização de um programa, conexões entre componentes de programas, a forma de um diagrama de objeto ou de um modelo de objeto.

Em um padrão de projeto o foco está no problema e na solução do padrão, que quer dizer aonde ele se aplica de uma maneira genérica, para diversos problemas recorrentes nos projetos, assim um determinado padrão de projeto pode ser reutilizado em outro problema mas com algumas mudanças, mas isto não quer dizer que seja um novo padrão, e sim que é o mesmo padrão utilizado para uma solução x e com mudanças de algumas particularidades do mesmo possa ser aplicado a solução y.

Para a descrição de um padrão de projeto existem varias maneiras para documentar este padrão de projeto, mas todas elas exigem que os problemas e soluções envolvidos no padrão sejam documentadas de maneira estruturada, criando assim como se fosse um padrão para a criação de um padrão de projeto.

É de conhecimento de todos da complexidade na hora da criação de um software, e nessa hora seria imprescindível que os projetistas mais experientes pudessem passar isso aos mais novos, trazendo assim soluções cada vez mais aprimoradas, mas nem sempre essa transferência é fácil. Um dos métodos usados pelos projetistas é evitar que uma determinada solução seja construída do inicio ao fim, que quer dizer que entendido o problema, deve-se verificar dentro de uma solução problemas recorrentes e aplicar a estes a reutilização de técnicas já conhecidas.

Na maioria dos casos o foco esta na solução em si, sempre enfatizando “*o que*” e o “*como*” e não o “*porque*” da solução, e desta maneira a documentação pode não servir.

O problema chave para qualquer padrão de projeto é a importante descrição do problema e forma como sua solução é aplicável, mostrando suas limitações e consequências no emprego desta solução.

Os padrões de projeto visam suprir essas necessidades, tornando-se uma linguagem de fácil entendimento a todos os desenvolvedores do projeto, fazendo com que todos possam compartilhar destes padrões, de maneira mais rápida e com melhor qualidade, garantindo assim melhores projetos de softwares, não permitindo que soluções prontas, testadas e eficazes sejam reinventadas novamente e sim apenas concentrarem seus esforços para o desenvolvimento de problemas desconhecidos.

2.2 Características

Os padrões de projeto exibem várias características:

- Os padrões de projeto sempre justificam suas soluções para problemas concretos e bem definidos.
- As soluções aplicadas devem ser comprovadas com testes e experimentos.
- Para que o problema seja um padrão de projeto o mesmo deve ocorrer em diferentes sistemas, pois caso contrario o mesmo não pode ser um padrão de projeto.
- Os padrões de projeto devem absorver a evolução e o aprimoramento das soluções , bem como equilibrar os pontos fortes e fracos encontrados, não sendo assim soluções obvias ou triviais.
- Os padrões de projeto devem interagir com outros padrões, criando assim uma linguagem de padrões.

Pode-se desta maneira definir que em um padrão de projeto no que diz respeito a sua idéia é bem mais amplo do que uma classe definida em OO, pois mostra os conceitos e estruturas que nem sempre são objetos.

2.3 Descrição de Padrões de Projeto

Para que os padrões de projeto possam ser utilizados como um dicionário de soluções entre os projetistas, os mesmos devem ser especificados adequadamente, como os padrões de projeto são uma linguagem de alto nível,

os mesmos são descritos textualmente, de modo a caracterizar o problema, onde ele ocorre, sua análise e solução aplicada.

Sabe-se que a descrição textual de um padrão de projeto pode ser estruturada de diversas formas, então os padrões seguem um cronograma que oriente os projetistas sobre quais elementos de um padrão devem ser documentados, tornando sua descrição de fácil utilização e entendimento para os profissionais da área.

Entre as diversas formas que existem os mais populares são de:

- Alexander (Alexander, 1977 pag x –xl),
- GOF (Gamma, ET AL., 1995, p. 6-7),
- Portland (Cunningham, 2002) ou
- Coplien (Coplien, 1996, p 14).

Em todas as formas descritas pelos projetistas acima, a estrutura essencial para a descrição de um padrão seria: nome do padrão, propósito, problema, contexto, dificuldades, limitações ou restrições, solução, resultados e esboços.

Como se sabe a forma GOF é o padrão mais utilizado pelos projetistas, por se tratar de uma forma objetiva, organizada e dirigida ao software OO. A estrutura definida por GOF é:

Nome e classificação:

O nome tem uma importância muito grande para o padrão, pois tem que dar a idéia do problema, solução, características e consequências o que nem sempre é fácil.

Propósito:

Descrição sintética do propósito do padrão, o que faz, sua idéia e qual o problema pretende resolver.

Nome secundário:

Caso venha a ter, que o outro nome pode caracterizar este padrão.

Motivação:

Descreve o problema e onde ele ocorre, modelando como os elementos deste padrão podem solucionar o problema.

Aplicabilidade:

Descreve em que contexto o padrão deve ser aplicado, mostrando um projeto problemático, no qual o uso deste padrão poderia ser aplicado, mostrando como identificar essas situações em uso.

Estrutura:

Exemplo gráfico das classes envolvidas no padrão, utilizando uma notação padronizada, que hoje a mais adequada seria a UML.

Participantes:

Especifica todos os elementos envolvidos na solução e suas responsabilidades, bem como elas interagem, mostrando as idéias em um nível mais alto do que a implementação da solução.

Colaborações:

Descreve como os participantes se relacionam, como desempenham seus papéis na solução.

Conseqüências:

Descreve os resultados do uso do padrão de projeto, vantagens e desvantagens, como flexibilidade portabilidade e outras características.

Implementação:

Relaciona dicas úteis na implementação do padrão de projeto, como conselhos, avisos sobre problemas potenciais, como por exemplo qual linguagem específica utilizar.

Exemplo:

Trechos de códigos com linguagens representativas (C++, Java) que mostre a implementação do padrão.

Usos conhecidos:

Situações onde o padrão de projeto foi utilizado.

Padrões Relacionados:

Outros padrões que fazem parte de padrão de projeto, e que poderia ou não ser usado no problema em análise para a sua solução.

2.4 Classificação GOF

Em geral os padrões de projeto podem, segundo Gamma et. al., (2000), ser classificados em três diferentes tipos:

Padrões de criação:

Abstraem o processo de criação de objetos a partir da instanciação de classes.

Padrões estruturais:

Tratam da forma como classes e objetos estão organizados para a formação de estruturas maiores.

Padrões comportamentais:

Preocupam-se com algoritmos e a atribuição de responsabilidade entre objetos.

2.4.1 Padrões de criação

Abstract Factory:

Fornece uma interface para a criação de famílias de objetos relacionados ou dependentes sem especificar suas classes concretas. Uso conhecido: transportabilidade entre diferentes bibliotecas de interfaces gráficas (Gnome e KDE).

Factory Method:

Define uma interface para criar um objeto, mas deixa as subclasses decidirem qual classe será instanciada. Permite a uma classe postergar a instanciação de subclasses. Uso conhecido: bastante usado em *toolkits* e *frameworks* para a instanciação de objetos.

Singleton:

Garante que uma classe tenha somente uma instância e fornece um ponto global de acesso para ela. Uso conhecido: programas que só podem ter uma instância em uso em um dado momento.

2.4.2 Padrões estruturais

Adapter:

Converte a interface de uma classe em outra interface esperada pelos clientes. Permite que certas classes trabalhem em conjunto, pois de outra forma seria impossível por causa de suas interfaces incompatíveis. Uso conhecido: popularmente conhecido como *wrapper*, usado para adaptar a interface de classes.

Composite:

Compõe objetos em estrutura de árvore para representar hierarquias do tipo parte-todo. Permite que os clientes da estrutura tratem objetos individuais e composições de objetos de maneira uniforme. Uso conhecido: para representar hierarquias parte-todo.

Decorator:

Atribui responsabilidades adicionais a um objeto dinamicamente. Fornece uma alternativa flexível à utilização de subclasses para a extensão de funcionalidades. Uso conhecido: para a atribuição de efeitos gráficos e outras funcionalidades acessórias a *widgets*.

2.4.3 Padrões comportamentais

Command:

Encapsula uma solicitação como um objeto, permitindo a parametrização de clientes com diferentes solicitações, o enfileiramento e o registro de solicitações e o suporte a operações que possam ser, por exemplo, desfeitas. Uso conhecido: suporte a “desfazer”.

Iterator:

Fornece uma maneira de acessar seqüencialmente os elementos de um objeto agregado sem expor sua representação subjacente. Uso conhecido: C++ STL (*Standard Template Library*).

Observer:

Define uma dependência um-para-muitos entre objetos, de modo que, quando um objeto muda de estado, todos os seus dependentes são automaticamente notificados e atualizados. Uso conhecido: propagação de mudanças e atualizações com acoplamento fraco entre os objetos.

Strategy:

Define uma família de algoritmos, encapsula cada um deles e os faz intercambiáveis. Permite que o algoritmo varie independentemente dos clientes que o utilizam. Uso conhecido: sistemas de otimização.

3 MODELO MVC (MODEL VIEW CONTROLLER)

A orientação a objetos vem crescendo nos últimos tempos, tanto que segundo o representante do Gartner Group em 2004 afirmou que dentre as tecnologias que irão sobreviver no mercado nos próximos anos estão Java e agora o .Net, será praticamente impensável produzir aplicações que não sejam orientadas a objetos.

O sucesso em aplicações orientadas a objetos, está diretamente ligada a organização de aplicações em camadas e na utilização dos padrões utilizados pelo mercado.

A organização em camadas é a estratégia para a independência entre os componentes e assim se atingir os objetivos de eficiência, escalabilidade, reutilização, e facilidade de manutenção.

O termo camada significa uma separação da aplicação em si em partes isoladas, cada uma com suas respectivas responsabilidades como demonstrado ao longo deste capítulo.

3.1 Camadas

3.1.1 Aplicação de uma camada

Antigamente na era do computador pessoal, uma aplicação era desenvolvida para ser usada em uma única máquina, esta aplicação geralmente tinha todas as funcionalidades em um único núcleo com uma imensa quantidade de linhas de código, o que tornava a manutenção do software penosa, a interação do usuário, lógica de negócio, acesso aos dados estava tudo presente em um mesmo lugar, como ilustrado na Figura 1.



Figura 1 - Aplicação de uma camada.

3.1.2 Aplicação de duas camadas

A grande necessidade de distribuir a lógica de acesso a dados aos vários usuários simultâneos de uma mesma aplicação, fez com que surgissem as aplicações em duas camadas.

Na estrutura de duas camadas foi colocado o acesso a base de dados em uma máquina específica, separando ela das máquinas que apenas executam a aplicação, neste modelo de duas camadas as estações dos clientes contêm todas a lógicas das aplicações, o que ai gera um grande problema pois para cada nova versão da mesma aplicação, se o sistema estiver instalado em 200 máquinas clientes, o mesmo deverá novamente ser instalado 200 vezes a fim de que se aplique a atualização do software. A Figura 2 ilustra o MVC de duas camadas.

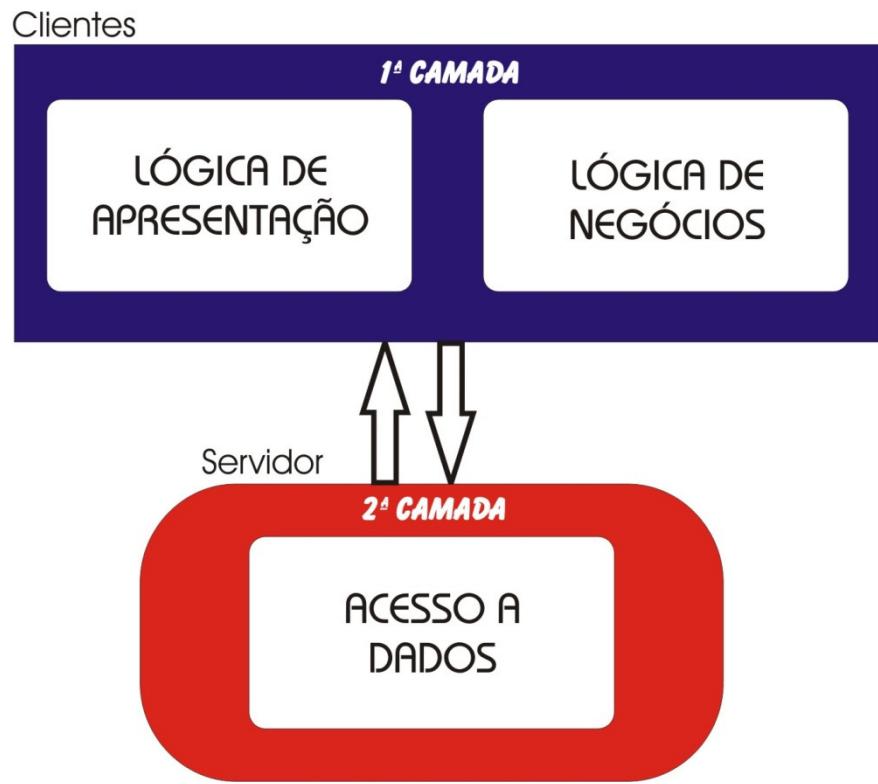


Figura 2 - Aplicação com duas camadas.

3.1.3 Aplicação de três camadas

Com o grande crescimento da internet houve um movimento para separar a lógica de negócio da interface com o usuário, a idéia é que os usuários da *web* possam acessar as mesmas aplicações sem ter que instalar estas aplicações em suas máquinas, os clientes que antes eram considerados ricos e agora são considerados pobres (clientes ricos eram os clientes que possuíam em sua máquina a aplicação toda instalada, e hoje os clientes pobres significam que eles não precisam mais ter nenhuma aplicação instalada em sua máquina para ter acesso a aplicação).

Neste modelo a aplicação é movida junto ao servidor e um navegador web é usado como um cliente pobre, a aplicação é executada em servidores web que geram o código HTML para ser exibido no cliente.

No modelo de três camadas a lógica de apresentação está separada em sua própria camada lógica e física, a separação em camadas lógicas torna as aplicações mais flexíveis, permitindo que as partes possam ser modificadas de forma independente, as funcionalidades da camada de negócio podem ser divididas em classes e as mesmas podem ser agrupadas em pacotes ou componentes reduzindo assim as dependências entre as classes e pacotes,

podem também ser reutilizadas por diferentes partes da aplicação e até mesmo por outras aplicações, a aplicação de três camadas transformou assim a arquitetura padrão para sistemas corporativos com base na web, a Figura 3 ilustra o MVC de três camadas.

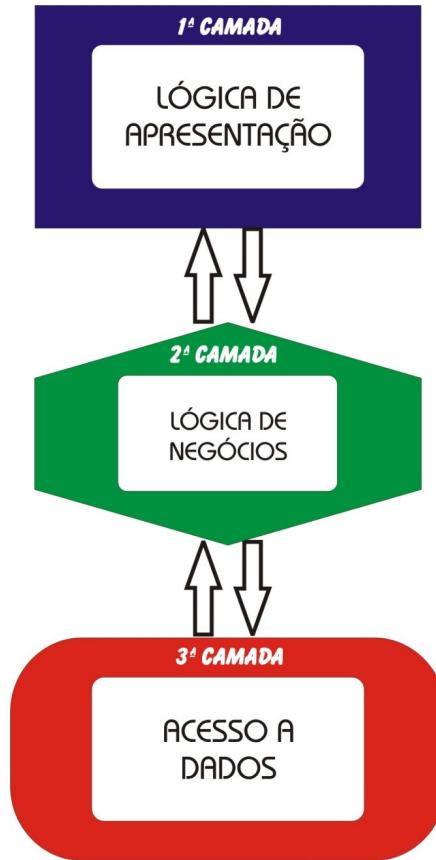


Figura 3 - Aplicação MVC de três camadas.

3.2 O Padrão MVC

No capítulo 3.1 este trabalho procurou descrever um breve histórico para que se tenha conhecimento da aplicação em camadas, para poder demonstrar o padrão MVC de onde ele veio e o porquê de sua utilização.

O padrão MVC adota uma maneira de repartir a funcionalidade envolvida na manutenção e apresentação dos dados de uma aplicação.

A arquitetura MVC não é recente apesar de sua importância atualmente, ainda mais após o surgimento da Internet este padrão ganhou força, ela foi originalmente implementada para mapear as tarefas tradicionais de entrada, processamento e saída para um modelo de interação com o usuário, utilizando o

padrão MVC fica fácil absorver os conceitos no domínio de aplicações web multicamadas.

O modelo de três camadas físicas (*3 tier*) divide uma aplicação de maneira que a lógica de negócio resida no meio das três camadas físicas, quer dizer que, que ela é chamada de camada física intermediaria ou camada física de negócio, a grande parte do código implementado reside na camada de apresentação e na camada de negócio.

O MVC foi um dos primeiros padrões identificados, foi criado pela comunidade Smalltalk em 1979 e mesmo após todos estes anos ele ainda é muito aplicado nos dias de hoje em aplicações interativas que necessitam de interfaces flexíveis, o MVC está dividido em três camadas, que são:

Modelo (*Model*):

Estrutura lógica dos dados;

Visão (*View*):

Coleção de classes que dão suporte a interface com o usuário;

Controle (*Controller*):

Realiza a comunicação entre o modelo e a visualização.

3.2.1 Objetivos

Os objetivos deste padrão são separar dados ou lógica de negócios (*model*) da interface do usuário da interface do usuário (*view*) e do fluxo da aplicação (*controller*), a arquitetura permite que uma mensagem lógica de negócios possa ser acessada e visualizada através de varias interfaces, na arquitetura MVC a lógica de negócios não sabe quantas nem quais interfaces com o usuário estão sendo exibidas, o modelo MVC está preocupado em separar a informação da apresentação.

3.2.2 Características

A separação entre dados, apresentação e controlador, que gerencia as relações entre o modelo e a apresentação, separa também a lógica da apresentação, possui reusabilidade, responsabilidades mais definidas, reduz esforços na camada de apresentação, é uma metodologia ou *padrão de projeto* tendo assim um projeto com seus objetos bem definidos e distribuídos,

relacionando a interface do usuário aos seus dados, o MVC foi um dos primeiros padrões reconhecidos, utilizado inicialmente em SMALLTALK-80, bem como utilizado pelo GOF como exemplo de padrão.

3.3 Camadas no MVC

3.3.1 Camada de Apresentação

Um componente de visualização renderiza o conteúdo de uma parte restrita do modelo e envia para o controlador os eventos de ações do usuário, acessa também os dados do modelo através do controlador e define como esses dados deverão ser apresentados, não se preocupa de que maneira a informação é obtida ela apenas exibe a informação, isto é, inclui os elementos de exibição no cliente, como HTML, ASP, XML, *applets*, é a camada de interface com o usuário e é também utilizada para receber a entrada de dados e assim apresentar seus resultados.

3.3.2 Camada Lógica da Aplicação

No padrão MVC o modelo representa os dados da aplicação e as regras de negócio que governa o acesso e a atualização dos dados, ele também mantém o estado de persistência do negócio e fornece para o controlador a capacidade de acessar as funcionalidades da aplicação escondidas (encapsuladas) pelo próprio modelo, pode ser definida como o coração ou o cérebro da aplicação, é responsável por tudo o que a aplicação vai processar, modela os dados e o comportamento por trás do processo de negócios, se preocupa apenas com o armazenamento e geração dos dados, é um encapsulamento de dados e de comportamento independente da apresentação.

3.3.3 Camada de Controle da Aplicação

O controlador especifica o comportamento da aplicação, é responsável pela interpretação das ações do usuário e por enviar tais ações para o modelo, em um cliente de aplicações *web* essas ações poderiam ser cliques em botões ou seleções de menus ou a entrada de dados, nas ações realizadas pelo modelo estão inclusas as ações de ativar processos de negócio ou modificar o estado do modelo, de acordo com a ação do usuário e no resultado do processamento do modelo, o controlador seleciona a visualização a ser utilizada e exibida como

resposta a ação ou solicitação do usuário, normalmente existe um controlador para cada grupo de funcionalidades relacionadas, determinando o fluxo da apresentação, servindo como uma camada intermediaria entre a camada de visualização com a camada da lógica, ou seja ela controla e mapeia as ações, a Figura 4 ilustra o padrão MVC.

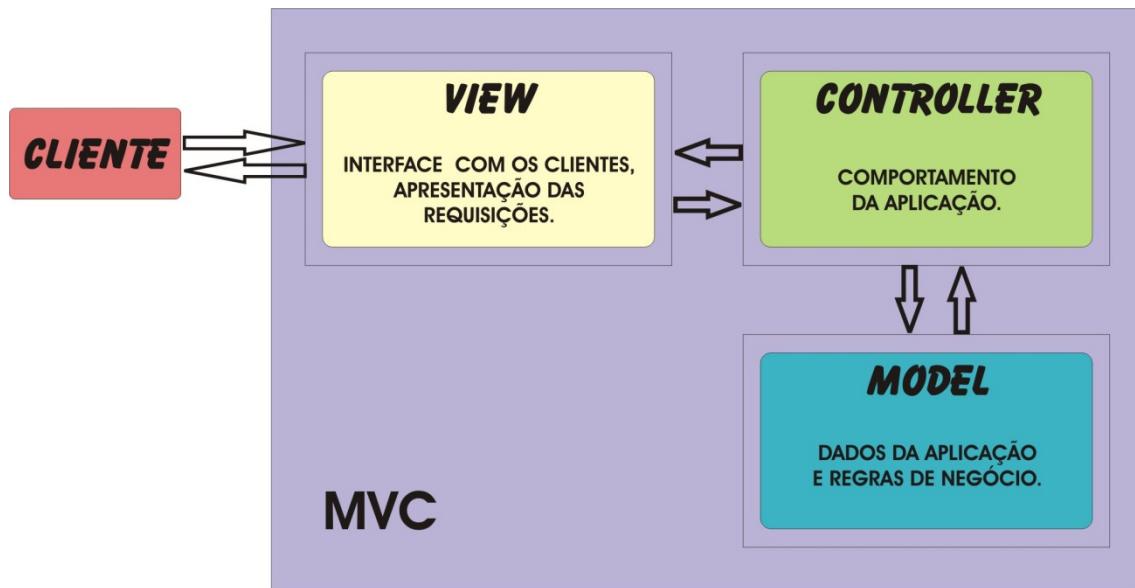


Figura 4 – Arquitetura do padrão MVC.

3.4 *Vantagens e desvantagens do MVC*

3.4.1 Vantagens

O MVC gerencia múltiplos visualizadores usando apenas um modelo, é fácil manter, testar e atualizar sistemas múltiplos, e muito simples na adição de novos clientes, apenas incluindo aos clientes seus visualizadores e controles, a aplicação pode ser escalável, torna-se possível o desenvolvimento em paralelo ao modelo, visualizador e controle, pois são camadas independentes.

3.4.2 Desvantagens

O MVC requer um tempo maior para analisar e modelar o sistema, exigindo desenvolvedores especializados no assunto, além de não ser aconselhável para aplicações de pequeno porte.

3.5 Frameworks MVC

O protótipo que mais a frente será mostrado (Capítulo 5), foi desenvolvido com o *framework* JSF 1.2 que cada dia ganha mais adeptos do mundo da plataforma Java por se tratar de um *framework* baseado no padrão de projeto MVC (Capítulo 3), atualmente existem dois estilos de programação para *web* muito discutidos, como diria Gary et al (2007,pag. 3), “*desenvolvimento rápido*” *com um ambiente totalmente visual como o Microsoft ASP.NET* e “*programação aprofundada*” que *exige muitas linhas de códigos-fonte para dar suporte a um backend de alto desempenho como o JAVA EE*.

Existem vários *frameworks* para a utilização do MVC, neste tópico é discutido a fundo o *framework* JSF 1.2 (Capítulo 4), mostrando o porque de sua utilização na criação de aplicações *web* com o padrão MVC, só para enfatizar este trabalho lembra que existem ainda vários outros *frameworks* como por exemplo, Tapestry, Struts, SpringMVC, Webwork, VRaptor, Mentawai etc.

4 O FRAMEWORK JSF (*Java Server Faces*)

Existem diversos *frameworks* baseados no padrão MVC, neste trabalho foi escolhido o framework JSF 1.2 para o desenvolvimento do protótipo (Capítulo 5). As razões desta escolha foram as seguintes:

- JSF 1.2 é um padrão adotado pelo mercado, presente na especificação Java EE 5.0.
- Baixa curva de aprendizagem da equipe de desenvolvimento, com componentes prontos, vários pontos de extensão para validadores, conversores e eventos.
- Acessibilidade a novas tecnologias de mercado como, *browsers*, celulares, PDAs, reusabilidade e componentes extensíveis.
- Integração com diversos outros *frameworks* do mercado, tais como JBoss Seam, Spring e muitos outros. No tocante aos IDEs quase todos, dentre os mais conhecidos, suportam o JSF 1.2, como por exemplo o Eclipse e, Netbeans.

Além de todas estas características, o JSF ainda possui uma imensa quantidade de ferramentas adicionais e visuais comentadas ao longo deste capítulo. O motivo mais forte é o *custo de desenvolvimento*, isto é, a baixa curva de aprendizagem aliada a imensa quantidade de ferramentas adicionais que proporcionam ganho de produtividade em projetos, bem como qualidade, o que se reflete no custo total de desenvolvimento do projeto.

4.1 MVC no JSF

4.1.1 Controle:

A camada controle é composta por: um *servlet* chamado FacesServlet, por um conjunto de manipuladores de ações, arquivos de configuração e observadores de eventos. O FacesServlet é responsável pelas requisições da *web* e seu redirecionamento para o modelo, bem como pelo envio da resposta.

Os arquivos de configuração são responsáveis por realizar mapeamentos e associações de ações e pela definição das regras de navegação.

Os manipuladores de eventos se responsabilizam por receber os dados, que vem da camada de visualização, acessar o modelo e então devolver o resultado para o FacesServlet.

4.1.2 Modelo

O modelo representa os objetos de negócio e executa uma lógica de negócio ao receber os dados vindos da camada de visualização.

4.1.3 Visualização

A visualização é composta por *component trees ou UI (User Interface)*, tornando possível unir um componente ao outro para formar interfaces com um grau de complexidade maior. Na figura 5, pode ser verificado como funciona a tecnologia JSF para as UIs.

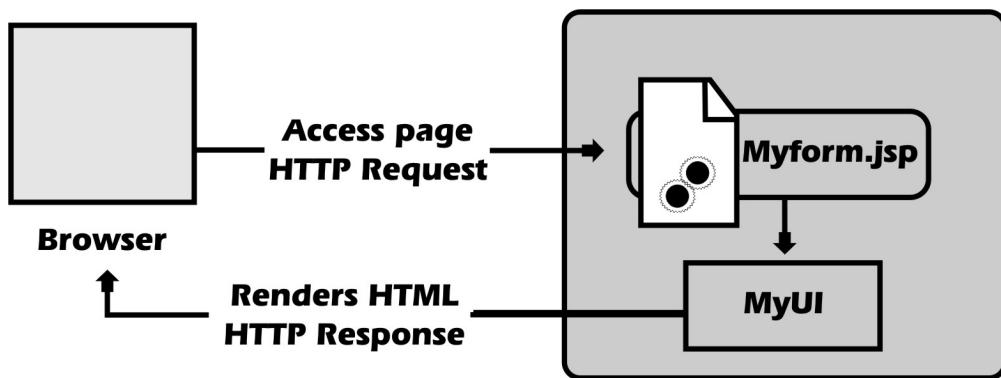


Figura 5 – Tecnologia JSF para UIs.

4.2 Ciclo de vida do JSF.

O ciclo de vida no JSF estão divididas em seis fases que são:

Restaurar visão:

Quando um usuário fornece dados ilegais ele mantém a pagina atual reexibindo o formulário atual.

Aplicar valores de requisição:

Faz iterações sobre os objetos dos componentes, verifica qual objeto pertence ao objeto em questão e armazena-o.

Processar validações:

Nesta fase a *string* passada é convertida para valores locais, que pode ser um objeto de qualquer tipo, quando passa pela validação segue seu fluxo normal, quando não passa segue para a fase renderizar.

Atualizar valores do modelo:

Após a validação e conversão, os valores locais armazenados são utilizados para atualizar os *beans* vinculados aos componentes.

Invocar aplicação:

É quando é invocado o método *action* do botão ou *link* que causou o envio do formulário, ele retorna uma string para o *handler* de navegação, e o mesmo se encarrega de verificar qual pagina deve ser carregada.

Renderizar resposta:

A fase renderizar é responsável por codificar a resposta e envia-a para o navegador, e a partir daí o ciclo se renova novamente.

A Figura 6 ilustra de maneira clara como funciona o ciclo de vida do JSF.

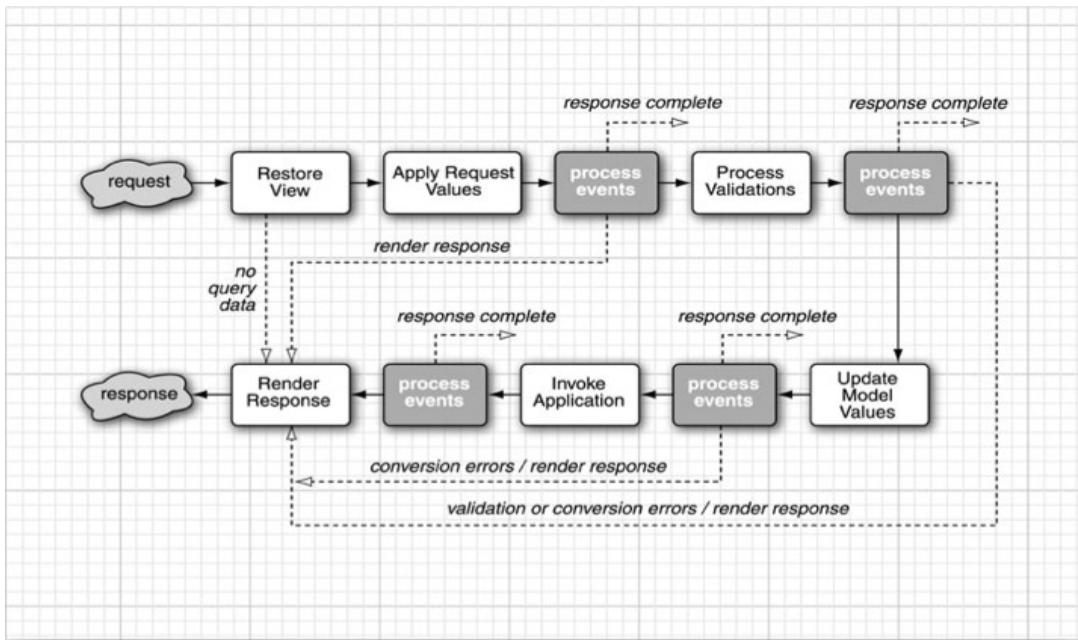


Figura 6 – Ciclo de vida do JSF. Fonte: Gary et al(2007,p 29)

4.3 Características

As características do JSF 1.2 permitem que o desenvolvedor crie UIs através de um conjunto de componentes UIs pré-definidos. Além disso:

- Fornece um conjunto de tags JSP para acessar os componentes.
- Reutiliza componentes da página.
- Associa os eventos do lado cliente com os manipuladores dos eventos do lado do servidor, os componentes de entrada possuem um valor local representando o estado no lado servidor.
- Fornece separação de funções que envolvem a construção de aplicações *web*.
- Suporte a internacionalização e acessibilidade.
- Um conjunto padrão de componentes de interface de usuário que possibilitam validação padronizada.

- Duas bibliotecas de etiquetas (*tag libraries*) especiais do JSP (*Java Server Pages*) para expressar a interface do Java Server Faces dentro de uma página JSP.
- Um modelo de eventos do lado servidor (*server-side event model*).
- Gerência de estados,
- Gerenciamento de Beans (Java Beans criados por meio da injeção de dependência ou *dependency injection*).
- Linguagem de Expressão Unificada (*Unified Expression Language*) para JSP 2.0 e JSF 1.2.

4.4 Tags do JSF

Nesta seção são mostradas as principais tags do JSF, o trabalho procura mostrar um conjunto de tags fundamentais para o desenvolvimento de qualquer projeto utilizando este *framework*, não foi mostrado aqui todas as *tags* do JSF para se ter uma referencia completa de todas as *tags* do JSF, no capítulo 7 consta o *link* para o download completo de todas as *tags* do JSF, as Tabelas 1, 2, 3 ilustram as *tags* mais importantes bem como para que cada uma serve.

Tabela 1- Tags Fundamentais JSF.

Tag	Descrição
<code>view</code>	Cria visão de nível mais alto.
<code>subview</code>	Cria um subvisao de uma visão.
<code>attribute</code>	Define um atributo Key/Value em seu componente de origem (param)
<code>param</code>	Adiciona um parâmetro subordinado a seu componente de origem.
<code>actionListener</code>	Adiciona um listener de ação a um componente.
<code>setPropertyActionListener</code>	Adiciona um listener de ação que define uma propriedade.
<code>phaseListener</code>	Adicionar um listener de fase a visão de origem.
<code>converter</code>	Adiciona um conversor arbitrário a um componente.
<code>converterDateTime</code>	Adiciona um conversor de data e hora a um componente.
<code>convertNumber</code>	Adiciona um conversor numérico a um componente.
<code>validator</code>	Adiciona um validador a um componente.
<code>loadBundle</code>	Carrega um resource bundle; armazena propriedades como um Map.

<code>selectItems</code>	Especifica itens para um componente selectOne ou selectMany.
<code>selectItem</code>	Especifica apenas um item para um componente select one ou many.

Tabela 2 - Tags html do JSF.

Tag	Descrição
<code>form</code>	Formulário Html.
<code>input</code>	Entrada de texto de uma única linha.
<code>inputSecret</code>	Entrada de texto em forma de senha.
<code>outputText</code>	Saída de texto de uma única linha.
<code>commandButton</code>	Botões: Submit, reset ou pushbutton.
<code>message</code>	Exibe a mensagem mais recente de um componente.
<code>graphicImage</code>	Exibe uma imagem.
<code>selectOneListbox</code>	Caixa de listagem para seleção de um único item.
<code>selectOneMenu</code>	Menu para seleção de um único item.
<code>selectOneRadio</code>	Conjunto de botões de radio.
<code>selectManyCheckbox</code>	Conjunto de caixas de verificação.
<code>selectManyMenu</code>	Menu para seleção de vários itens.
<code>panelGrid</code>	Tabela HTML.
<code>panelGroup</code>	Dois ou mais componentes dispostos com um só.
<code>dataTable</code>	Controle de tabela com muitos recursos.
<code>column</code>	Coluna em um dataTable.

Tabela 3 - Atributos Básicos Comuns a quase todos os componentes.

Atributos	Tipos de Componentes	Descrição
<code>id</code>	A(25)	Identificador para um componente.
<code>binding</code>	A(25)	Faz o <i>link</i> deste componente com a propriedade de um backing bean.
<code>rendered</code>	A(25)	Atributo booleano; se for <i>false</i> (falso) interrompe a renderização.
<code>styleClass</code>	A(23)	Nome da classe CSS (<i>Cascading Style Sheet</i>).
<code>value</code>	I,O,C (19)	Valor de um componente, Geralmente é uma expressão de atribuição de valor.
<code>valueChangeListener</code>	I(11)	Expressão de atribuição de método que responde a mudança de valores.
<code>converter</code>	O(15)	Nome da classe do conversor.

<code>validator</code>	I(11)	Nome da classe do validador que é criado e vinculado a um componente.
<code>required</code>	I(11)	Atributo booleano; se for true exige a inserção de um valor no campo associado.
<code>converterMessage</code> , <code>validatorMessage</code> , <code>validatorMessage</code>	I(11)	Mensagem customizada a ser exibida quando ocorre erro de conversão ou de validação, ou quando falta um <i>input</i> exigido.
A=TODOS, I=INPUT, O=OUTPUT, C=COMANDOS, (N)= NUMERO DE TAGS COM O ATRIBUTO.		

5 PROTOTIPO MVC

Como o objetivo principal deste trabalho é desenvolver um protótipo baseado no padrão MVC, foi escolhido o *framework* JSF, para o qual existem diversas ferramentas adicionais que auxiliam no desenvolvimento de aplicações *web*, com baixa curva de aprendizagem, além de ser um padrão de mercado presente na especificação do Java EE 5.0.

5.1 *Descrição do ambiente desenvolvimento*

Para o desenvolvimento de aplicações *web* com o *framework* JSF 1.2, foi utilizado o IDE NETBEANS 6.1, pois se trata de um produto que oferece um ambiente todo configurado para o desenvolvimento de aplicações *web*, não sendo necessário configurar ou instalar servidores, *plugins* etc.. Este IDE conta com dois servidores: Apache Tomcat 6.0.16, e Glassfish 1.2, além de uma implementação do *framework* JSF 1.2 totalmente visual, que traz consigo a tag `<webuijsf>` integrado com as tags padrão `<h:>` e `<f:>` do framework JSF 1.2

No desenvolvimento do protótipo deste trabalho foram utilizados os seguintes recursos:

- Notebook: Processador dual core 1.86 GHz, 2GBytes de memória RAM.
- Virtual Machine: Java 5
- Framework: JSF 1.2
- IDE: NETBEANS 6.1
- Servidor: Glassfish 1.2

- Banco de dados: MySql 5.0

5.2 Descrição do Protótipo

O objetivo principal deste trabalho é a construção de um protótipo de uma aplicação *web* para um mini *call center* para atendimentos telefônicos de empresas de ônibus, utilizando o *framework* JSF 1.2.

Nos dias de hoje a utilização de sistemas *web* vem crescendo constantemente, pela facilidade de acesso a todas as pessoas que precisam deste sistema seja em seu em seu local de trabalho ou em qualquer outro local, bastando apenas que tenha um browser instalado e conexão com a internet.

No protótipo que se segue é mostrado como é possível num mesmo sistema via *web*, tanto o cliente como a empresa prestadora de serviços utilizarem o mesmo sistema, cada um em seu respectivo local de trabalho, ou em qualquer outro lugar como mencionado acima.

O protótipo desenvolvido tem por objetivos oferecer as seguintes funções:

Manipulação dos dados envolvendo operações com o banco de dados como *insert*, *delete*, *update* e *select*, bem como mostrar o funcionamento do JSF em seu interior, procura também mostrar como o padrão MVC esta dividido dentro deste *framework*.

5.2.1 Fluxo de Navegação

Neste tópico foi ilustrado (Figura 7) como o fluxo de navegação está configurado, Através da imagem fica fácil a interpretação do esquema de navegação do protótipo desenvolvido.

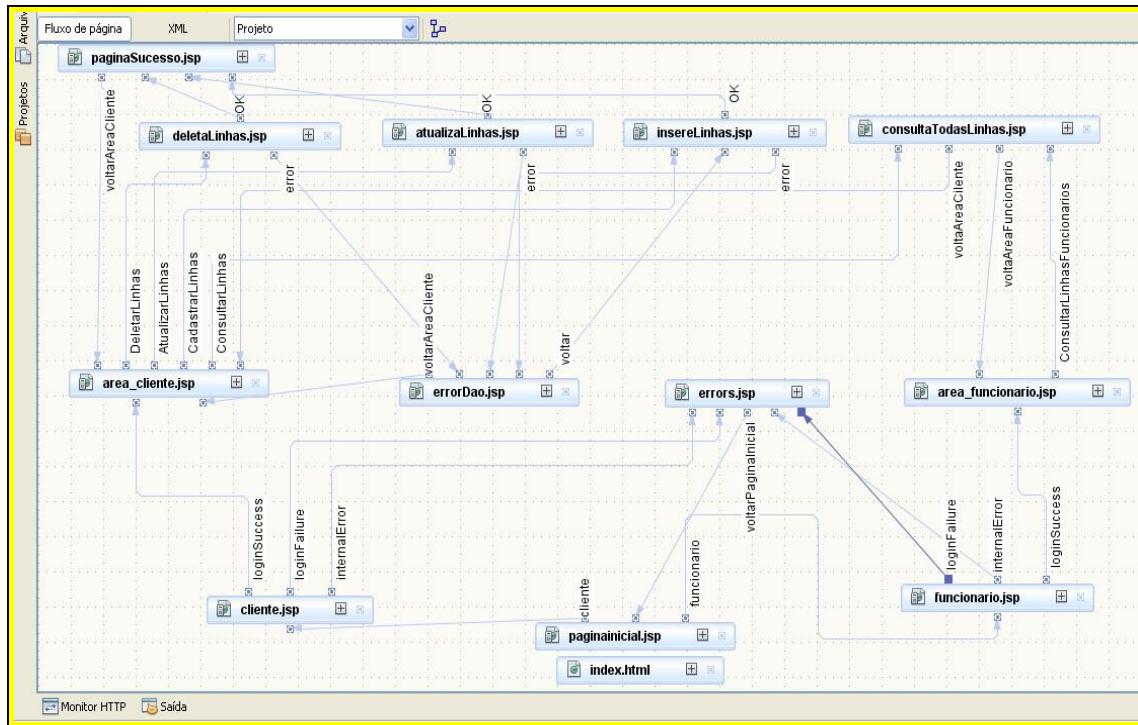


Figura 7 - Fluxo de navegação do protótipo desenvolvido.

Como ilustrado na Figura 7 este é o fluxo de navegação do protótipo desenvolvido ele representa o nosso arquivo faces-config.xml, como ilustrado na figura 8.

```
<?xml version="1.0"?>
<faces-config xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-facesconfig_1_2.xsd"
    version="1.2">

<managed-bean>
    <managed-bean-name>user</managed-bean-name>
    <managed-bean-class>br.com.MiniCallCenterTcc.UserBean</managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
<managed-bean>
    <managed-bean-name>Nomes</managed-bean-name>
    <managed-bean-class>br.com.MiniCallCenterTcc.Nome</managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
<managed-bean>
    <managed-bean-name>DAO</managed-bean-name>
    <managed-bean-class>br.com.MiniCallCenterTcc.MiniCallCenterDao</managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
<managed-bean>
    <managed-bean-name>DAOK</managed-bean-name>
    <managed-bean-class>br.com.MiniCallCenterTcc.MiniCallCenterDao</managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
<managed-bean>
```

Figura 8 – Fragmento do arquivo de configuração faces-config.xml

5.2.2 Diagrama de Classes

Neste capítulo através da Figura 9 este trabalho mostra como foram criadas as classes deste protótipo, bem como as relações entre elas, na verdade como mostrado nos capítulos que se seguem estas são nossas classes *beans*.

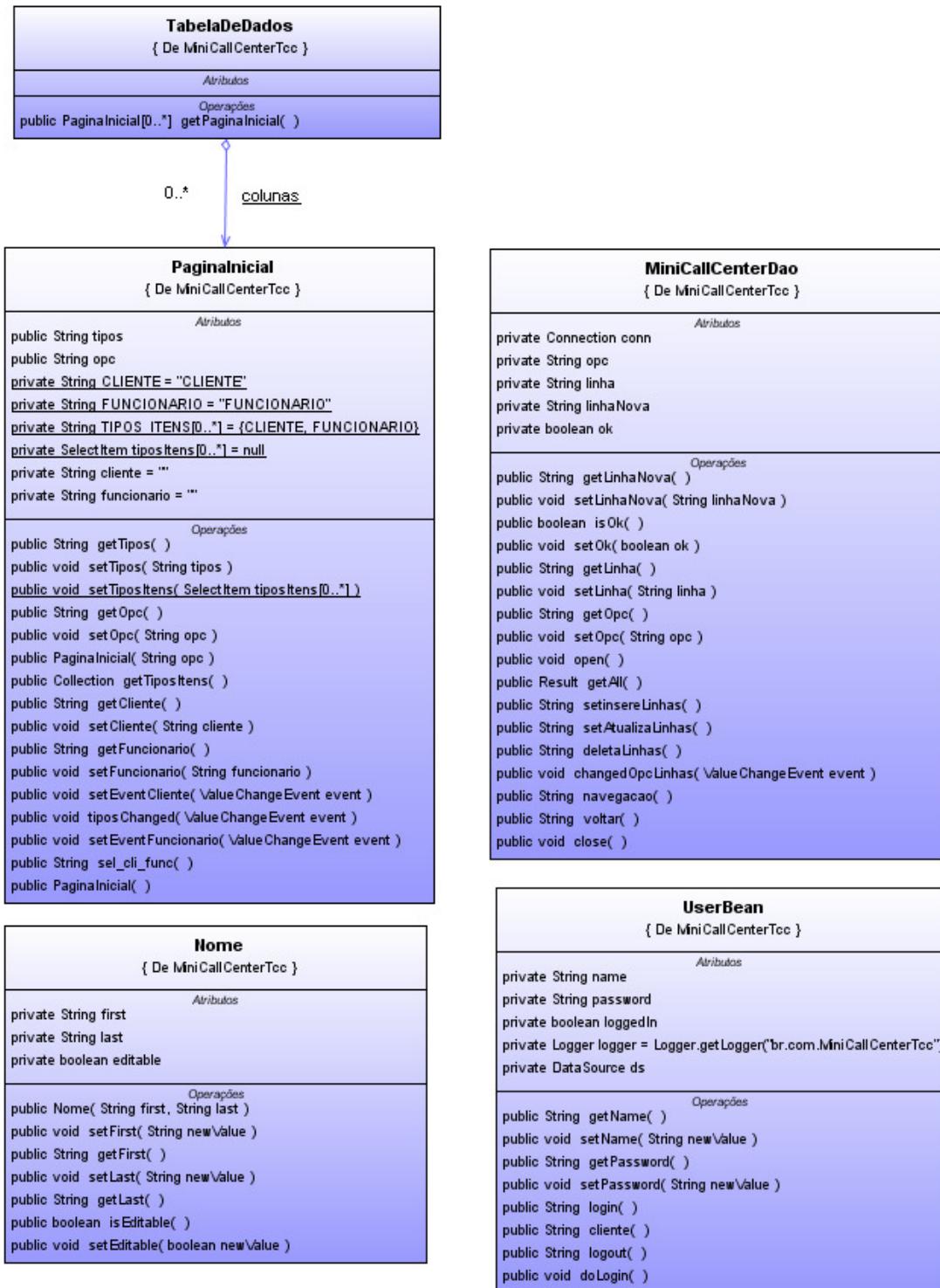


Figura 9 – Diagrama de Classes do Protótipo Desenvolvido.

5.2.3 Área Pública

Ao acessar a pagina inicial, uma tela será apresentada pedindo que o usuário escolha se ele é um cliente ou um funcionário da empresa.

5.2.4 Área Cliente

O usuário do sistema será direcionado a página de clientes uma vez em que o mesmo tenha escolhido a opção clientes na área publica do sistema, abrirá a ele uma página para que seja feito o seu *login*.

Quando o cliente digitar seu *login* e sua senha, serão validados os dados inseridos, caso sejam aceitos uma nova página se abrirá contendo um menu com as seguintes opções:

- **Consulta:** a consulta permite ao cliente consultar todas as linhas cadastradas para sua empresa
- **Inserção:** a inserção permite ao cliente que ele cadastre novas linhas para sua empresa.
- **Deleção:** a deleção permite ao cliente que ele delete as linhas não mais utilizadas pelo cliente.
- **Atualização:** a atualização permite ao cliente atualizar uma linha cadastrada.

Caso o cliente forneça um *login* ou senha não cadastrados ou errados, o mesmo será direcionado a página de erro pedindo que o mesmo retorne a página anterior, e entre com os dados corretamente.

5.2.5 Área Funcionários

O usuário do sistema será direcionado a página de funcionários uma vez em que o mesmo tenha escolhido a opção funcionários na área pública do sistema será aberta uma página para que seja feito o seu *login*.

Aos usuários gerais do sistema é apenas permitido consultar o sistema, neste projeto o usuário final se trata de funcionários da empresa, responsáveis por passar a informações de nosso cliente aos clientes dos mesmos (usuários das empresas cadastradas).

Caso o cliente forneça um *login* ou senha não cadastrados ou errados, o mesmo será direcionado a página de erro pedindo que o mesmo retorne a página anterior, e entre com os dados corretamente.

5.3 Desenvolvimento do Protótipo

Esta seção pretende mostrar de maneira clara o MVC dentro do ambiente de desenvolvimento, bem como as particularidades do *framework* JSF 1.2.

5.3.1 MVC no Protótipo

Na Figura 10 esta ilustrado como o MVC está dividido neste projeto, de maneira que o *view* representa as páginas.jsp, que o controller esta representado nos arquivos de configuração, é nestes arquivos que se encontram os *servlets*, bem como os arquivos responsáveis pela navegação das páginas, e por último pode-se analisar o *model* que é onde se encontra toda a lógica de negócios de nosso protótipo, é nele que estão nossas classes *Beans* para que as mesmas processem as regras do negócio.

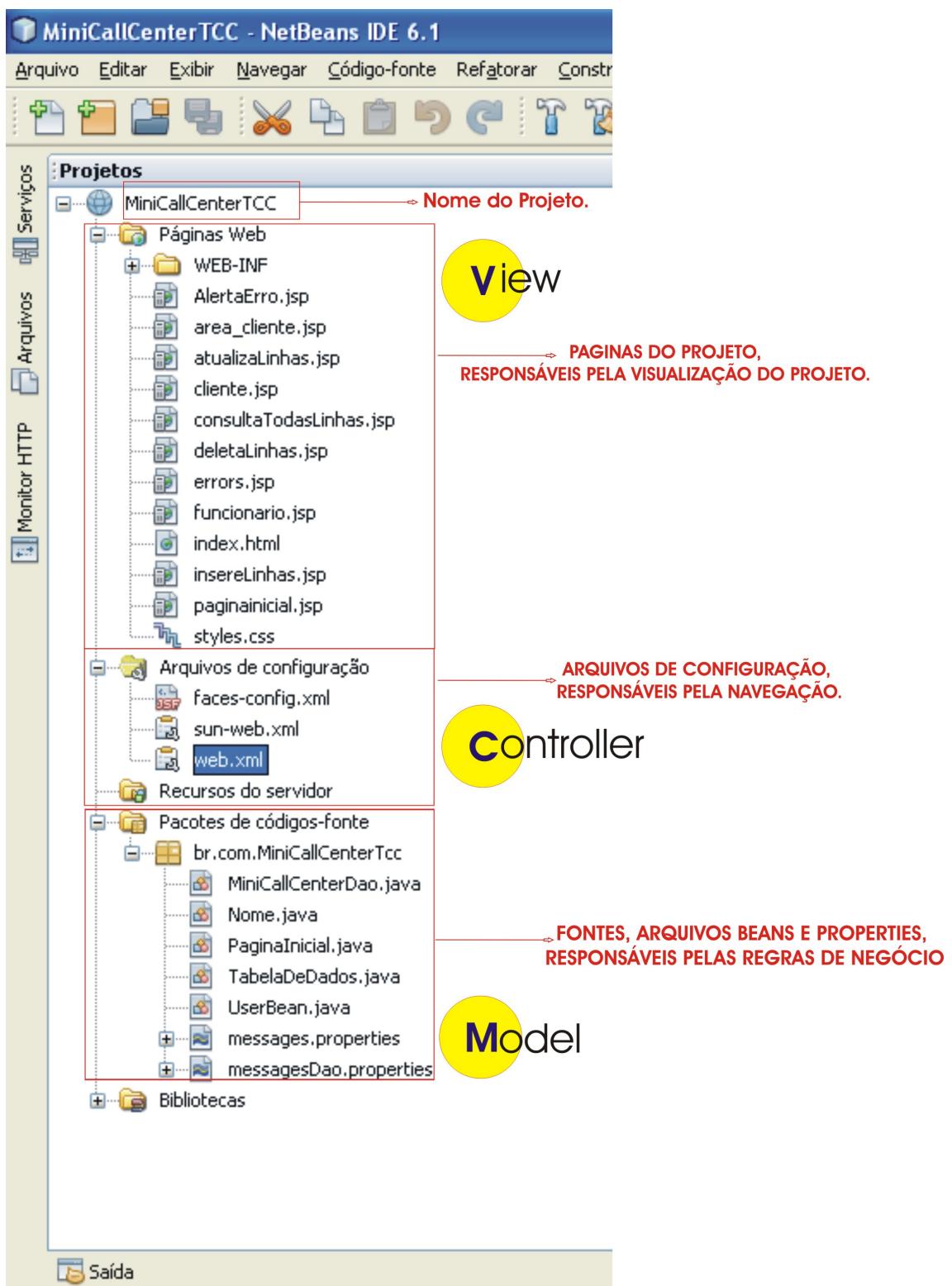


Figura 10 - MVC no framework JSF 1.2

5.3.2 Desenvolvimento da Área Pública

Como foi utilizado o JSF, em todas as páginas deste projeto as seguintes tags serão incluídas:

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
```

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
```

The diagram illustrates the development process of the initial page. It consists of five numbered sections (4, 5) showing screenshots and their corresponding code snippets.

- Section 4:** Shows a screenshot of a Windows Internet Explorer browser window titled "BEM VINDO AO SISTEMA CALLCENTERTCC! - Windows Internet Explorer". The URL is "http://localhost:8080/MiniCallCenterTcc/paginainicial.faces". The page content includes a title bar with "BEM VINDO AO SISTEMA CALLCENTERTCC!" and a menu bar with "Arquivo", "Editar", "Exibir", "Favoritos", "Ferramentas", and "Ajuda". A red box highlights the title bar. To the right, under the heading "TAGS JSF 1.2", is the code snippet:

```
<title><h:outputText value="#{msgs.title}" /></title>
<h:outputText value="#{msgs.pagini}" />
```
- Section 5:** Shows a screenshot of the application interface with the title "BEM VINDO AO SISTEMA CALLCENTERTCC!". Below it, a message reads "POR FAVOR SELECIONE UMA DAS OPÇÕES ABAIXO:" followed by two radio buttons: "CLIENTES" and "FUNCIONARIOS", and a "Ir Pagina" button. A red box highlights the message. To the right, under the heading "messages.properties", is the code snippet:

```
title=BEM VINDO AO SISTEMA CALLCENTERTCC!
pagini=POR FAVOR SELECIONE UMA DAS OPÇÕES ABAIXO:
```
- Section 3:** Shows a screenshot of the "CONFIGURANDO AS MENSAGENS PADROES NO FACES-CONFIG.XML" section. It contains the XML configuration code:

```
<application>
  <resource-bundle>
    <base-name>br.com.MiniCallCenterTcc.messages</base-name>
    <var>msgs</var>
  </resource-bundle>
</application>
```

Figura 11 - Desenvolvimento da página inicial.

Como ilustrado na Figura 11, o quadro 1, possui a seguinte tag `<title><h:outputText value="#{msgs.title}" /></title>`, e ela é responsável por imprimir o título no *browser* da aplicação mostrado no quadro 4, ainda no mesmo quadro a seguinte tag `<h:outputText value="#{msgs.pagini}" />` é responsável por imprimir a mensagem do quadro 5 na aplicação, isso só é possível, pois foi definido em nosso arquivo de configuração faces-config.xml, o quadro 3 mostra um trecho do arquivo faces-config.xml mostrando aonde se encontra o arquivo padrão para as mensagens que em nosso caso é o pacote `br.com.MiniCallCenterTcc.messages`, ainda no quadro 3 é possível ver o apelido que foi dado para este arquivo de mensagens, que em nosso caso é `msgs`, já o conteúdo da mensagem passada se encontra no arquivo `messages.properties`, e quando invocado no quadro 1, a tag `value="#{msgs.title}"`, primeiro ele procura no arquivo de configuração o caminho do arquivo `msgs`, e logo após ele vai ate o

arquivo messages.properties e procura pelo nome title, traz a mensagem e a imprime na tela, o mesmo acontece para a segunda mensagem exibida no quadro 5.

POR FAVOR SELECIONE UMA DAS OPÇÕES ABAIXO:

CLIENTES
 FUNCIONARIOS

```

<h:selectOneRadio
    id="home"
    layout="pageDirection" onclick="submit()"
    valueChangeListener="#{pagIni.tiposChanged}"
    required="true"
    requiredMessage="por favor selecione um dos dois" >
    <f:selectItem itemValue="cliente" itemLabel="CLIENTES"/>
    <f:selectItem itemValue="funcionario" itemLabel="FUNCIONARIOS"/>
</h:selectOneRadio>
```

Java Beans Associado : Paginal inicial.java

```

public void tiposChanged(ValueChangeEvent event) throws Exception {
    String ret= (String) event.getNewValue();
    setOpc(ret);
}
```

Declaracão de um managed bean dentro do arquivo faces-config.xml

```

<managed-bean>
    <managed-bean-name>pagIni</managed-bean-name>
    <managed-bean-class>br.com.MiniCallCenterTcc.Paginal inicial</managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
```

Figura 12 – Desenvolvimento da página inicial.

Como ilustrado na Figura 12, o quadro 1 é responsável por imprimir os *radiobuttons* na tela, como pode ser visto existem várias *tags* dentro deste componente, conforme apresentado na seção 4.4. As *tags* mais importantes no quadro 1 são: `<h:selectOneRadio>` que é a *tag* que nos permite imprimir os radios na pagina, *tag* `<f:selectItem >` - representa cada item do radiobutton mostrado, o comando `onclick="submit()"` tem a função de enviar os dados do formulário, através do comando `valueChangeListener="#{pagIni.tiposChanged}"` é possível passar para nosso método dentro do *bean* mostrado no quadro 2 o valor do item que foi selecionado, caso na página seja clicado CLIENTES, onclick envia os dados do formulário para o método `tiposChanged` de nosso *bean*, e o mesmo através do comando `event.getnewValue()` retorna uma *string* com o valor do `itemValue` definido na página jsp, na Figura 13 foi ilustrado o seu funcionamento.

No quadro 3 ilustra como declarar um *bean* no arquivo de configuração, a tag <managed-bean-name> representa o apelido dado a sua classe *bean*, já a tag <managed-bean-class> mostra o caminho completo onde se encontra a sua classe *bean*, assim desta maneira quando invocado a tag valueChangeListener="#{pagIni.tiposChanged}", mostrado no quadro 1, ele sabe que pagIni, representa nossa classe PaginalInicial.java, e que ela se encontra dentro do diretório br.com.MiniCallCenterTcc.PaginalInicial, justamente por que assim foi configurado no arquivo de configuração faces-config.xml.

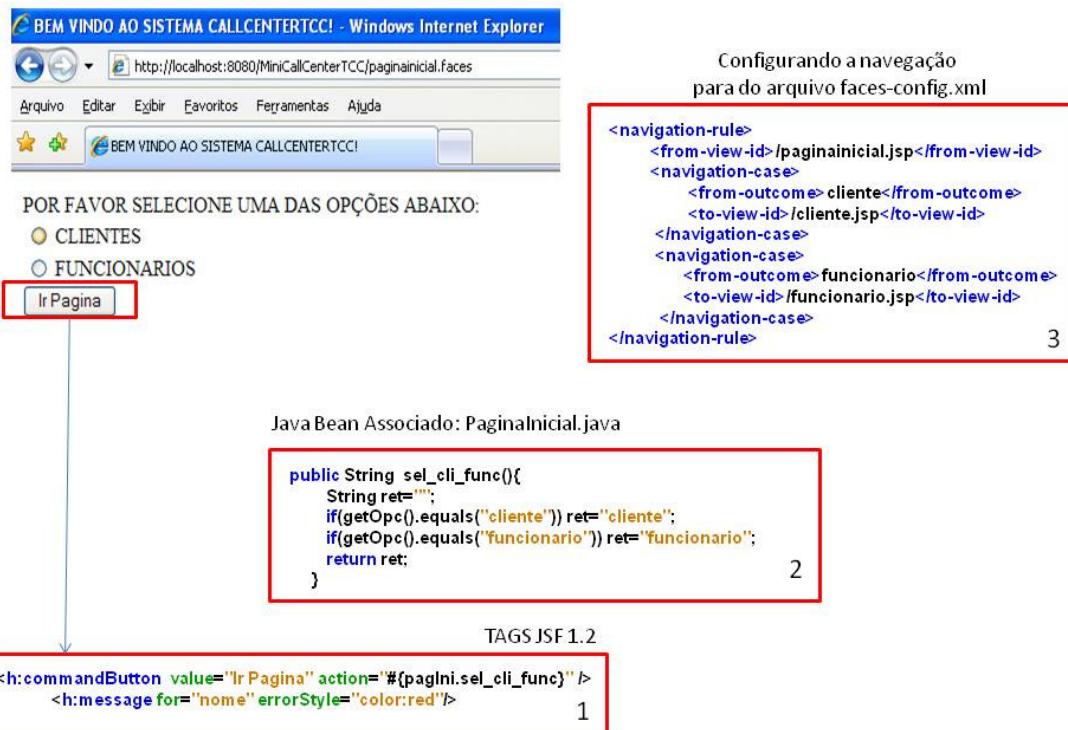


FIGURA 13 - Desenvolvimento da página inicial.

A Figura 13 ilustra no quadro 1 duas *tags* a primeira é <h:commandButton> esta tag imprime um botão em nossa página inicial, e o comando action="#{pag_Ini.sel_cli_func}" tem por finalidade que cada vez que o botão for clicado, ele entra no método sel_cli_func(), de nosso *bean* mostrado no quadro 2, e este método retorna uma *string*, esta *string* é recebida pela aplicação, que procura em nosso arquivo de configuração faces-config.xml mostrado no quadro 3, e de acordo com o retorno, também será a página exibida.

Como já mencionado, o arquivo de navegação é mostrado no quadro 3, a tag <from-view-id>/paginainicial.jsp</from-view-id> mostra a página em que se

encontra a nossa aplicação e a tag <from-outcome>cliente</from-outcome> representa o retorno esperado pelo *bean*, se o retorno for uma *string* do tipo cliente entao a tag <to-view-id>/cliente.jsp</to-view-id> indica que a página que deve ser carregada é a página cliente.jsp, o mesmo ocorre com as outras tags, neste exemplo tem as opções de ir para a página clientes.jsp, funcionarios.jsp.

Existe uma segunda tag <h:message for="nome" errorStyle="color:red"/>, esta tag permite que caso o usuário não selecione nenhuma das opções, o mesmo não tome atitude nenhuma, apenas imprimindo na tela que o usuário deve selecionar uma das opções, como ilustrado na Figura 14, na Figura 12 no quadro 1, foi definida uma tag id="nome" que significa o nome para o componente, já o comando required=true indica que o campo é obrigatório e requiredMessage="por favor selecione um dos dois", esta é a mensagem que será exibida na tela caso o usuário não selecione nenhuma opção o comando for="nome" está ligado a id="nome" e como foi definido que seu valor é obrigatório, a mensagem acima definida será apresentada como ilustra a Figura 14.

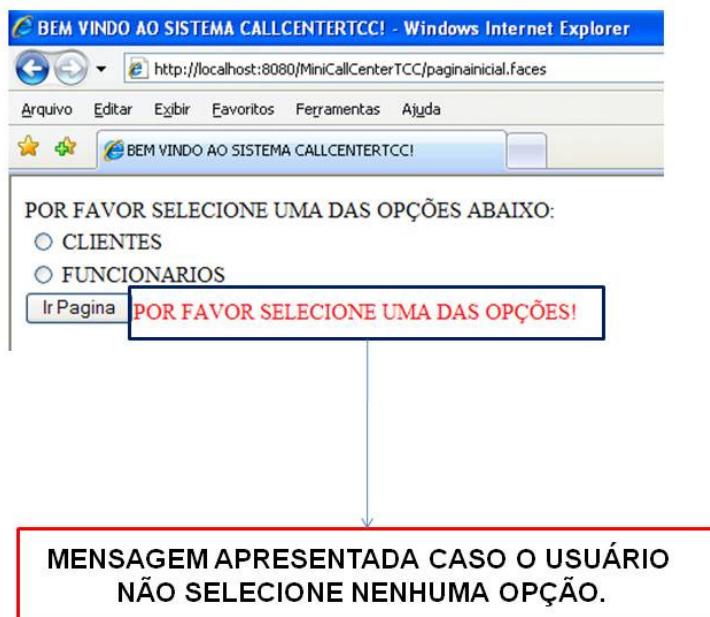
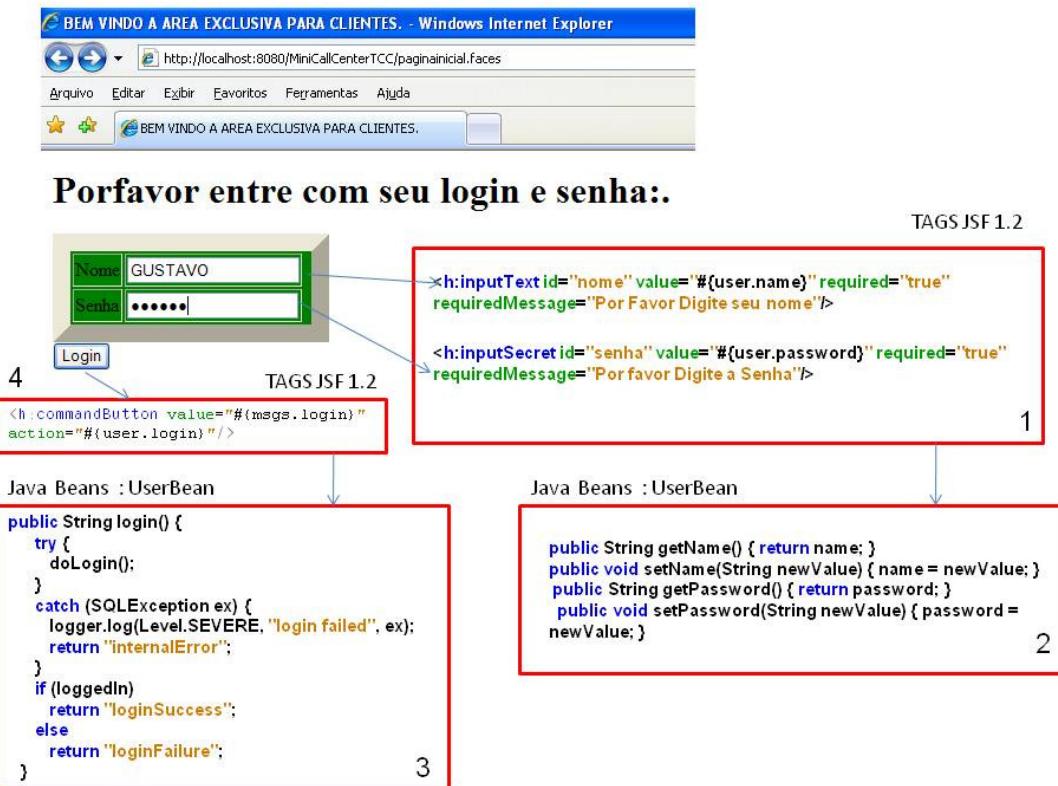


Figura 14 - Desenvolvimento da página inicial.

5.3.3 Área Cliente

A Figura 15 ilustra a área aonde o cliente deve informar seu *login* e senha, as duas *tags* apresentadas neste quadro, quando utilizados em um *input*, e quando esta referenciada por um bean como nos exemplos acima, cada uma representa seu respectivo método *set* na classe bean, a única *tag* em especial nesta figura que é discutida é a *tag* `<h:inputSecret>` que permite que a senha digitada pelo usuário permaneça em segredo.



A validação é feita como ilustrado na Figura 16, ilustrando como nosso protótipo faz a validação da senha e do usuário através de um serviço externo, em nosso protótipo com o banco de dados mysql, após a validação completada, ela retorna, uma das três *strings* mostradas no quadro 3, e a navegação ocorre do mesmo jeito que antes.

Java Beans :UserBean – Acessando Banco de Dados

```
@Resource(name="jdbc/mydb")
private DataSource ds;

public void doLogin() throws SQLEException {
    if (ds == null) throw new SQLEException("No data source");
    Connection conn = ds.getConnection();
    if (conn == null) throw new SQLEException("No connection");
    try {
        PreparedStatement passwordQuery = conn.prepareStatement(
            "SELECT senhacli from clientes WHERE nomecli = ?");
        passwordQuery.setString(1, name);

        ResultSet result = passwordQuery.executeQuery();
        if (!result.next()) return;
        String storedPassword = result.getString("senhaCli");
        loggedin = password.equals(storedPassword.trim());
    }
    finally {
        conn.close();
    }
}
```

Arquivo web.xml configurando
um Pool de Conexão

```
<resource-ref>
<res-ref-name>jdbc/mydb</res-ref-name>
<res-type>javax.sql.DataSource</res-type>
<res-auth>Container</res-auth>
</resource-ref>
```

Figura 16 - Desenvolvimento área cliente.

Como ilustrado na Figura 16, deve-se apenas declarar através da expressão @Resource(name="jdbc/mydb") aonde nossa fonte de dados se encontra, bem como a mesma deve ser adicionada no arquivo web.xml e pronto sua aplicação está pronta para utilizar a nova fonte de dados, para que a fonte de dados esteja configurada, os desenvolvedores não precisam se preocupar com este detalhe, pois para estas configurações praticamente todas as IDEs que suportam o JSF 1.2 já fazem esta parte do serviço automaticamente.

Este trabalho não entra nos méritos de conexões, nem mesmo nos comandos SQL, este trabalho parte da premissa que todos as pessoas que se utilizarem dele tenham os conhecimentos necessários para compreender os comandos mostrados na Figura 16.

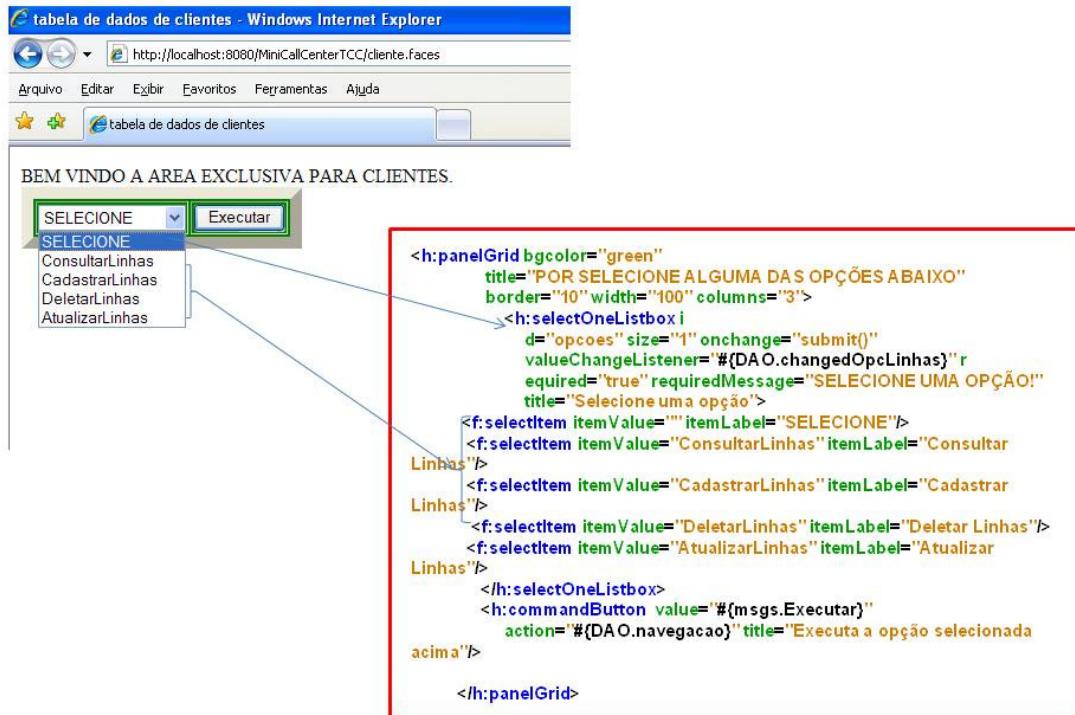


Figura 17 - Desenvolvimento área cliente.

Através da Figura 17 ilustrada, é possível mostrar como utilizar um *listbox* através da tag `<h:selectOneListbox>`, neste trecho de código assim que o usuário selecionar uma opção, o método `DAO.changedOpcLinhas` de nosso bean é acionado, armazenando o valor selecionado pelo usuário, assim que o usuário clicar o botão executar o mesmo capta o valor armazenado e navega para a respectiva página.

Operações com *insert*, *update*, *delete*, *select*:

Cadastro de Novas Linhas

The first screenshot shows a 'BEM VINDO AO SISTEMA CALLCENTERTCC!' page with a 'INserir LINHA' button and a 'Nome da Linha' input field.

The second screenshot shows a 'tabela de dados de clientes' page with a message 'Operação Realizada com sucesso!' and a 'Voltar Pag. Anterior' button.

Figura 18 – Operação de *insert*.

Como ilustrado na Figura 18, quando é inserida uma nova linha, a aplicação navega para a página de sucesso, o mesmo ocorre para as operações de *delete*, *update*, na Figura 19 é ilustrado cada *tag* com seu respectivo *bean*, todas elas navegarão para a página sucesso caso obtenha como resposta o sucesso na operação.

Comandos SQL.

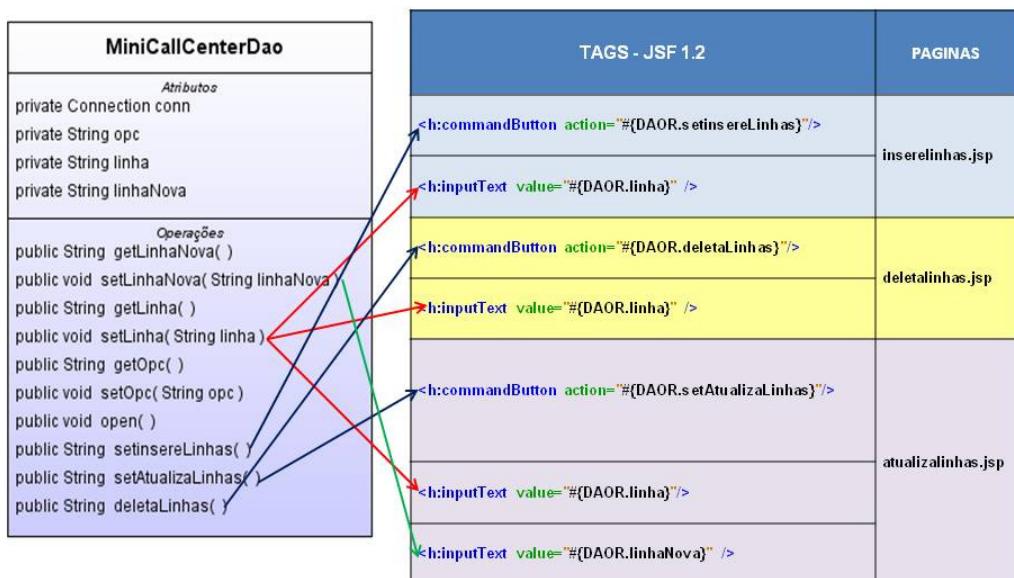


Figura 19 – Bean X páginas.

A Figura 19 ilustra como cada página está ligada ao nosso *bean*, já a navegação, como mostrada em quadros anteriores, ocorre da mesma maneira.

5.3.4 Área Funcionário

Aqui não será detalhado o que já foi discutido anteriormente na área cliente, mas vale ressaltar que, como já mostrado na descrição do protótipo, fica definido que os funcionários desta empresa tem apenas a função de consultar o sistema.

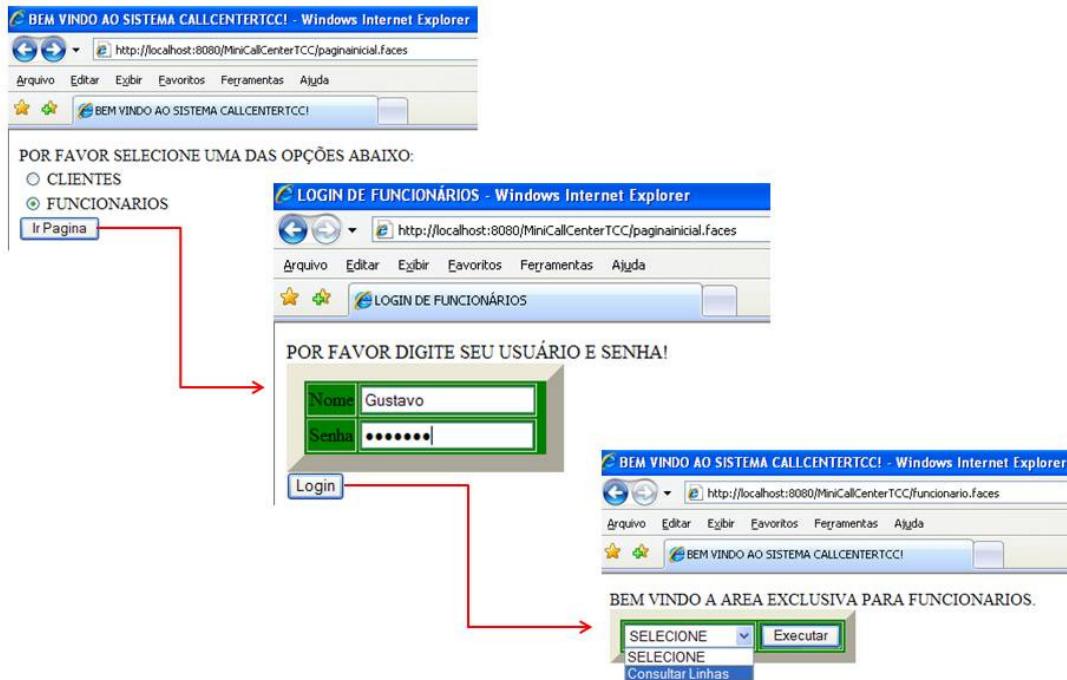


Figura 20 –Desenvolvimento área funcionário.

A Figura 20 ilustra todas as telas antes que um funcionário acesse todas as linhas que deseja consultar em nosso sistema, este capítulo não mostra os detalhes sobre a navegação da área funcionários, nem como os arquivos foram configurados, pois tudo que a Figura 20 ilustra já foi anteriormente mostrado na área do cliente.

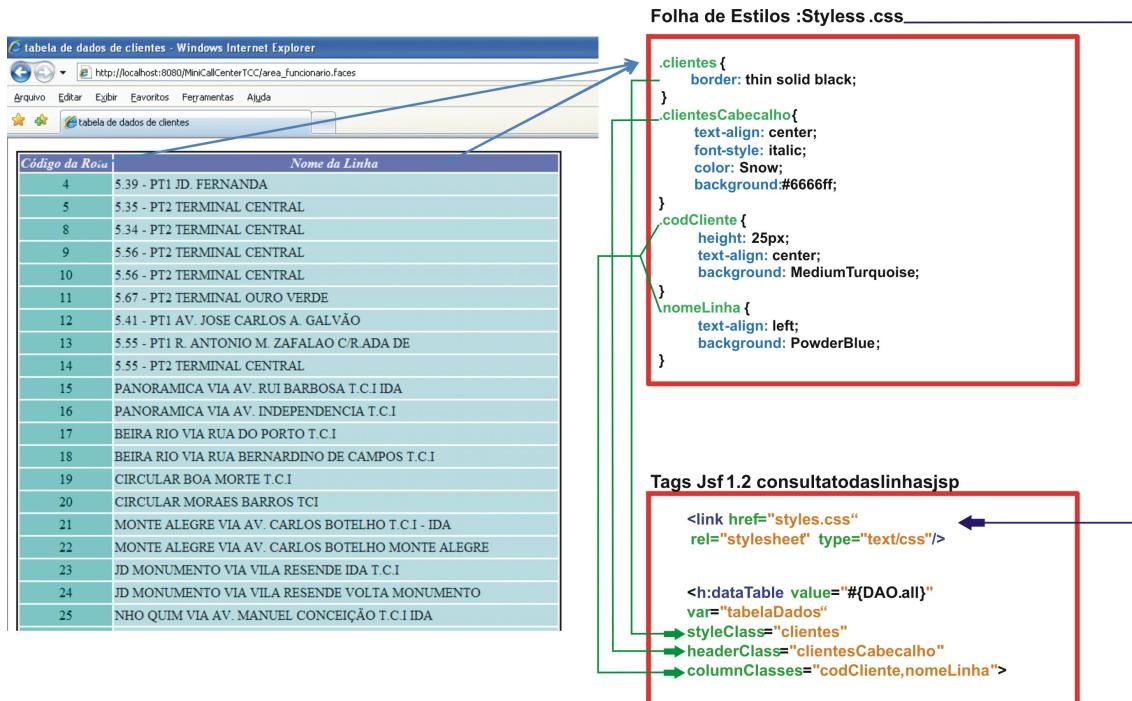


Figura 21 – Página x estilo.

A Figura 21 ilustra mais um detalhe do JSF 1.2, a seta em azul no canto direito ilustra a tag utilizada para chamar nossa folha de estilos configurada, já as setas de cor verde a esquerda dos quadros mostram como invocar os arquivos configurados em nossa folha de estilo para que sejam representados em nossa página jsp, vale ainda dar ênfase a um seguinte detalhe, que esta é apenas uma idéia básica de como se utilizar a folha de estilos neste projeto, lembrando que ainda existem diversas opções de configuração que não será discutido neste projeto.

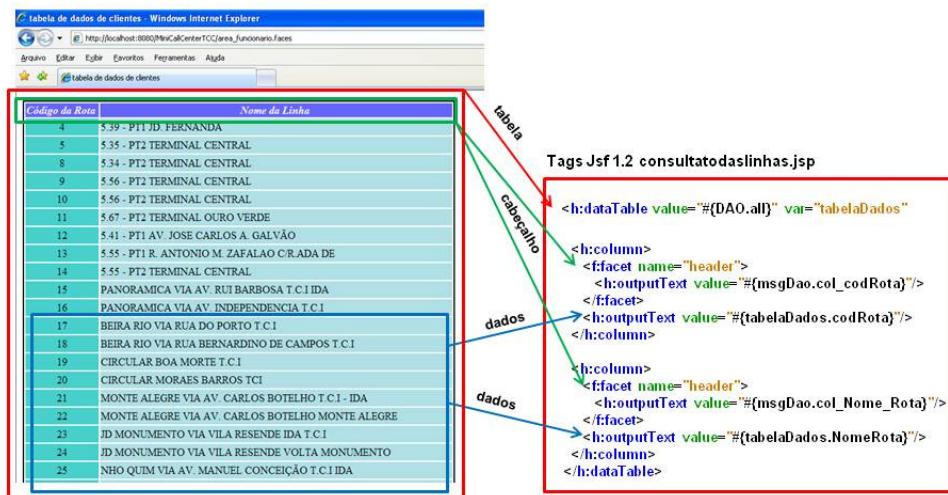


Figura 22 – Página consulta x tags tabela.

A tag `<h:dataTable>` ilustrada na Figura 22 é um poderoso componente, pois pode ser associada tanto a folhas de estilos (Figura 22), bem como a objetos, arrays, coleções, bancos de dados como mostrado neste exemplo, a tag `value="#{DAO.all}"` representa o método declarado em nosso bean, através deste método uma consulta é gerada e assim são retornadas todas as linhas cadastradas no sistema, a tag value dentro de `<h:datatable>` (Figura 22) traz todos os resultados da consulta e a cada iteração ele implementa uma nova linha com seus respectivos valores, a tag `<var="tabelaDados">` é responsável por armazenar todos os dados que a consulta trouxe, e através do nome dela declarada com um ponto é possível trazer todos os campos definidos na consulta, neste projeto como ilustrado pelas tags `value="#{tabelaDados.NomeRota}", value="#{tabelaDados.codRota}"`.

Vale lembrar também que os nomes dos campos após o ponto seletor são respectivamente os nomes das colunas declaradas em nosso consulta, ainda existe mais uma tag ilustrada na Figura 22 com a seguinte sintaxe `<f:facet name="header">` esta tag tem por finalidade colocar na tabela um cabeçalho, através do campo name "header", foi declarado quais colunas fazem parte do cabeçalho, neste projeto não foi utilizado, mas é importante frisar que para ter implementado um rodapé, basta adicionar a seguinte tag `<f:facet name="footer"/>`. Não será mostrado nesta seção, como se dá a navegação nem como o bean foi utilizado, pois o conceito aqui aplicado é o mesmo ilustrado em exemplos anteriores.

6 CONCLUSÕES

Este trabalho conclui que realmente é possível obter mais qualidade em aplicações por meio do uso dos padrões de projetos, e quanto ao foco deste trabalho conclui-se, que o JSF realmente é uma especificação para desenvolvimento *web*, promovida através JCP/SUN, implementa o padrão MVC2, e se parece muito com *frameworks* para desenvolvimentos visuais, sua arquitetura é flexível, escalável, e se integra com outros *frameworks*, o JSF, é um ótimo *framework* para se ganhar em tempo de desenvolvimento desde que a equipe tenha uma bom conhecimento da ferramenta utilizada, seu aprendizado é fácil para desenvolvedores já experientes e ele deve ser usado para aplicações e médio a grande porte, hoje é possível afirmar que com todas as inovações das IDEs, *frameworks*, o JSF pode ser considerado como um ambiente totalmente visual.

O protótipo desenvolvido neste trabalho pode contribuir como um exemplo para novos desenvolvedores, que ainda tenham pouco conhecimento no modelo MVC, bem como no *framework* JSF, para que os mesmos, desde que atendendo as premissas declaradas ao longo deste trabalho, tenham os conceitos básicos sobre este poderoso padrão de projeto MVC e *framework* jsf.

Dado que o *framework* JSF tem muitas funcionalidades não discutidas neste trabalho, fica a sugestão de um aprofundamento deste trabalho para um futuro trabalho.

7 REFERÊNCIAS BIBLIOGRÁFICAS

Jandl Jr., Peter, **Mais Java**: São Paulo: Futura, 2003.

Geary, David et al. **Core Java Server Faces Fundamentos**: 2. Ed., Rio De Janeiro: Alta Books, 2007.

Gamma, Erich. **Padrões de Projeto**: Porto Alegre: Bookman, 2000.

Geary,David et al. **Jsf Tags**: Disponível em:
http://horstmann.com/corejsf/jsf-tags.html#Table4_3. Acesso em: 15 Jun. 2008.

Barros,Tiago et al. **State MVC**: Estendendo o Padrão MVC Para Uso no Desenvolvimento de Aplicações Para Dispositivos Móveis. Disponível em:
http://www.cesar.org.br/files/file/SMVC_01_0104.pdf. Acesso em: 4/06/2008.

Macoratti, José C. **O Modelo MVC**: Disponível em:
http://www.macoratti.net/vbn_mvc.htm - Acesso em: 19 jul. 2008

Pereira, M.N.B., **Artigo Sobre MVC**: Disponível em:
http://anacarol.blog.br/old/aulas/artigos_uteis/modelo_visualizacao_controle.pdf . Acesso em: 19 jul. 2008

Ponte, Rafael. **Anatomia do JSF – Java Server Faces**: Disponível em:
<http://www.rponte.com.br/wp-content/uploads/2007/10/anatomia-jsf-javaserver-faces-cct.pdf> . Acesso em: 23 ago. 2008

Wagner, Gustavo. **Conceitos-Chave de JSF**: Disponível em:
http://www.gustavowagner.com/wiki/lib/exe/fetch.php?id=tecnologias_web_-_iesp...tw:aula7-introducao-jsf.pdf. Acesso em: 25 ago. 2008

Pitanga, Talita. **Java Server Faces**: A Mais Nova Tecnologia Java Para Desenvolvimento Web. Disponível em:
http://209.85.173.132/search?q=cache:vvuX76fOKdwJ:www.guj.com.br/content/articles/jsf/jsf.pdf+jsf.pdf&hl=pt-BR&ct=clnk&cd=1&gl=br&lr=lang_pt. Acesso em: 15 set. 2008

_____.**Java Server Faces** : Disponível em:
http://java.sun.com/javaee/javaserverfaces/1.1_01/docs/api/index.html. Acesso em: 10 nov. 2008.

_____.JSR 252: **Java Server Faces 1.2:** Disponível em:
<http://www.jcp.org/en/jsr/detail?id=252>. Acesso em: 05 out. 2008.

_____.**Java e .Net Disputam Lugar na Web:** Disponível em:
http://www.companyweb.com.br/lista_artigos.cfm?id_artigo=4. Acesso em: 15 nov. 2008.

_____**MYSQL:** Disponível em:
<http://www.mysql.com/>. Acesso em: 22 ago. 2008.

_____**Netbeans:** Disponível em:
<http://www.netbeans.org/downloads/>. Acesso em: 07 mai. 2008.