UNIÃO EDUCACIONAL MINAS GERAIS S/C LTDA
FACULDADE DE CIÊNCIAS APLICADAS DE MINAS
Autorizada pela Portaria no 577/2000 – MEC, de 03/05/2000
BACHARELADO EM SISTEMAS DE INFORMAÇÃO

SISTEMA DE GESTÃO PARA HOTELARIA:

PORTO BELLO PALACE HOTEL

GUSTAVO ESTEVES BORGES
HERLANDRO PINTO DA SILVA HERMOGENES
LUCAS FREDERICO CÂMARA
LUCAS MORENO MAGALHÃES LELES
RUAN RAMON COSTA

Uberlândia
2009

GUSTAVO ESTEVES BORGES
HERLANDRO PINTO DA SILVA HERMOGENES
LUCAS FREDERICO CÂMARA
LUCAS MORENO MAGALHÃES LELES
RUAN RAMON COSTA

SISTEMA DE GESTÃO PARA HOTELARIA:

PORTO BELLO PALACE HOTEL

Trabalho de Conclusão de curso submetido à UNIMINAS como parte dos requisitos para a obtenção do grau de Bacharel em Sistemas de Informação.
.

Orientador: Prof. Esp. Walteno Martins Parreira Júnior

Uberlândia
2009

GUSTAVO ESTEVES BORGES
HERLANDRO PINTO DA SILVA HERMOGENES
LUCAS FREDERICO CÂMARA
LUCAS MORENO MAGLHÃES LELES
RUAN RAMON COSTA

SISTEMA DE GESTÃO PARA HOTELARIA: PORTO BELLO PALACE HOTEL

Trabalho de Conclusão de curso submetido à UNIMINAS como parte dos requisitos para a obtenção do grau de Bacharel em Sistemas de Informação.

Orientador: Prof. Dr. Walteno Martins

Banca Examinadora:

Uberlândia, 21 de dezembro de 2009.

Prof. Esp. Walteno Martins Parreira Jr (Orientador)
Profa. Dra. Kátia Lopes Silva
Prof. Dr. Mauro Hemerly Gazzani

Uberlândia
2009

Dedico este trabalho aos meus familiares, que acreditaram em mim, à minha namorada, que amo muito, pela presença e apoio e a todos os meus amigos que sempre estiveram ao meu lado.

AGRADECIMENTOS

A Prof Walteno Martins braço amigo de todas as etapas deste trabalho.

As famílias, pela confiança e motivação, paciência, compreensão e amor.

Agradeço a Deus por ter guiado nossos passos e nos protegido nos momentos difíceis.

RESUMO

Este trabalho tem como objetivo implementar um aplicativo para gestão de hotelaria utilizando uma arquitetura em três camadas e os padrões de projeto (*design patterns*), facilitando o controle das operações básicas de um hotel (hospedagem e reservas). A utilização de um *design* adequado, com as opções bem distribuídas e práticas para uso diário, através de uma interface simples, possibilitam a automatização de todo o serviço de gerenciamento, atendendo as necessidades do cliente e permitindo que os usuários tenham acesso às informações e sejam mais produtivos no trabalho. A aplicação foi construída com a linguagem de programação Java utilizando as tecnologias *JavaServerPages*, *Servlets*, e *TagLibrary*, tornando o sistema eficiente, eficaz e ao mesmo tempo de fácil utilização. Como a aplicação é baseada na Web, possibilitará uma melhor utilização de recursos físicos (*Hardware*) e de sistema. Foram utilizados os padrões da Sun Front Controller, Application Controller, Context Object que nessa fase final do projeto foram unificados pelo *Struts*, e para a camada de persistência o padrão DAO foi utilizado. Na visualização foram utilizadas *CSS*, *JavaScript*, *Tableless*, *Display Tag Library*, *Struts Tiles* e a chamada “Uma expressão regular”.Para a persistência dos dados foi utilizado o banco de dados *MySQL* a ferramenta de Mapeamento Objeto-Relacional utilizada foi o *Hibernate*.

Palavras Chave: Aplicação Web, Hotelaria on-line, Gerenciamento, Java.

ABSTRACT

This paper aims to implement an application for hotel management, using a three-tier architecture and design patterns (design patterns), facilitating control of the basic operations of a hotel (lodging and reservations). The use of an appropriate design, with options well distributed and practices for everyday wear, through a simple interface, allow for the automation of the management service, meeting the needs of the customer and allowing users to have access to information and be more productive at work. The application was built with the Java programming language using the technologies JavaServerPages, Servlets, and TagLibrary, making the system efficient, effective and yet easy to use. Because the application is Web-based, will enable a better use of physical resources (hardware) and system. We used the standard Sun Front Controller, Application Controller, Context Object that this final phase of the project were unified by Struts, and the persistence layer the DAO pattern was used. On display were used CSS, JavaScript, Tableless, Display Tag Library, Struts and Tiles called "A regular expression." For the persistent data was used MySQL database tool for Object-Relational Mapping Hibernate was used.

Keywords: Web Application, Hospitality Online, Management, Java.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SÍMBOLOS

ABNT – Associação Brasileira de Normas Técnicas

API – Application Programming Interface

ARPA – Agência de Projetos de Pesquisa Avançada

CERN – Organização Europeia para Investigação Nuclear

CHI – Computer-Human Interaction (Interação Computador-Humano)

CPF – Cadastro de Pessoa Física

CS4 – Creative Suite 4

CSS – *Cascading Style Sheets*

CTPS – Carteira de Trabalho e Previdência

DAO – *Data Access Object*

FK – *Foreign key*

GoF – *Gang of Four*

HQL – *Hibernate Query Language*

HTML – *Hyper Text MarkUp Language*

HTTP – *Hipertext Transfer Protocol*

IDE – *Integrated Development Environment*

ISO – *International Standartization Organization*

ITCP – *Internet Transmission Control Program* (Programa de Controle de Transmissão Entreredes)

Java EE – *Java Enterprise Edition*

Java SE – *Java Standart Edition*

JSP – *Java Server Pages*

MVC – *Model-View-Controller*

NCSA – Centro Nacional de Aplicações de Supercomputação

OO – Orientado a Objeto

OOP – *Object Oriented Programmimg*

PK – *Primary Key*

POO – Programação Orientada a Objetos

RAM – *Random Access Memory*

RG – Registro Geral

SGBD – Sistema de Gerenciamento de Banco de Dados

SGPB – Sistema de Gerenciamento Porto Bello

SQL – *Structured Query Language*

W3C – *World Wide Web Consortium*

WaSP – Projeto de Padrões Web ou Web Standards Project

WYSIWYG – *What You See Is What You Get* (O que você vê é o que você tem)

XHTML – *Extensible Hyper Text MarkUp Language*

XML – *Extensible Markup Language* (Linguagem extensível de formatação)

SUMÁRIO

1 INTRODUÇÃO

1.1 Cenário atual

A indústria hoteleira entre diversas constituições empresariais permanece em constante renovação, na hotelaria novos tipos e meios de hospedagem surgem conforme o desenvolvimento sócio-econômico. Desta forma causa-se no mercado uma busca incessante por novos conceitos, que no mínimo acompanhe as inovações do mundo globalizado e que em grande maioria espera-se que supere as expectativas de uma demanda de clientes cada vez mais exigente. O ramo hoteleiro no Brasil possui hoje em torno de 25 mil meios de hospedagem, e desta totalização 20 mil são hotéis e pousadas.

No geral, 70% são empreendimentos de pequeno porte o que representa mais de um milhão de empregos e a oferta de quase um milhão de apartamentos em todo o país.

Sabe-se que de cada 10 empregos da população economicamente ativa brasileira um é de turismo, o que demonstra um número expressivo e razoável em comparação aos outros segmentos de força e representatividade nacional. No setor hoteleiro anualmente há um crescimento que varia de 8 a 10% que dependendo de região para região sofrendo oscilações para mais ou para menos. O turismo de negócios é forte em todo o Brasil e até internacionalmente tendo um crescimento com níveis superiores a 10%, aproximando em até 15% ao ano. No Brasil, país considerado continental, a variação de crescimento torna-se bem mais perceptível sobre o turismo de uma forma geral que possui uma média de 3 a 4 dias.

Com este cenário percebe-se que a hotelaria brasileira é fortemente dependente das contas corporativas, e como tal se encaixam o turismo de negócios, eventos e incentivos. No cenário atual há a necessidade de excelência em sistemas de informação que apóiem a gestão, automatizando rotinas, padronizando atividades e armazenando informações tornando ferramentas importantes no processo de gerenciamento de reservas e hospedagens de uma rede hoteleira.

Existem no mercado vários softwares voltados para o ramo de hotelaria.

Fazer uso de sistemas padronizados ou adquirir *softwares* customizados é uma decisão importante. As duas possibilidades possuem vantagens e desvantagens.

1.2 Identificação do problema

O Porto Bello Palace Hotel é um estabelecimento do ramo de serviços voltado para acomodação de hóspedes e outros serviços. Oferece apartamentos amplos e arejados, equipados com TV a cabo, ar refrigerado, frigobar e *room service* 24h, com quartos nas categorias simples, duplo e triplo. Possui o Centro de Convenções com 10 salas e salões, totalmente preparado para a realização de feiras, congressos, convenções e espaço para montagem de stands, além de salas de apoio e secretaria e toda infra-estrutura para *workshops*, reuniões, cursos, treinamentos e conferências. Está localizado na Avenida João Naves de Ávila, número 3685, Bairro Santa Mônica, Uberlândia-MG. A demanda de hóspedes é constante e mais pessoas estão providenciando suas reservas. Assim, o fluxo de clientes se estabelece em constante movimento, aumentando o fluxo de acomodações e de recursos humanos. Devido a isso, a organização busca sempre oferecer um serviço de qualidade, além de tradição e conforto em suas acomodações para proporcionar maior satisfação e permanência de seus hóspedes.

A implantação de um sistema de informação que gerencie o fluxo de clientes, de acomodações e de funcionários do hotel se fez necessária por proporcionar aos usuários uma ferramenta de maior controle das operações de entrada e saída de hóspedes, da ocupação dos apartamentos, do movimento de funcionários do hotel, entre outros aspectos. Com este melhor gerenciamento, os gestores da organização poderão realizar relatórios, balanços e previsões futuras das temporadas de ocupação do hotel.

1.3 Objetivos do trabalho

1.3.1 Objetivo Geral

Desenvolver uma ferramenta de gerenciamento que proporcione maior controle do fluxo de clientes, cadastro das acomodações e dos funcionários do Porto Bello Palace Hotel.

1.3.2 Objetivos Específicos

Para que o objetivo geral seja alcançado, este trabalho apresenta os seguintes objetivos específicos:

- Levantar os requisitos do sistema.
- Documentar os casos de uso
- Fazer a modelagem do sistema, implementando os Diagramas de Classes, de Seqüências e de Casos de Usos, projetados por meio do *Enterprise Architect*.
- Implementar o aplicativo em três camadas.
- Desenvolver o sistema utilizando a linguagem de programação Java.
- Pesquisar ferramentas e tecnologias (*Servlets, JavaServer Pages, Html Forms* e Banco de dados *MySQL)* para atendimento do projeto.
- Estudar o negócio do cliente, para uma melhor estruturação da aplicação.
- Utilizar Padrões de Projeto para as camadas.
- Utilizar uma base para armazenamento de dados.
- Implementar a camada de apresentação utilizando *Front Controller.*
- Implementar a camada de negócio utilizando *Business Object*.
- Implementar camada de integração utilizando *DAO – Data Access Object.*
- Utilizar *o mySQL* como SGBD.
- Utilizar o *framework Hibernate* para mapeamento objeto-relacional.
- Utilizar *framework Struts*.

1.4 Justificativa para a pesquisa

O cenário globalizado exige precisão na definição dos paradigmas corporativos, cada vez mais as organizações buscam softwares personalizados e que possam atender às suas necessidades de forma mais objetiva. Os sistemas de hotelaria já desenvolvidos encontrados no mercado possuem relativamente um menor custo, porém apresentam funcionalidades que não se encaixam no negocio das organizações.

Cada Hotel tem uma visão, uma missão, e suas regras de negócio, e um software padronizado segue uma diretriz de desenvolvimento focada nas similaridades comuns entre as organizações, não identificando os pontos críticos de cada uma. O cuidado em identificar pontos críticos e o atendimento das preferências do cliente é um legado que o avanço tecnológico e social proporcionou, e que são exigidos pelos clientes.

A implantação de um sistema de gerenciamento de hotelaria que irá proporcionar para o hotel uma redução considerável do tempo, através de operações simplificadas, melhorando sua gerência e oferecendo recursos para ações de planejamento estratégico da organização e atendendo as necessidades especificas da corporação.

1.5 Organização do Trabalho

Para a construção deste *software*, dividiu-se o trabalho em:

- Análise e Projeto: busca identificar o que o sistema irá fazer, quais serão suas funcionalidades e como será feito, de forma a satisfazer o cliente em questão. Esta é a primeira fase de desenvolvimento do *software*.
- Estrutura e Código: é como o código ficará organizado e o que será utilizado nesta organização (estruturação).
- Interface: trata da construção do *layout* e visual da ferramenta, auxiliando de forma clara no desenvolvimento.
- Banco de Dados: é onde irá conter as informações dispostas para serem

organizadas e depois serem armazenadas.

Para uma melhor compreensão do desenvolvimento deste trabalho subdividiu-se em capítulos organizados da seguinte maneira:

- **Capítulo 1**: Introdução – Este capítulo descreve o cenário atual do Porto Bello Palace Hotel, faz-se a identificação do problema, apresenta os objetivos do trabalho, a justificativa para a pesquisa e a organização do trabalho.
- **Capítulo 2:** Análise e Projeto - Descreve as especificidades bem como o detalhamento das regras de negócio, implementação dos diagramas de caso de uso, diagramas de classe de análise e de projeto, diagramas de seqüência dentre outros.
- **Capítulo 3**: Arquitetura – Refere-se à arquitetura do projeto, apresentando todas as tecnologias utilizadas, como os *frameworks Struts* e *Hibernate*. Ao descrever os processos e códigos, utilizou-se como modelo o cadastro de apartamentos.
- **Capítulo 4:** Interface do sistema - Refere-se às apresentações das telas bem como os respectivos códigos, descrevendo todas as metodologias, tecnologias e padrões utilizados, além de detalhar os resultados obtidos pela interface associados a cada uma delas.
- **Capítulo 5**: Persistência de dados - Esse capítulo demonstra o desenvolvimento do modelo de dados e contém o Diagrama de Entidade e Relacionamentos e a explicação de cada tabela do diagrama e os conceitos e tecnologias que foram empregados durante o desenvolvimento do sistema.
- **Capítulo 6**: Conclusões - Apresentação das conclusões observadas pelo grupo no projeto e sugestões para a continuidade de trabalhos futuros.

2 ESPECIFICAÇÃO DO PROBLEMA

2.1 Introdução

A rede hoteleira Porto Bello, com vários empreendimentos no Brasil, buscava melhorar o nível de informação sobre seus hóspedes, bem como garantir a perfeita comunicação entre suas unidades. O objetivo era um só, ganhar eficiência e aprimorar a qualidade do atendimento aos hóspedes. Para isso foi proposto o desenvolvimento de um software de gerenciamento de estadias, que facilitará atividades comumente desempenhadas pela organização:

- Controle de clientes com todas as informações completas e necessárias para sua estadia, registros de funcionários, apartamentos, reservas e hospedagens. Efetuando interações que contemplam cadastros, alterações, remoções e buscas desses dados.
- Ter um panorama completo de seu Hotel com um único *software*.
- O sistema deve gerar relatórios para administração do hotel, além de gerenciar as reservas do estabelecimento.
- A hospedagem pode ser calculada em dias, o controle de reservas deve ter uma interface muito simples e prática, permitindo uma visualização rápida do *status* de ocupação e reserva de cada quarto do estabelecimento, diária ou mensalmente. Telas bem desenhadas com as opções bem distribuídas e práticas para uso diário.
- Consultas simples e eficazes.
- Efetuar reservas com praticidade em uma interface muito intuitiva.
- Na tela de Atendimento de Hotéis deve-se controlar a chegada e saída do Hóspede fazendo o calculo do valor das diárias automaticamente.

Por se tratar de uma rede de hotéis, o sistema deverá ser desenvolvido de maneira que uma futura iteração com outros sistemas seja de fácil implementação.

Para atender as especificações e necessidades do cliente, apresentou-se uma solução pratica e simples. A tecnologia permite integrar e controlar todo o gerenciamento de estadias.

O projeto deverá ser desenvolvido na linguagem JAVA e deve ter suporte a Web, utilizando uma arquitetura (MVC) em três camadas e os padrões de projeto (*design patterns*). Esse desenvolvimento possibilitará aos funcionários centralizar todas as informações em um único servidor, que se destina ao controle de toda rede.

2.2 Levantamento de Requisitos

Foram efetuadas varias reuniões com o cliente para o levantamento das funcionalidades do novo sistema, foi discutido também o grau de importância e qual informação é relevante para a execução das atividades desempenhadas pela organização. Durante a reunião foram definidos quais processos o sistema deverá executar e como ele deverá executar essas funcionalidades que seguem abaixo.

- Cadastro de funcionários – São necessárias informações do funcionário como: nome, RG, CPF, data de nascimento, endereço, número da carteira de trabalho e telefone. Caso esse funcionário venha a utilizar o sistema também se deve definir o nível de acesso a esse funcionário.
- Cadastro de hóspedes – São requisitadas informações como nome, RG, CPF, data de nascimento, endereço, telefone, eao longo do tempo se este hóspede se tornar um cliente freqüente também fica registrado suas preferências quanto a apartamentos.
- Cadastro de apartamentos – Existem três tipos de apartamentos diferentes, o normal (para uma pessoa), o duplo (para duas pessoas) e o triplo (para três pessoas), sendo que para cada tipo de apartamento existe um valor diferente da diária.
- Registro de Reservas – No registro de reservas serão buscadas todas as informações do cliente registradas no seu cadastro além de informações como o apartamento desejado e a data da chegada do hóspede e a quantidade prevista

de diárias, caso seja uma data longínqua se faz necessário uma confirmação antecipada. Não existe limite de data a frente para se fazer uma reserva.

- *Check-in* – Esse processo é semelhante ao processo de reserva com a diferença de que a data de chegada será igual à data atual.
- *Check-out* – É feita a somatória de todos os gastos com o valor da diária multiplicada pelo número de dias de permanência e é realizado o fechamento da conta. As informações dessa hospedagem deverão ser salvos em um histórico do hóspede.
- Impressão de relatório – O relatório devera conter a relação de todos os apartamentos do Hotel, quais estão ocupados, quem esta ocupando esse apartamento e qual a previsão de liberação desse apartamento.
- Consulta – O sistema deve ter a funcionalidade de fazer consultas. Essas consultas devem ser sobre quais hóspedes estão no hotel. Outro tipo de consulta será uma consulta no histórico do sistema, sendo que devera ser pesquisado o cadastro e/ou o histórico de um determinado hóspede.

Para entender as soluções de automação de tais requisitos especificados acima se faz os seguintes questionamentos:

- Quantas vezes ao dia os hóspedes do seu hotel, normalmente impacientes e apressados, se queixam pela demora no fechamento de suas contas e realização do seu *check-out*?
- Quanto dinheiro é perdido pelo fechamento errôneo de contas?
- As reservas em seu hotel podem ser feitas via internet?
- O Turista em seu hotel encontra todas as informações que precisa para uma boa estada?
- O seu hotel aproveita todas as oportunidades de negócio que o turista representa?
- Você consegue acompanhar a taxa de ocupação do seu hotel constantemente?
- A taxa de ocupação está baixa e você não sabe o que fazer para melhorá-la?
- Baixar o preço da diária será que é a resposta? Você possui alguma

estatística sobre a satisfação dos hóspedes sobre os serviços e instalações do hotel?

Diante dessas necessidades foi desenvolvido um sistema que automatiza todos esses processos tornando–os mais seguros e rápidos, facilitando o gerenciamento da informação bem como os processos a ela ligados. Não somente na agilidade de processos, mas também a melhoria do serviço também serão fatores abordados, já que uma base de conhecimento é formada em cada estadia, assim o hotel pode aprender mais sobre aquele hóspede e oferecer um atendimento personalizado.

2.2.1 Especificação dos Requisitos

Na Especificação dos Requisitos de *Software*, é fundamental descrever cuidadosamente, quais os requisitos do sistema e como o sistema deverá implementar tais requisitos, além de descrever objetivamente, como essa implementação poderá ser testada e verificada pelo usuário.

Os requisitos podem ser classificados em:

- **Requisitos Funcionais**: referem-se aos requisitos que estão relacionados com a maneira com que o sistema deve operar, onde se especificam as entradas e saídas do sistema e o relacionamento comportamental entre elas, assim como a interação com o usuário.

- **Requisitos Não Funcionais:** são aqueles que não estão especificamente relacionados com a funcionalidade do sistema. Eles impõem restrições no produto a ser desenvolvido e/ou no processo de desenvolvimento do sistema como também especificam restrições externas as quais o produto precisa atender. Eles referem-se a questões como: segurança, confiabilidade, usabilidade, desempenho, entre outros.

2.3 Documentação de Requisitos

O documento de requisitos é a especificação oficial dos requisitos do sistema

para clientes, usuários finais e desenvolvedores de *software*. Formalmente, podemos definir que o documento de requisitos contém os serviços e funcionalidades que o sistema deve prover, além de restrições e informações sobre o domínio da aplicação, bem como restrições no processo usado para desenvolver o sistema.

Além disso, tal documento pode ser visto como um contrato entre o cliente e o gerente de projeto, pois valida a conformidade segundo a especificação de requisitos do cliente para definição do escopo.

Uma vez que os requisitos foram levantados, cabe agora organizá-los em grupos correlacionados, separando em funcionalidades e agregando a cada uma as suas determinadas restrições.

No projeto desenvolvido, foram identificados os seguintes requisitos funcionais:

- Cadastrar Funcionário;
- Cadastrar Hóspede;
- Cadastrar Apartamento;
- Manter Funcionário;
- Manter Hóspede;
- Manter Apartamento;
- Registrar Reserva;
- Consultar Reserva;
- Registrar entrada por reservas;
- Registrar entrada;
- Registrar de saída;
- Gerar Mapa de Acomodações.

Os quadros a seguir representam a descrição de tudo que o sistema deve fazer referente às funcionalidades citadas anteriormente, além de mostrar as restrições colocadas sobre como o sistema deve realizar a mesma.

Nome: F1 – Cadastrar funcionários			**Oculto ( ) Evidente (** x **)**	
Descrição: O sistema permite o cadastro (inclusão) do funcionário na empresa				
Requisitos Não–Funcionais				
Nome	**Restrição**	**Categoria**	**Obrigatoriedade**	**Permanência**
NF1.1 – Preencher campos;	O sistema deverá permitir a inserção dos campos do funcionário a ser cadastrado como: cargo na empresa, data admissão, CPF, RG, nome, endereço, data de nascimento, n° da CTPS, telefone e opção de operador ou não do sistema.	Especificação	**O**(X) **D**( )	**P**(X) **T**( )
NF1.2 – Validar CPF;	O sistema permitirá o prosseguimento do cadastro mediante uma inserção de CPF válido.	Segurança	**O**(X) **D**( )	**P**(X) **T**( )
NF1.3 – Gerar matricula;	Após a conclusão do cadastro de funcionário o sistema gerará a matricula.	Especificação	**O**(X) **D**( )	**P**(X) **T**( )
NF1.4 – Armazenar os dados.	O sistema permite armazenar os dados mediante a conclusão do cadastro.	Especificação	**O**(X) **D**( )	**P**(X) **T**( )
NF1.5 – Definir nível de acesso.	O sistema permite a inserção do nível de acesso do funcionário caso ele for definido como operador do sistema.	Segurança	**O**(X) **D**( )	**P**(X) **T**( )

Legenda: O – Obrigatório **D** – Desejável **P** – Permanente **T** – Transitório

Quadro 1 – Documento de Requisito Cadastrar Funcionário

No cadastro de um funcionário todos os campos apresentados devem ser preenchidos obrigatoriamente, visto que o RH da empresa necessita das informações para facilitar o gerenciamento de férias, temporização de serviço, a comunicação externa, provendo o acompanhamento pessoal. Deve–se fornecer também um CPF válido, por se tratar de um instrumento auxiliar na autenticação da identidade do indivíduo.

Após a identificação do funcionário faz–se necessário definir o nível de acesso no sistema que o mesmo deve possuir, uma vez que, as atribuições de seu cargo definem as funcionalidades habilitadas ao seu perfil.

Depois de concluído o cadastro do funcionário, o sistema gera automaticamente uma matrícula, com objetivo de identificar cada usuário do sistema por ser única e de fácil memorização. Os dados devem ser salvos para que o histórico seja mantido.

Nome: F2 – Cadastrar Hóspede.			**Oculto ( ) Evidente (** x **)**	
Descrição: O sistema permite o cadastro (inclusão) do hóspede no hotel.				
Requisitos Não–Funcionais				
Nome	**Restrição**	**Categoria**	**Obrigatoriedade**	**Permanência**
NF2.1 – Preencher campos;	O sistema deverá permitir a inserção dos dados pessoais do hóspede a ser cadastrado como**:** nome, RG, CPF, data de nascimento, endereço, telefone ou permitir a entrada dos dados mediante as informações de uma reserva realizada, confirmando o registro de entrada no hotel.	Especificação	**O**(X) **D**( )	**P**(X) **T**( )
NF2.2 – Validar CPF.	O sistema permitira o prosseguimento do cadastro mediante uma inserção de CPF válido.	Segurança	**O**(X) **D**( )	**P**(X) **T**( )
NF2.3 – Registrar preferências.	O sistema permite a inserção de suas preferências quanto a apartamentos se este hóspede se tornar ao longo do tempo um cliente freqüente.	Especificação	**O**( ) **D**(X)	**P**(X) **T**( )
NF2.4 – Armazenar os dados.	O sistema permite armazenar os dados mediante a conclusão do cadastro do hóspede.	Especificação	**O**(X) **D**( )	**P**(X) **T**( )

Legenda: O – Obrigatório **D** – Desejável **P** – Permanente **T** – Transitório

Quadro 2 – Documento de Requisito Cadastrar Hóspede

Para que o hóspede seja cadastrado, seus dados pessoais obrigatoriamente devem ser inseridos e seu CPF validado, tal cadastro faz–se necessário para manter o histórico do hóspede, pois facilita o *check–in* na próxima hospedagem, uma vez que as informações estarão armazenadas. Ao cadastrar o hóspede, há a opção de registrar as preferências em relação a sua estadia, auxiliando na manutenção da qualidade no atendimento.

Nome: F3 – Cadastrar Apartamentos.			**Oculto ( ) Evidente (** x **)**	
Descrição: O sistema deverá permitir o cadastro do apartamento				
Requisitos Não–Funcionais				
Nome	**Restrição**	**Categoria**	**Obrigatoriedade**	**Permanência**
NF3.1 – Preencher campos	O sistema deverá permitir a inserção dos dados dos apartamentos a ser cadastrado como**:** tipo, valor da diária, número, descrição.	Especificação	**O**(X) **D**( )	**P**(X) **T**( )
NF3.2 – Armazenar os dados.	O sistema permite armazenar os dados mediante a conclusão do cadastro do apartamento.	Especificação	**O**(X) **D**( )	**P**(X) **T**( )

Legenda: O – Obrigatório **D** – Desejável **P** – Permanente **T** – Transitório

Quadro 3 – Documento de Requisito Cadastrar Apartamentos

O quadro 3 descreve o requisito funcional cadastrar apartamentos e suas respectivas restrições. Para o cadastro de um apartamento é necessário que todos os campos visualizados sejam obrigatoriamente preenchidos, todas as informações são de extrema importância, pois descrevem detalhadamente os atributos do apartamento facilitando o controle de ocupação, e a localização dos mesmos.

Nome: F4 – Manter funcionário			**Oculto ( ) Evidente (** x **)**	
Descrição: O sistema permite a consulta, alteração e exclusão do funcionário				
Requisitos Não–Funcionais				
Nome	**Restrição**	**Categoria**	**Obrigatoriedade**	**Permanência**
NF4.1 – Atualizar campos;	O sistema deverá permitir a alteração dos campos do funcionário que já está cadastrado como: a função, a data de admissão, CPF, RG, nome, endereço, data de nascimento, n° da CTPS, telefone.	Especificação	**O**(X) **D**( )	**P**(X) **T**( )
NF4.2 – Validar CPF;	O sistema permitirá a consulta do funcionário e alteração dos campos mediante a validação do CPF caso seja alterado.	Segurança	**O**(X) **D**( )	**P**(X) **T**( )
NF4.3 – Armazenar os dados.	O sistema permite armazenar os dados mediante a conclusão da alteração.	Especificação	**O**(X) **D**( )	**P**(X) **T**( )

Legenda: O – Obrigatório **D** – Desejável **P** – Permanente **T** – Transitório

Quadro 4 – Documento de Requisito Alterar Funcionário

O quadro 4 descreve o requisito funcional Manter Funcionário e suas

respectivas restrições. Para alterar o cadastro deve–se informar o CPF para que faça a busca dos dados já cadastrados. O usuário poderá alterar as informações que julgar necessário até mesmo o CPF exceto a matrícula que é gerada automaticamente, sendo válido para que todos os itens modificados sejam atualizados na base de dados. Para a consulta dos dados do funcionário há necessidade de informar o CPF.

Nome: F5 – Manter Hóspede.			**Oculto ( ) Evidente (** x **)**	
Descrição: O sistema permite a consulta, alteração e exclusão do hóspede.				
Requisitos Não–Funcionais				
Nome	**Restrição**	**Categoria**	**Obrigatoriedade**	**Permanência**
NF5.1 – Atualizar campos.	O sistema deverá permitir a consulta e alteração dos dados pessoais do hóspede já cadastrado no sistema**:** nome, RG, CPF, data de nascimento, endereço, telefone.	Especificação	**O**(X) **D**( )	**P**(X) **T**( )
NF5.2 – Validar CPF.	O sistema deverá permitir a contulta e alteração do cadastro mediante a validação do CPF do Hóspede.	Segurança	**O**(X) **D**( )	**P**(X) **T**( )
NF5.3 – Registrar preferências.	O sistema permite a atualização de suas preferências de hospedagem, se este hóspede se tornar ao longo do tempo um cliente freqüente.	Especificação	**O**( ) **D**(X)	**P**(X) **T**( )
NF5.4 – Armazenar os dados.	O sistema permite armazenar os dados mediante a conclusão da alteração do hóspede.	Especificação	**O**(X) **D**( )	**P**(X) **T**( )

Legenda: O – Obrigatório **D** – Desejável **P** – Permanente **T** – Transitório

Quadro 5 – Documento de Requisito Alterar Hóspede

O quadro 5 descreve o requisito funcional Manter Hóspede e suas restrições. Para alterar o cadastro deve–se informar o CPF para que faça a busca dos dados já cadastrados. O usuário poderá alterar as informações que julgar necessário até mesmo o CPF, sendo válido para que todos os itens modificados sejam atualizados na base de dados.

Nome: F6 – Manter Apartamentos.			**Oculto ( ) Evidente (** x **)**	
Descrição: O sistema deverá permitir a consulta, alteração e exclusão dos apartamentos				
Requisitos Não–Funcionais				
Nome	**Restrição**	**Categoria**	**Obrigatoriedade**	**Permanência**
NF6.1 – Atualizar campos	O sistema deverá permitir a consulta e alteração dos dados dos apartamentos cadastrados como**:** tipo, valor da diária, número, descrição.	Especificação	**O**(X) **D**( )	**P**(X) **T**( )
NF6.2 – Armazenar os dados.	O sistema permite armazenar os dados mediante a conclusão da alteração do apartamento.	Especificação	**O**(X) **D**( )	**P**(X) **T**( )

Legenda: O – Obrigatório **D** – Desejável **P** – Permanente **T** – Transitório

Quadro 6 – Documento de Requisito Cadastrar Apartamentos

O quadro 6 descreve o requisito funcional Atualizar Apartamentos e suas respectivas restrições. Para alterar o cadastro deve–se informar o código do apartamento para que faça a busca dos dados já cadastrados. O usuário poderá alterar as informações que julgar necessário até mesmo o próprio código, sendo um novo código, para que todos os itens modificados sejam atualizados na base de dados.

Nome: F7 – Registrar reserva.			**Oculto ( ) Evidente (** x **)**	
Descrição: O sistema deverá registrar de acordo com a política da empresa.				
Requisitos Não–Funcionais				
Nome	**Restrição**	**Categoria**	**Obrigatoriedade**	**Permanência**
NF7.1 – Buscar informações do hóspede.	O sistema permite verificar se o hóspede é cliente do hotel com base nas informações do cadastro de cliente.	Especificação	**O**(X) **D**( )	**P**(X) **T**( )
NF7.2 – Verificar disponibilidade de apartamentos.	O sistema permite verificar com base no cadastro de apartamentos sua disponibilidade.	Especificação	**O**(X) **D**( )	**P**(X) **T**( )
NF7.3 – Preencher campos	O sistema permite o registro dos campos como: tipo de apartamento, número do quarto, data da chegada do hóspede, quantidade dias previstos, e informações do cliente como: nome, RG, CPF, data de nascimento, endereço, telefone, com base no cadastro de clientes.	Especificação	**O**(X) **D**( )	**P**(X) **T**( )
NF7.4 – Armazenar os dados.	O sistema permite armazenar os dados mediante a conclusão do registro da reserva.	Especificação	**O**(X) **D**( )	**P**(X) **T**( )

Legenda: O – Obrigatório **D** – Desejável **P** – Permanente **T** – Transitório

Quadro 7 – Documento de Requisito Registrar Reserva

O quadro 7 descreve o requisito funcional Registrar Reserva e suas respectivas restrições. Para registrar uma reserva é necessário que o hóspede esteja cadastrado e o tipo de apartamento desejado esteja disponível para assim prosseguir a reserva e preencher os dados para concluir e gerar o código da reserva.

Nome: F8 – Consultar Reserva.			**Oculto ( ) Evidente (**X**)**	
Descrição: O sistema permite que seja feita uma consulta de reserva.				
Requisitos Não–Funcionais				
Nome	**Restrição**	**Categoria**	**Obrigatoriedade**	**Permanência**
NF8.1 – Buscar informações da reserva_	O sistema permite consultar uma reserva através do CPF do cliente ou pelo numero da reserva, Deve trazer todas as informações da reserva.	Especificação	**O**(X) **D**( )	**P**(X) **T**( )

Legenda: O – Obrigatório **D** – Desejável **P** – Permanente **T** – Transitório

Quadro 8 – Documento de Requisito Consultar Reserva

O quadro 8 descreve o requisito funcional Registrar Reserva e suas respectivas restrições. Para registrar uma reserva é necessário que o hóspede esteja cadastrado e o tipo de apartamento desejado esteja disponível para assim prosseguir a reserva e preencher os dados para concluir e gerar o código da reserva.

Nome: F9 – Registrar Entrada por reserva			**Oculto ( ) Evidente (** x **)**	
Descrição: O sistema deverá permitir o registro da entrada através de reserva.				
Requisitos Não–Funcionais				
Nome	**Restrição**	**Categoria**	**Obrigatoriedade**	**Permanência**
NF9.1 – Consulta informações de reserva.	O sistema permite consultar a existência da reserva com base no código da reserva ou CPF válido do cliente que solicitou.	Especificação	**O**(X) **D**( )	**P**(X) **T**( )
NF9.2 – Registrar entrada	O sistema permite o registro dos campos como: tipo de apartamento, número do quarto, data da chegada do hóspede, quantidade dias previstos.	Especificação	**O**(X) **D**( )	**P**(X) **T**( )
NF9.3 – Armazenar dados de clientes	O sistema permite armazenar os dados mediante a conclusão do registro de entrada.	Especificação	**O**(X) **D**( )	**P**(X) **T**( )

Legenda: O – Obrigatório **D** – Desejável **P** – Permanente **T** – Transitório

Quadro 9 – Documento de Requisito Registrar Entrada por Reserva

O quadro 9 descreve o requisito funcional Registrar Entrada por Reserva e suas respectivas restrições. Para registrar uma entrada por reserva é necessário

que o cliente tenha registrado e confirmada sua reserva para assim prosseguir a o *check–in* e preencher os dados complementares para efetivar a sua estrada de estadia no hotel.

Nome: F10 – Registrar entrada.			**Oculto ( ) Evidente (** x **)**	
Descrição: O sistema permite o cadastro(inclusão) do hóspede no hotel.				
Requisitos Não–Funcionais				
Nome	**Restrição**	**Categoria**	**Obrigatoriedade**	**Permanência**
NF10.1 – Consulta hóspede	O sistema permite consultar a existência de cadastro de hóspede com base no CPF válido ou nome do cliente.	Especificação	**O**(X) **D**( )	**P**(X) **T**( )
NF10.2 – Verificar disponibilidade de apartamentos.	O sistema permite verificar com base no cadastro de apartamentos sua disponibilidade.	Especificação	**O**(X) **D**( )	**P**(X) **T**( )
NF10.3 – Registrar entrada	O sistema permite o registro dos campos como: tipo de apartamento, número do quarto, data da chegada do hóspede, quantidade dias previstos.	Especificação	**O**(X) **D**( )	**P**(X) **T**( )
NF10.4 – Armazenar dados de clientes	O sistema permite armazenar os dados mediante a conclusão do registro de entrada.	Especificação	**O**(X) **D**( )	**P**(X) **T**( )

Legenda: O – Obrigatório **D** – Desejável **P** – Permanente **T** – Transitório

Quadro 10 – Documento de Requisito Registrar Entrada

O quadro 10 descreve o requisito funcional Registrar Entrada e suas respectivas restrições. Para registrar uma entrada é necessário que o cliente esteja cadastrado para assim prosseguir a o check–in e preencher os dados da estadia para ara efetivar a sua estrada no hotel.

Nome: F11 – Registro de Saída			**Oculto ( ) Evidente (** x **)**	
Descrição: O sistema permite o fechamento da conta do hóspede.				
Requisitos Não–Funcionais				
Nome	**Restrição**	**Categoria**	**Obrigatoriedade**	**Permanência**
NF11.1 – Consultar Extrato de Gastos do Cliente	O sistema deverá permitir acesso à consulta do extrato de gastos de clientes que se encontra no campo histórico.	Especificação	**O**(X) **D**( )	**P**(X) **T**( )
NF11.2 – Gerar total de gastos do cliente	O sistema deverá permitir o valor total de gastos de clientes.	Especificação	**O**(X) **D**( )	**P**(X) **T**( )
NF11.3 – Fechar conta.	O sistema deverá atribuir ao registro de saída sua data e hora do encerramento da conta e valor total pago.	Especificação	**O**(X) **D**( )	**P**(X) **T**( )

Legenda: O – Obrigatório **D** – Desejável **P** – Permanente **T** – Transitório

Quadro 11 – Documento de Requisito Registrar Saída

O quadro 11 descreve o requisito funcional Registrar Saída e suas respectivas restrições. Para que o hóspede feche sua estadia há necessidade que seja lançado todos os valores dos serviços oferecidos pelo hotel para assim efetivar o *check–out* do hóspede.

Nome: F12 – Gerar Mapa de Acomodações			**Oculto ( ) Evidente (**X**)**	
Descrição: O sistema permite a criação de um relatório de acomodações do hotel				
Requisitos Não–Funcionais				
Nome	**Restrição**	**Categoria**	**Obrigatoriedade**	**Permanência**
NF12.1 – Gerar *Home List*	O sistema permite gerar um relatório que devera conter a relação de todos os apartamentos do Hotel, quais estão ocupados, quem esta ocupando esse apartamento e qual a previsão de liberação desse apartamento.	Especificação	**O**(X) **D**( )	**P**(X) **T**( )

Legenda: O – Obrigatório **D** – Desejável **P** – Permanente **T** – Transitório

Quadro 12 – Documento de Requisito Gerar Mapa de Acomodações.

O quadro 12 descreve o requisito funcional Gerar Mapa de Acomodações e suas respectivas restrições. Para que o usuário obtenha as informações deverá possuir pelo menos um apartamento cadastrado, assim trazendo todos os dados de ocupações ou reservas existentes para ele.

3 ANÁLISE E PROJETO

3.1 Introdução

3.1.1 Engenharia de Software

Durante a “crise do *software*”, ficou explícito que as abordagens informais utilizadas na época para desenvolvimento de *software* eram ineficientes, propiciando atrasos para o fim dos projetos e gerando um custo acima do previsto. Com intuito de buscar a produção efetiva de *software* de alta qualidade com custos e prazos reduzidos, foi criada uma abordagem sistemática, a Engenharia de *Software*.

Um sistema de *software* complexo se caracteriza por um conjunto de componentes abstratos de *software* (estruturas de dados e algoritmos) encapsulados na forma de procedimentos, funções, módulos, objetos ou agentes interconectados entre si, que deverão ser executados em sistemas computacionais. Essa engenharia envolve o uso de modelos abstratos e precisos que permitem ao engenheiro especificar, planejar e gerenciar o processo de desenvolvimento.

Segundo Gaudel (1996), a Engenharia de *Software* pode ser vista como a arte de especificar, analisar, conceber, realizar, verificar e fazer evoluir programas, documentações, procedimentos de qualidade com meios e prazos razoáveis, a fim de poder usar um sistema computacional para tratar certos problemas.

No Levantamento de Requisitos é muito importante que os requisitos sejam extraídos da maneira correta e objetiva. Caso o sistema seja utilizado por uma grande quantidade de pessoas, é recomendado que as entrevistas sejam feitas com presença de um conjunto de representantes do cliente (dependendo da complexidade do sistema) divididos por áreas de atuação e outros especialistas das áreas de interesse.

> Para desenvolver um *software* que satisfaça o propósito pretendido, é necessário reunir-se e interagir com os usuários de uma maneira disciplinada, buscando determinar os requisitos reais do sistema

(BACALÁ, 2003, P.12).

Requisito pode ser descrito como uma condição ou capacidade necessitada por um usuário para resolver um problema ou alcançar um objetivo. Enfim, tudo o que o sistema deve fazer para implementar uma necessidade de automação requerida pela solução. Na prática, requisito é o que o sistema tem que ter para atender plenamente ao propósito para o qual foi criado.

Depois de feito o levantamento de requisitos, há necessidade de uma análise envolvendo a equipe funcional que deve verificar cada requisito, em termos de existir consistência, e checar os mesmos como um todo.

- A especificação não deverá haver a preocupação com a solução técnica que será adotada, poderá, no entanto, há a necessidade de descrever as opções já verificadas ou desejadas pelos usuários no que se refere à tecnologia pretendida.

3.1.2 A Linguagem UML

Para possibilitar a visualização dos requisitos e da arquitetura, facilitando a compreensão do problema proposto é necessário utilização de uma ferramenta de modelagem, pois durante a execução das mais diversas atividades devem-se construir modelos para representar a estrutura e o comportamento desejado do sistema. Para o desenvolvimento do projeto utilizou-se a modelagem UML (*Unified Modeling Language*) que tem como objetivo:

- Fornecer aos usuários uma linguagem de modelagem visual expressiva e pronta para o desenvolvimento para os modelos de negócio;
- Ser independente de linguagens de programação e processos de desenvolvimento;
- Prover uma base formal para prover uma linguagem de modelagem;
- Encorajar o Crescimento no número de ferramentas orientadas a objeto no mercado;

- Suportar conceitos de desenvolvimento de nível mais elevado tais como colaborações, estruturas de trabalho, padrões e componentes;
- Integrar as melhores práticas;

A *Unified Modeling Language* é muito mais que a padronização de uma notação, é o desenvolvimento de novos conceitos. Por essa razão entender UML não é apenas aprender a ler uma simbologia, mas significa aprender a modelar orientado a objetos.

> A UML é uma linguagem padrão para especificar, visualizar, documentar e construir artefatos de um sistema e pode ser utilizada com todos os processos ao longo do ciclo de desenvolvimento e através de diferentes tecnologias de implementação (FURLAN, 1998, P.3).

Segundo Bacalá (2003), a UML é uma linguagem de modelagem simples, intuitiva, homogênea, coerente e genérica que pode ser estendida e configurada pelo usuário, pois é suficientemente expressiva para modelar a estrutura, o comportamento e o fluxo de trabalho de vários tipos de atividades.

Um modo para descrever os vários aspectos de modelagem é através de uma notação definida pelos seus vários tipos de digramas. Um diagrama é uma apresentação gráfica de um conjunto de elementos e são usados para visualizar o sistema sob diferentes perspectivas e define um número de diagramas que permite dirigir o foco para aspectos diferentes do sistema de maneira independente. Se bem usados, os diagramas facilitam a compreensão do sistema que está sendo desenvolvido.

A UML pode ser usada para mostrar as fronteiras de um sistema e suas funções principais utilizando atores e casos de uso, representar uma estrutura estática de um sistema utilizando um diagrama de classe, modelar o comportamento de objetos com diagrama de composição de estado, revelar a arquitetura de implementação física com diagramas de componente de implantação, estender sua funcionalidade através de estereótipos.

A *Unified Modeling Language* possui diversos diagramas que podem ser divididos em estáticos (diagrama de classes, diagrama de objetos, diagrama de

componentes e diagramas de implantação), para representação estrutural dos dados e dinâmico (diagrama de casos de uso, diagramas de seqüência, diagramas de gráficos de estado, diagrama de atividades), usado para representação do controle do sistema.

Para este Projeto foram modelados os diagramas de caso de uso, diagramas de classe, diagrama de pacote, diagramas de seqüência além da documentação de casos de uso e de requisitos.

3.2 Diagrama de Casos de Uso

Casos de uso (do inglês *Use Case*) é uma excelente forma de entender o ponto de vista do usuário simplesmente pelo fato de que modela o que ele precisa executar.

Outros aspectos, tais como: como cada caso de uso é efetivamente utilizado ou onde ficarão armazenados os dados que são utilizados para fazer os casos de uso funcionassem, são assuntos para discussão em próximos diagramas.

Segundo Scheneider (1998), um caso de uso é uma descrição de um conjunto de seqüências representando a interação de itens externos ao sistema (atores) com o próprio sistema. Os casos de uso são técnicas de levantamento e captura de requisitos e tem como principal característica a simplicidade da representação facilitando a comunicação do usuário, além de representar uma funcionalidade do sistema, sem a necessidade de especificar como será implementado

A implementação de um tipo de Caso de Uso é apresentada como uma colaboração de objetos e vínculos junto com possíveis seqüências de fluxos, mensagens que produzem o efeito do Caso de Uso, intensificando a exploração de cada cenário de Caso de Uso de um sistema a partir de um caminho normal pelo cenário descrevendo as exceções e condições de erro que podem acontecer explicitamente.

Para identificação dos casos de uso, podem–se observar os seguintes aspectos:

O ator precisa ler criar, destruir, modificar ou armazenar algum tipo de informação no sistema? *O trabalho diário do ator pode ser simplificado ou tornado mais eficiente através de novas funções no sistema?* *O ator tem de ser notificado sobre eventos no sistema ou ainda notificar o sistema entre si?* *Quais são as funções que o ator necessita do sistema?* *O que o ator necessita fazer?* *Quais são os principais problemas com a implementação atual do sistema?* *Quais são as entradas e as saídas, juntamente com sua origem e destino que o sistema requer?*

Quadro 13 – Aspectos de identificação dos Casos de Uso

Um caso de uso não deve se preocupar com o "como" das funções, mesmo que seja apenas uma maneira de conhecer o fluxo de entrada e saída de dados. É importante encapsular ações que em conjunto sejam relevantes e que de alguma forma influenciam algo ou alguém no sistema. Identificar um caso de uso, portanto, é um esforço que envolve:

- Descobrir um ator (alguém que influencia ou é influenciado pelo sistema);
- Verificar para esse ator ações das quais ele participaria;
- Agrupar tais ações de forma que possuam um nome em comum.

3.2.1 Descrição dos Atores

Os Casos de Uso sempre estão associados a um ator que representa qualquer entidade com a qual o sistema interage tais como usuários e outros sistemas de informação.

Os atores podem ser classificados como atores principais, quem utilizará a

principal funcionaidade do sistema, e atores secundários, quem irá manter, administrar e fazer com que o sistema permaneça operando.

Para descrever–se um Caso de Uso corretamente é necessário identificar cada ator pelo papel que representa na interação com o sistema, por exemplo, cliente, gerente, funcionário.

No diagrama de casos de uso do Sistema de gerenciamento de hotelaria do Porto Bello Palace Hotel, foram identificados dois atores: o Cliente e o Funcionário.

- **Cliente:** Este ator é a pessoa que se beneficia com a realização das atividades realizadas no sistema, em todos os casos de uso em que tem participação é considerado ator secundário (check–in por reserva, check–in, check–out, cadastrar hóspede).
- **Funcionário**: Este ator é a pessoa que atua no sistema para realizar todas as funcionalidades previstas no sistema, em todos os casos de uso tem o papel de ator primeiro, uma vez que inicia todos os casos de uso do sistema desenvolvido.

Cada ator comunica–se com o sistema enviando e recebendo mensagense um caso de uso inicia–se a partir do momento que um ator envia uma mensagem, e é representado por uma elipse conectada a símbolos de atores. Um retângulo mostra os limites (fronteiras) do sistema, contendo em seu interior o conjunto de casos de uso e, do lado externo, os atores interagindo com o sistema. Cada linha entre o ator e uma elipse de caso de uso mostra uma interação entre eles.

As interações são muito importantes em um diagrama de casos de uso, pois descrevem como deve ser a comunicação entre atores e casos de usos e entre casos de usos. No Diagrama de Casos de Uso do Porto Bello Palace Hotel utilizou–se as interações de comunicação, extensão(<<extends>>) e inclusão (<<include>>).

A comunicação é utilizada para conectar o ator com o caso de uso por um caminho sólido.

A extensão é um relacionamento de um caso de uso para o outro, especificando como o comportamento definido para o primeiro pode ser inserido no comportamento definido para o segundo. Pode se disser que é utilizada para expressar rotinas de exceção e pra descrever cenários opcionais de um Caso de

Uso que somente ocorrerão em uma situação específica, se uma determinada condição for satisfeita.

Um relacionamento de inclusão entre casos de uso ocorre quando há uma parcela de comportamento similar entre eles sugerindo uma reutilização em vez de uma nova cópia para descrição do comportamento.

Em uma modelagem de casos de uso é importante que não inclua aspectos de implementação, Conforme já citado anteriormente, o "como" deve ser deixado para outros níveis de abstração, ou seja, para os diagramas de classes e de interação.

No entanto, o aspecto mais perigoso nesse momento seria o abuso de *includes* e *extends*. A relação de inclusão deve ser empregada somente quando houver realmente um compartilhamento da função indicada; caso contrário estará abusando do detalhamento.

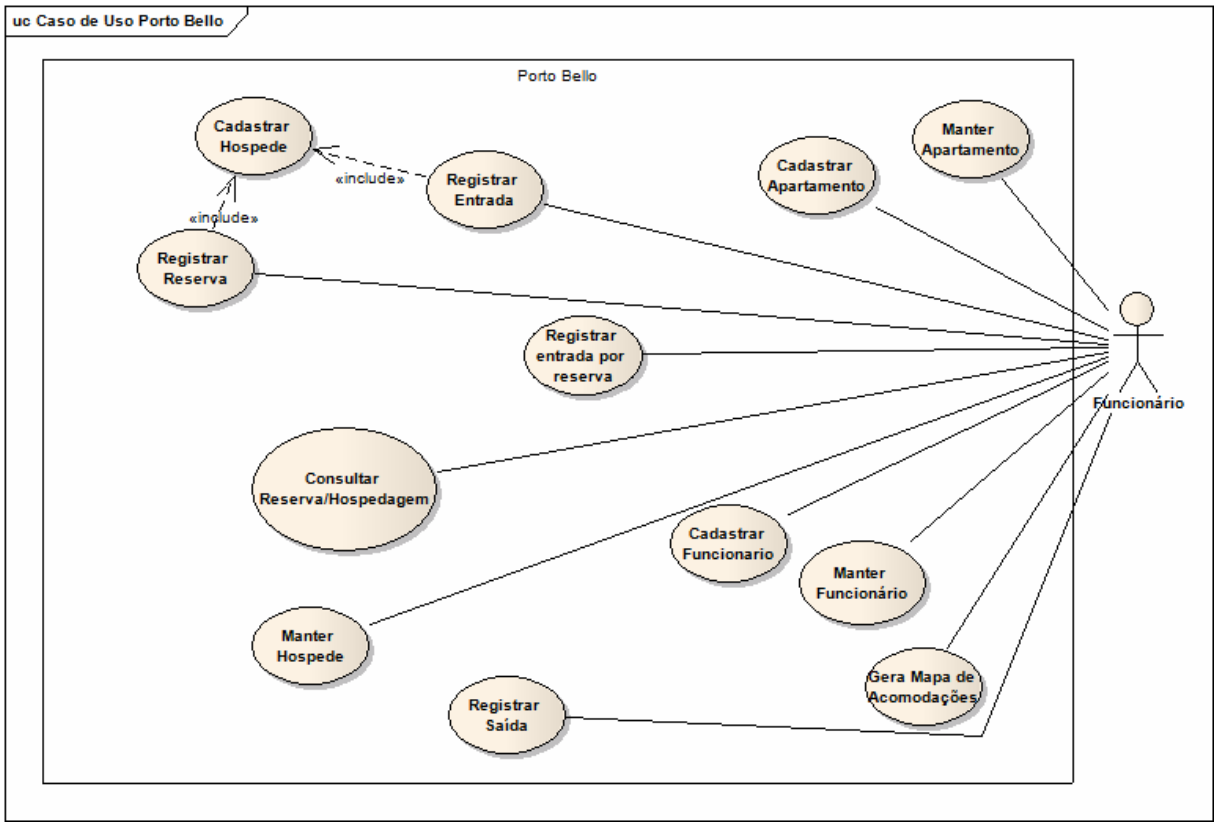

Figura 1 – Diagrama de Casos de Uso

A figura 1 apresenta o diagrama geral de casos de uso do Porto Bello Palace

Hotel.

3.3 Detalhamento dos Casos de Uso

O modelo usado para descrever os casos de uso, os atores e a documentação do caso de uso apresenta os seguintes itens:

- Nome: é o mesmo que aparece no diagrama de caso de uso.
- Sumário: é uma pequena descrição do caso de uso.
- Ator primário: nome do ator que inicia o caso de uso.
- Ator(s) secundário(s): nomes dos demais atores participantes do caso de uso.
- Pré–condição: define que hipóteses são assumidas como verdadeiras para que se tenha início um caso de uso.
- Pós–condição: define a finalização do caso de uso, encerramento deste.
- Fluxo principal: descreve o que normalmente acontece quando o caso de uso é realizado, é uma descrição da seqüência de passos.
- Fluxo alternativo: utilizado para descrever o que acontece quando o ator faz uma escolha alternativa, diferente da descrita no fluxo principal, para alcançar o seu objetivo.
- Fluxo de exceção: descrição de situações que acontecem quando algo inesperado ocorre na iteração entre o usuário e o caso de uso.
- Regras de negócio: a descrição de caso de uso pode fazer uma referencia a uma ou mais regras de negócio, são políticas, condições ou restrições.

Abaixo serão apresentadas as regras de negócios identificadas referente aos casos de uso do sistema SGPB no quadro 14.

Nome	Permissão	**RN-01**
Descrição	Somente poderão operar o sistema usuários que efetuaram o *login*.	
Nome	Cadastro de dados pessoais	**RN-02**
Descrição	Somente serão cadastrados funcionários/hospedes com CPF válido.	
Nome	Preenchimento de dados obrigatórios	**RN-03**
Descrição	Somente salvará os dados após preenchimentos de todos os dados.	
Nome	Cliente cadastrado	**RN-04**
Descrição	Somente com cliente cadastrado que é efetivada a reserva ou entrada no hotel.	
Nome	Apartamento disponível	**RN-05**
Descrição	Somente com apartamento disponível que é efetivada a reserva ou entrada no hotel.	

Quadro 14 – Regra de Negócio

Abaixo serão demonstrados os quadros com as especificações dos casos de uso identificados no sistema.

Nome	Cadastrar funcionário	**CSU–01**
Sumário	Este caso de uso descreve as etapas necessárias para o cadastro de funcionários na empresa.	
Ator primário: Funcionário**.** **Ator secundário:** N/A**.** **Pré–condição:** O Funcionário deve estar logado ao sistema e o funcionário deverá possuir um CPF válido. **Pós–condição:** O funcionário cadastrado.		
Fluxo Principal		
1. O Caso de uso inicia quando o funcionário clica no menu funcionário. 2. O sistema apresenta um submenu referente às opções do funcionário. 3. O Funcionario clica em cadastrar funcionário. 4. O sistema apresenta uma tela com os campos necessários para realizar o cadastro. 5. Funcionário fornece os dados do futuro funcionário, define o cargo e clicar em cadastrar. 6. O sistema deverá armazenar os dados obtidos e gerar um número de matricula. 7. O caso de uso finaliza quando o funcionário recebe na tela a mensagem que os dados foram salvos com sucesso.		
Fluxo de Exceção [3]: Dados Obrigatórios não Preenchidos.		
a. O sistema verifica se todos os campos obrigatórios do cliente foram preenchidos. b. O caso de uso retorna ao passo 2.		
Fluxo de Exceção [3]: CPF inválido.		
a. O sistema emite uma mensagem dizendo que o CPF está incorreto e nã disponibiliza o sucesso do cadastro do funcionário. b. O caso de uso retorna ao passo 3.		
Regra de Negócio Associadas		
RN–01, RN–02 e RN–03.		

Quadro 15 – Detalhe do caso de uso CSU–01

Nome	Cadastrar Hóspede	**CSU–02**
Sumário	Este caso de uso descreve as etapas necessárias para o cadastro de hóspedes no hotel.	
Ator primário: Funcionário. **Ator secundário**: Hóspede. **Pré–condição:** O Funcionário deve estar logado ao sistema e o Hóspede deverá possuir um CPF válido. **Pós–condição:** O funcionário cadastrado.		
Fluxo Principal		
1. O Caso de uso inicia quando o funcionário clica no menu Hóspede. 2. O sistema apresenta um submenu referente às opções do Hóspede. 3. O Funcionário clica em cadastrar hóspede. 4. O sistema apresenta uma tela com os campos necessários para realizar o cadastro e registro de preferências, se for o caso. 5. O Funcionário fornece os dados do cliente ao sistema e, se for o caso, fornece também as preferências do cliente e clica em cadastrar. 6. O sistema armazena os dados obtidos. 7. O caso de uso finaliza quando o funcionário recebe na tela a mensagem que os dados foram salvos com sucesso.		
Fluxo de Exceção [3]: Dados Obrigatórios não Preenchidos.		
a. O sistema verifica se todos os campos obrigatórios do cliente foram preenchidos. b. O caso de uso retorna ao passo 2.		
Fluxo de Exceção [3]: CPF inválido		
a. O sistema emite uma mensagem dizendo que o CPF está incorreto e não disponibiliza o sucesso do cadastro do hóspede. b. O caso de uso retorna ao passo 3.		
Regra de Negócio Associadas		
RN–01, RN–02 e RN–03.		

Quadro 16 – Detalhe do caso de uso CSU–02

Nome	Cadastrar Apartamento	**CSU–03**
Sumário	Este caso de uso descreve as etapas necessárias para o cadastro de apartamentos do hotel.	

Ator primário: Funcionário.

Ator secundário: não possui.

Pré–condição: O apartamento não está cadastrado.

Pós–condição: O apartamento cadastrado.

Fluxo Principal

1. O Caso de uso inicia quando o funcionário clica no menu Apartamento.
2. O sistema apresenta um submenu referente às opções do Apartamento.
3. O Funcionário clica em cadastrar Apartamento.
4. O sistema apresenta uma tela com os campos necessários para realizar o cadastro.
5. Funcionário fornece os dados do apartamento e clicar em cadastrar.
6. O sistema armazena os dados obtidos.

Fluxo de Exceção [3]: Dados Obrigatórios não Preenchidos.

a. O sistema verifica se todos os campos obrigatórios do cliente foram preenchidos.
b. O caso de uso retorna ao passo 2.

Fluxo de Exceção [3]: Apartamento já possui cadastro.

a. O sistema emite uma mensagem dizendo que o apartamento está cadastrado e não disponibiliza o sucesso do cadastro do Apartamento informado.
b. O caso de Uso retorna ao passo 2.

Regra de Negócio Associadas

RN–01 e RN–03.

Quadro 17 – Detalhe do caso de uso CSU–03

Nome	Manter funcionário	**CSU–04**
Sumário	Este caso de uso descreve as etapas necessárias para a Manutenção de um Funcionário.	

Ator primário: Funcionário.

Ator secundário: N/A.

Pré–condição: O Funcionário deve estar logado ao sistema e o funcionário deverá possuir um CPF válido.

Pós–condição: Manutenção do Funcionário efetuada.

Fluxo Principal

1. O Caso de uso inicia quando o funcionário clica no menu Funcionário.
2. O sistema apresenta um submenu referente às opções do Funcionário.
3. O Funcionário clica em Manter funcionário.
4. O sistema apresenta uma tela com o campo de CPF.
5. O Funcionário fornece CPF e clica em pesquisar.
6. O Sistema efetua a consulta e retorna os dados do funcionário.
7. O caso de uso finaliza quando o funcionário recebe na tela a mensagem que a manutenção desejada foi efetuada com sucesso.

Fluxo Alternativo [4]: Alteração

a. O Funcionário modifica os campos que desejar e clicar em atualizar.
b. O sistema atualiza os dados modificados e retorna para o passo cinco.

Fluxo Alternativo [4]: Exclusão

a. O Funcionário clicar em excluir.
b. O sistema exclui o registro apresentado na tela e retorna para o passo cinco.

Fluxo de Exceção [3]: CPF inválido.

a. O sistema emite uma mensagem dizendo que o CPF está incorreto e não disponibiliza a continuação da alteração ou retornos dos dados do funcionário.
b. O caso de uso retorna ao passo dois para consulta e retorna para o passo cinco para atualização dos dados.

Fluxo de Exceção [4]: Funcionário não encontrado.

a. O sistema emite uma mensagem dizendo que o Funcionário não está cadastrado e não disponibiliza o sucesso da busca dos dados do Funcionário.
b. O caso de uso retorna ao passo 2.

Fluxo de Exceção [5]: Dados Obrigatórios não Preenchidos.

a. O sistema emite uma mensagem dizendo que todos os campos obrigatórios para consulta ou de alteração devem ser preenchidos.
b. Na situação de consulta o caso de uso retorna ao passo 2
c. Na situação de alteação o caso de uso retorna ao fluxo alternativo alteração do passo 4.

Regra de Negócio Associadas

RN–01, RN–02 e RN–03.

Quadro 18 – Detalhe do caso de uso CSU–04

Nome	Manter Hóspede	**CSU–05**
Sumário	Este caso de uso descreve as etapas para a Manutenção de um Hóspede.	

Ator primário: Funcionário.

Ator secundário: Hóspede.

Pré–condição: O Funcionário deve estar logado ao sistema e o Hóspede deverá possuir um CPF válido.

Pós–condição: Manutenção do Hóspede efetuada.

Fluxo Principal

1. O Caso de uso inicia quando o funcionário clica no menu hóspede.
2. O sistema apresenta um submenu referente às opções do hóspede.
3. O Funcionário clica no submenu Manter hóspede.
4. O sistema apresenta uma tela com o campo de CPF.
5. O Funcionário fornece CPF e clica em pesquisar.
6. O Sistema efetua a consulta e retorna os dados do hóspede.
7. O caso de uso finaliza quando o funcionário recebe na tela a mensagem que a

Fluxo Alternativo [4]: Alteração

a. O Funcionário modifica os campos que desejar e clicar em atualizar.
b. O sistema atualiza os dados modificados e retorna para o passo cinco.

Fluxo Alternativo [4]: Exclusão

a. O Funcionário clicar em excluir.
b. O sistema exclui o registro apresentado na tela e retorna para o passo cinco.

Fluxo de Exceção [3/5]: CPF inválido

a. O sistema emite uma mensagem dizendo que o CPF está incorreto e não disponibiliza a continuação da alteração ou retornos dos dados do hóspede.
b. O caso de uso retorna ao passo dois para consulta e retorna para o passo cinco para atualização dos dados.

Fluxo de Exceção [4]: Hóspede não encontrado.

a. O sistema emite uma mensagem dizendo que o hóspede não está cadastrado e não disponibiliza o sucesso da busca dos dados do hóspede.
b. O caso de uso retorna ao passo 2.

Fluxo de Exceção [5]: Dados Obrigatórios não Preenchidos.

a. O sistema emite uma mensagem dizendo que todos os campos obrigatórios para consulta ou de alteração devem ser preenchidos.
b. Na situação de consulta o caso de uso retorna ao passo 2.
c. Na situação de alteação o caso de uso retorna ao fluxo alternativo alteração do passo 4.

Regra de Negócio Associadas

RN–01, RN–02 e RN–03.

Quadro 19 – Detalhe do caso de uso CSU–05

Nome	Manter Apartamento	**CSU–06**
Sumário	Este caso de uso descreve as etapas necessárias a Manutenção de apartamentos no hotel.	

Ator primário: Funcionário.

Ator secundário: não possui.

Pré–condição: O apartamento está cadastrado.

Pós–condição: Manutenção do Apartamento efetuada.

Fluxo Principal

1. Caso de uso inicia quando o funcionário clica no menu apartamento.
2. O sistema apresenta um submenu referente às opções do apartamento.
3. O funcionário clica em Manter Apartamento.
4. O sistema apresenta uma tela com o campo numero apartamento.
5. Sistema retorna dados do Apartamento.
6. Funcionário fornece o numero e clica em pesquisar.
7. O Sistema efetua a consulta e retorna os dados do apartamento.
8. O caso de uso finaliza quando o funcionário recebe na tela a mensagem que a manutenção desejada foi efetuada com sucesso.

Fluxo Alternativo [4]: Alteração

a. O Funcionário modifica os campos que desejar e clicar em atualizar.
b. O sistema atualiza os dados modificados e retorna para o passo 6.

Fluxo Alternativo [4]: Exclusão

a. O Funcionário clica em excluir.
b. O sistema exclui o registro apresentado na tela e retorna para o passo 6.

Fluxo de Exceção [4]: Apartamento não possui cadastro.

a. O sistema emite uma mensagem dizendo que o apartamento não está cadastrado e não disponibiliza o sucesso da busca dos dados do Apartamento informado.
b. O caso de uso retorna ao passo 2.

Fluxo de Exceção [5]: Dados Obrigatórios não Preenchidos.

a. O sistema emite uma mensagem dizendo que todos os campos obrigatórios do cliente devem ser preenchidos.
b. Na situação de consulta o caso de uso retorna ao passo 2.
c. Na situação de alteação o caso de uso retorna ao fluxo alternativo alteração do passo 4.

Regra de Negócio Associadas

RN–01 e RN–03.

Quadro 20 – Detalhe do caso de uso CSU–06

Nome	Registrar Reserva	**CSU–07**
Sumário	Este caso de uso descreve as etapas necessárias para o registro de reservas de acordo com a política da empresa.	

Ator primário: Funcionário

Ator secundário: Cliente

Pré–condição: A Reserva não está registrada.

Pós–condição: A Reserva está registrada.

Fluxo Principal

1. Caso de uso inicia quando o funcionário clica no menu Reserva.
2. O sistema apresenta um submenu referente às opções da Reserva.
3. O funcionário clica em Registrar Reserva.
4. O sistema apresenta uma tela com o campo de CPF.
5. Funcionário fornece CPF e clica em pesquisar.
6. O Sistema efetua a consulta e retorna dados da reserva e do hóspede.
7. O Sistema solicita o preenchimento do restante das informações de reserva.
8. Funcionário informa restante dos dados, o tipo de apartamento e clica em registrar.
9. O sistema verifica se há apartamentos disponíveis para reservas do tipo de apartamento informado.
10. O sistema armazena as atualizações efetuadas.
11. Caso de uso finaliza quando o funcionário recebe na tela a mensagem que os dados foram salvos com sucesso.

Fluxo de Exceção [3]: CPF inválido

a. O sistema emite uma mensagem dizendo que o CPF está incorreto e não disponibiliza o sucesso da busca dos dados do hóspede.
b. O caso de uso retorna ao passo dois.

Fluxo Alternativo [4]: Cliente não cadastrado

a. O sistema emite uma mensagem dizendo que o cliente não está cadastrado e não disponibiliza o sucesso do cadastro da reserva.
b. O caso de uso e desviado para o caso uso cadastrar hóspede CSU–02.
c. Após todas as etapas do cadastro do cliente esteja realizada, o caso de uso passa para o passo 5.

Fluxo de Exceção [6]: Dados Obrigatórios não Preenchidos.

a. O sistema verifica se todos os campos obrigatórios do cliente foram preenchidos.
b. O caso de uso retorna ao passo 5.

Fluxo de Exceção [7]: Não há apartamentos disponíveis

a. O sistema emite uma mensagem informando que não há disponibilidade para a reserva para o tipo de apartamento informado. b. Caso de uso retorna ao passo 6.
Regra de Negócio Associadas
RN–01, RN–02, RN–03, RN–04 e RN–05.

Quadro 21 – Detalhe do caso de uso CSU–07

Nome	Consultar Reserva	**CSU–08**
Sumário	Este caso de uso descreve as etapas necessárias para a consulta de reservas de acordo com a política da empresa.	
Ator primário: Funcionário **Ator secundário**: Cliente **Pré–condição:** Reserva já registrada **Pós–condição:** Dados da Reserva consultada.		
Fluxo Principal		
1. Caso de uso inicia quando o funcionário clica no menu reserva. 2. O sistema apresenta um submenu referente às opções da reserva. 3. O funcionário clica em Consultar reserva. 4. O sistema apresenta uma tela com o campo de CPF e numero de reserva. 5. Cliente informa numero de CPF ou numero de reserva.		
Fluxo de Exceção [3]: CPF inválido.		
a. O sistema emite uma mensagem dizendo que o CPF está incorreto e não disponibiliza o sucesso da busca dos dados do hóspede. b. O caso de uso retorna ao passo 2.		
Fluxo Exceção [4]: Reserva não cadastrada		
a. O sistema emite uma mensagem dizendo que o cliente não possui reserva. b. O caso de uso é encerrado.		
Regra de Negócio Associadas		
RN–01 e RN–02.		

Quadro 22 – Detalhe do caso de uso CSU–08

Nome	Registrar Entrada por Reserva	**CSU–09**
Sumário	Este caso de uso descreve as etapas necessárias para o registro de entrada por meio de reserva	
Ator primário: Funcionário **Ator secundário**: Cliente **Pré–condição:** O cliente ter efetuado uma reserva anteriormente. **Pós–condição:** A entrada do hóspede registrada.		
Fluxo Principal		
1. Caso de uso inicia quando o funcionário clica no menu hospedagem. 2. O sistema apresenta um submenu referente às opções da hospedagem. 3. O funcionário clica em Registrar Entrada por Reserva. 4. O sistema apresenta uma tela para ser colocado o CPF ou o numero da reserva. 5. O Funcionário fornece o CPF ou o nome do cliente 6. O sistema consulta, com base no CPF ou numero da reserva, a existência da reserva. 7. Sistema retorna dados da Reserva e solicita restante dos dados. 8. O funcionário informa restante dos dados e clica em registrar. 9. O sistema com base no registro de identificação de reserva armazena os dados do registro.		
Fluxo de Exceção [3]: CPF inválido		
a. O sistema emite uma mensagem dizendo que o CPF está incorreto e não disponibiliza o sucesso da entrada do hóspede. b. O caso de uso retorna ao passo 2.		
Fluxo de Exceção [4]: Reserva Não Cadastrada		
a. O sistema emite uma mensagem dizendo que o cliente não possui reserva e não dispibiliza o *check–in* do Hóspede. b. O caso de uso é encerrado.		
Fluxo de Exceção [6]: Dados Obrigatórios não Preenchidos.		
a. O sistema emite uma mensagem dizendo que todos os campos obrigatórios do cliente devem ser preenchidos. b. O caso de uso retorna ao passo 6.		
Regra de Negócio Associadas		
RN–01, RN–02, RN–03 e RN–05.		

Quadro 23 – Detalhe do caso de uso CSU–09

Nome	Registrar Entrada	**CSU–10**
Sumário	Este caso de uso descreve as etapas necessárias para o registro de entrada.	

Ator primário: Funcionário

Ator secundário: Cliente

Pré–condição: O Funcionário deve efetuar *login* no sistema e o cliente cadastrado.

Pós–condição: A entrada do hóspede registrada.

Fluxo Principal

1. Caso de uso inicia quando o funcionário clica no menu Hospedagem.
2. O sistema apresenta um submenu referente às opções da Hospedagem.
3. O funcionário clica em Registrar Entrada.
4. O sistema apresenta uma tela para ser colocado o CPF.
5. Funcionário fornece o CPF e clica em pesquisar.
6. O sistema consulta, com base no CPF ou nome fornecido, a existência do cadastro desse cliente.
7. O Sistema solicita o preenchimento do restante das informações do *check–in.*
8. Funcionário informa restante dos dados e o tipo de apartamento e clica em registrar.
9. O sistema verifica se há apartamentos disponíveis para reservas para o tipo de apartamento informado
10. O sistema registra a entrada do cliente com base no registro de identificação de reserva ou dados de cadastro de cliente
11. O sistema armazena os dados informados.

Fluxo de Exceção [3]: CPF inválido.

a. O sistema emite uma mensagem dizendo que o CPF está incorreto e não disponibiliza o sucesso da entrada do hóspede.
b. O caso de uso retorna ao passo três.

Fluxo Alternativo [4]: Cliente não cadastrado

a. O sistema emite uma mensagem dizendo que o cliente não está cadastrado e disponibiliza a opção de cadastro do Cliente.
b. Caso de uso e desviado para o caso uso cadastrar hóspede CSU–02.
c. Após todas as etapas do cadastro do cliente esteja realizada, o caso de uso passa para o passo 6.

Fluxo de Exceção [6]: Dados Obrigatórios não Preenchidos.

a. O sistema emite uma mensagem dizendo que todos os campos obrigatórios do cliente devem ser preenchidos.
b. O caso de uso retorna ao passo 6.

Fluxo de Exceção [7]: Não há apartamentos disponíveis
a. O sistema emite uma mensagem informando que não há disponibilidade para o *chek–in* do tipo de apartamento informado. b. Caso de uso retorna ao passo 6.
Regra de Negócio Associadas
RN–01, RN–02, RN–03, RN–04 e RN–05.

Quadro 24 – Detalhe do caso de uso CSU–10

Nome	Registrar Saída	**CSU–11**
Sumário	Este caso de uso descreve as etapas necessárias para o registro de saída do hóspede	

Ator primário: Funcionário

Ator secundário: Cliente

Pré–condição: O Funcionário deve estar logado no sistema e o cliente hospedado.

Pós–condição: A saída do hóspede efetuada

Fluxo Principal

1. Caso de uso inicia quando o funcionário clica no menu Hospedagem.
2. O sistema apresenta um submenu referente às opçoes da Hospedagem.
3. O funcionário clica em Registrar Sáida.
4. O sistema apresenta uma tela com o campo de CPF, numero do apartamento e da hospedagem.
5. O Cliente informa o CPF e número do apartamento ou o número da hospedagem e clica em pesquisar.
6. O sistema efetua a consulta e retorna os dados da hospedagem.
7. Funcionário lança os valores adicionais e clica em encerrar hospedagem.
8. O sistema calcula o valor da hospedagem e atualiza disponibilidade do apartamento.
9. O sistema emite o valor total da hospedagem

Fluxo de Exceção [3]: CPF inválido.

a. O sistema emite uma mensagem dizendo que o CPF está incorreto e não disponibiliza o sucesso da entrada do hóspede.

b. O caso de uso retorna ao passo 3.

Fluxo Alternativo [4]: Hospedagem não cadastrada

a. O sistema emite uma mensagem dizendo que não há hospedagem cadastrada e disponibiliza a opção da consulta da hospedagem.

b. Caso de uso retorna ao passo 2.

Regra de Negócio Associadas

RN–01 e RN–02.

Quadro 25 – Detalhe do caso de uso CSU–11

Nome	Gerar Mapa de Acomodações	**CSU–12**
Sumário	Este caso de uso descreve as etapas necessárias para a geração do mapa de acomodações.	
Ator primário: Funcionário **Ator secundário(s)**: N/A **Pré–condição:** O funcionário estar logado no sistema e apartamentos cadastrados. **Pós–condição:** Relatório gerado		
Fluxo Principal		
1. Caso de uso inicia quando o funcionário clica em Mapa de Acomodações. 2. O sistema busca todas as informações referentes às acomodações. 3. O Funcionário visualiza o relatório gerado e se desejar pode fazer pedir a impressão do relatório. 4. O caso de uso finaliza quando o funcionário clica em sair.		
Fluxo Alternativo [2]: Nenhum apartamento cadastrado		
a. O sistema emite uma mensagem dizendo que não há apartamentos cadastrados e não disponibiliza a geração das acomodações. b. Caso de uso e encerrado.		

Quadro 26 – Detalhe do caso de uso CSU–12

3.4 Diagrama de Classe de Analise

Segundo Furlan (1998), diagrama de classes trata–se de uma estrutura lógica, estática em uma superfície de duas dimensões mostrando uma coleção de elementos declarativos de modelo como classes, tipos e seus respectivos conteúdos e relações, e são encontrados com maior freqüência na modelagem de sistemas orientados objetos

As classes possuem além de um nome, os atributos e métodos. Uma relação indica um tipo de dependência entre as classes. Geralmente as classes não estão sós e se relacionam entre si. O relacionamento e a comunicação entre as classes definem responsabilidades, a UML reconhece três tipos mais importantes de relações: dependência, associação e generalização (ou herança).

A associação é um relacionamento estrutural entre instâncias e especificam

que objetos de uma classe estão ligados a objetos de outras classes. Podemos ter associação unária, quando a um relacionamento de uma classe consigo própria, binária, quando a duas classes envolvidas na associação de forma direta de uma para outra e n–ária, que é uma associação de três ou mais classes, mas uma única classe pode aparecer mais de uma vez. Em uma associação podemos definir dois tipos:

- Agregação Regular – tipo de associação (é parte de, todo/parte) onde o objeto parte é um atributo do todo; onde os objetos partes somente são criados se o todo ao qual estão agregados seja criado. Pedidos é composto por itens de pedidos.

- Agregação de Composição – Relacionamento entre um elemento (o todo) e outros elementos (as partes) onde as parte só podem pertencer ao todo e são criadas e destruídas com ele.

A dependência é um relacionamento de utilização no qual uma mudança na especificação de um elemento pode alterar a especificação do elemento dependente. A dependência entre classes indica que os objetos de uma classe usam serviços dos objetos de outra classe.

A generalização é um relacionamento entre um elemento mais geral e um mais específico. Onde o elemento mais específico (subclasse) herda as propriedades e métodos do elemento mais geral (superclasse). A relação de generalização também é conhecida como herança no modelo a objetos. Como a relação de dependência, ela existe só entre as classes.

A figura 2 apresenta graficamente os relacionamentos entre classes em uma representação de um diagrama de classes.

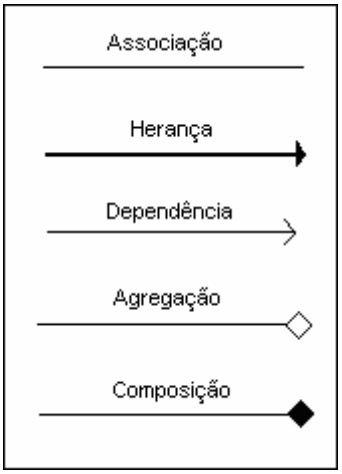

Figura 2 – Representação gráfica de relações entre classes

Uma classe é uma representação de um conjunto de coisas reais ou abstratas que são reconhecidas como sendo do mesmo tipo por compartilhar as mesmas características. A definição de classes inclui atributos e métodos para instâncias podendo ser pensada como uma fábrica que cria instâncias conforme necessário.

Um atributo representa uma propriedade que todos os objetos da classe têm, é a menor unidade que em si possui significância própria e apresenta um princípio do armazenamento de valores em células. Um método é um corpo de procedimento, ou seja, é a implementação de uma operação.

Para poder representar a visibilidade dos atributos e operações em uma classe utiliza–se as seguintes marcas e significados:

- + público – visível em qualquer classe
- # protegido – qualquer descendente pode usar
- – privado – visível somente dentro da classe

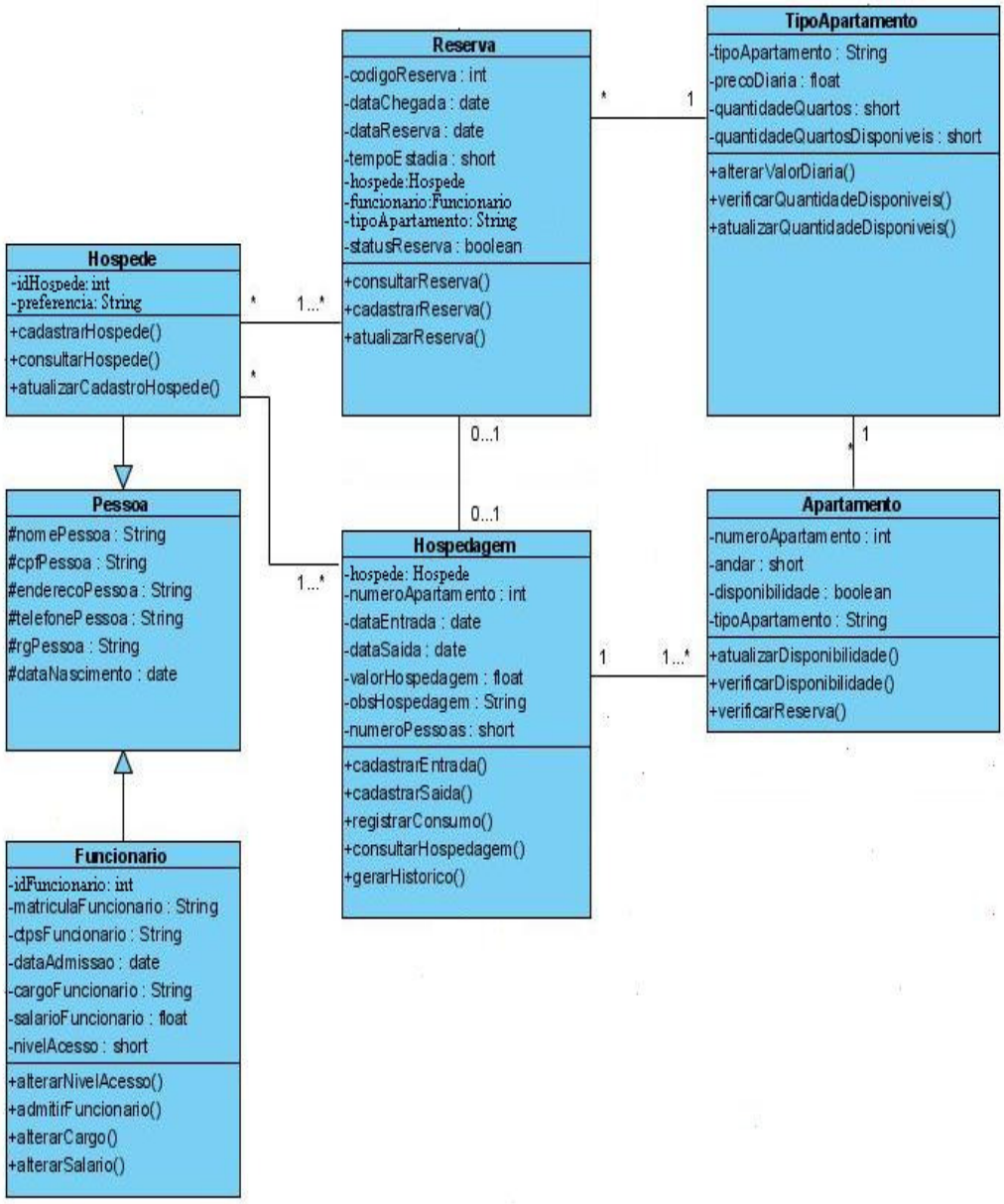

Figura 3 – Diagrama de classe de Análise

A figura 3 apresenta o diagrama de classes do Porto Bello Palace Hotel. Foram identificadas oito classes. Hotel, Pessoa, Hóspede, Funcionário, Reserva, Hospedagem, Apartamento e Tipo Apartamento.

3.5 DIAGRAMA DE SEQÜÊNCIA

Um diagrama de seqüência é um tipo de diagrama de interação que descreve como os objetos colaboram em algum comportamento ao longo do tempo e registra o comportamento de um único caso de uso. Esse diagrama é simples e lógico, com o objetivo óbvio a seqüência e o fluxo de controle.

> Os diagramas de seqüência dão ênfase à ordenação temporal das mensagens mostrando uma visão cronologia de interação entre os objetos que podem representar um cenário de execução ou vários possíveis com bifurcações ou laços. (BACALÁ, 2003, p. 26)

Os casos de uso vistos anteriormente representam conjunto de cenários que descrevem os diferentes processos que ocorrem no sistema. O diagrama de seqüência permite modelar estes processos através da troca de mensagens (eventos) entre os objetos do sistema.

Os objetos são representados por linhas verticais chamadas linhas de vida, e as mensagens como setas que partem do objeto que invoca outro objeto. As setas podem ser cheias para indicar uma mensagem de chamada ou tracejadas para indicar uma mensagem de retorno.

Cada mensagem no diagrama de seqüência corresponde a uma operação no diagrama de classes, como as mensagens são operações invocadas, elas devem estar presentes no objeto de destino, que são ativadas pelas mensagens no objeto de origem.

Abaixo seguem os diagramas de seqüência do Porto Bello, onde se pode observar que o tempo segue o eixo vertical de cima para baixo.

O Diagrama de Sequência Cadastrar Funcionário descreve os passos que devem ser seguidos para efetuar o cadastro de funcionários do Hotel. Através desse diagrama o funcionário cria-se também o login caso o funcionário utilize o sistema.

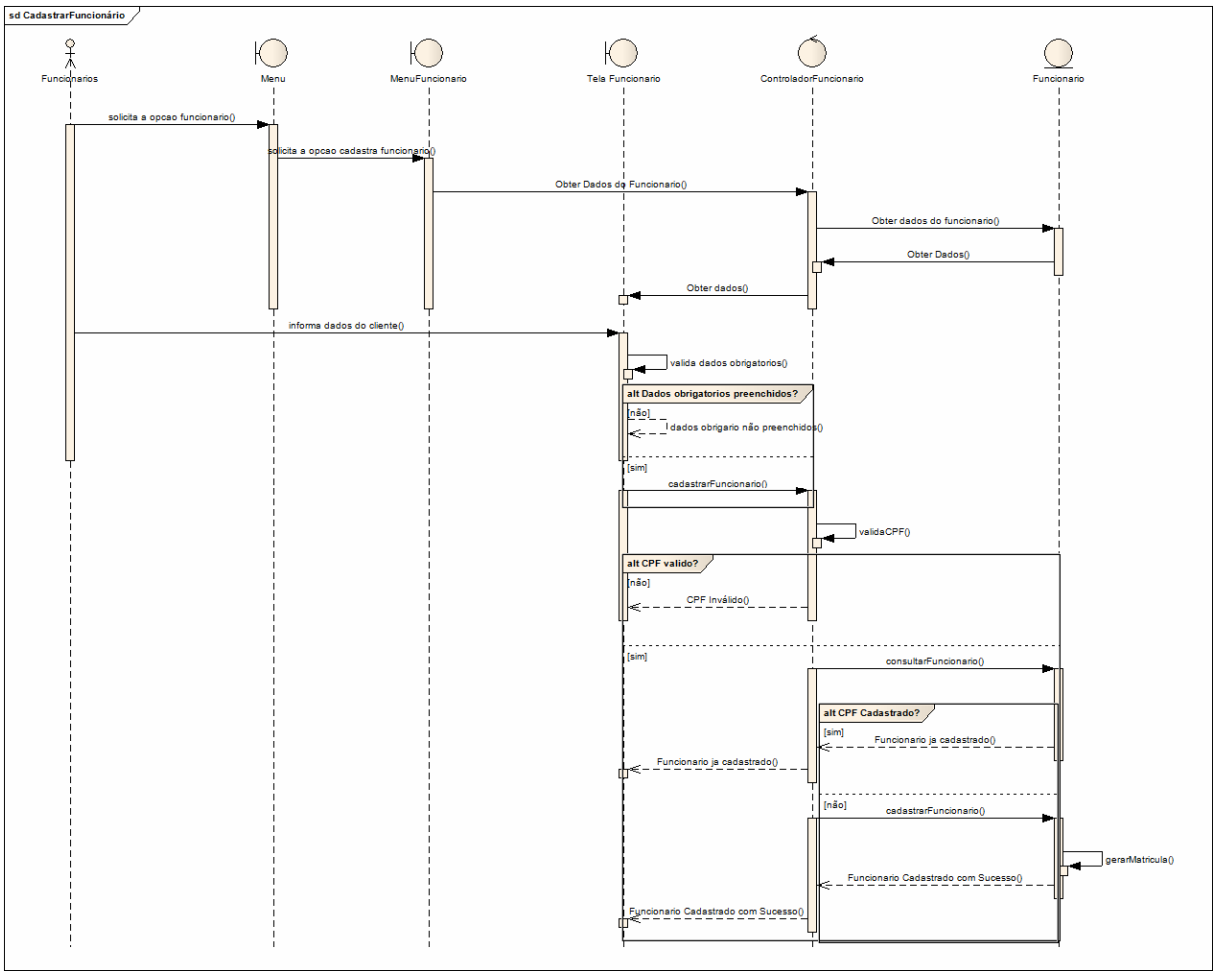

Figura 4 – Diagrama de seqüência Cadastrar Funcionário

O Diagrama de Sequência Cadastrar Hóspede descreve os passos que devem ser seguidos para efetuar o cadastro de Hóspedes do Hotel. Através desse diagrama gravam-se as preferências do hóspde se houver.

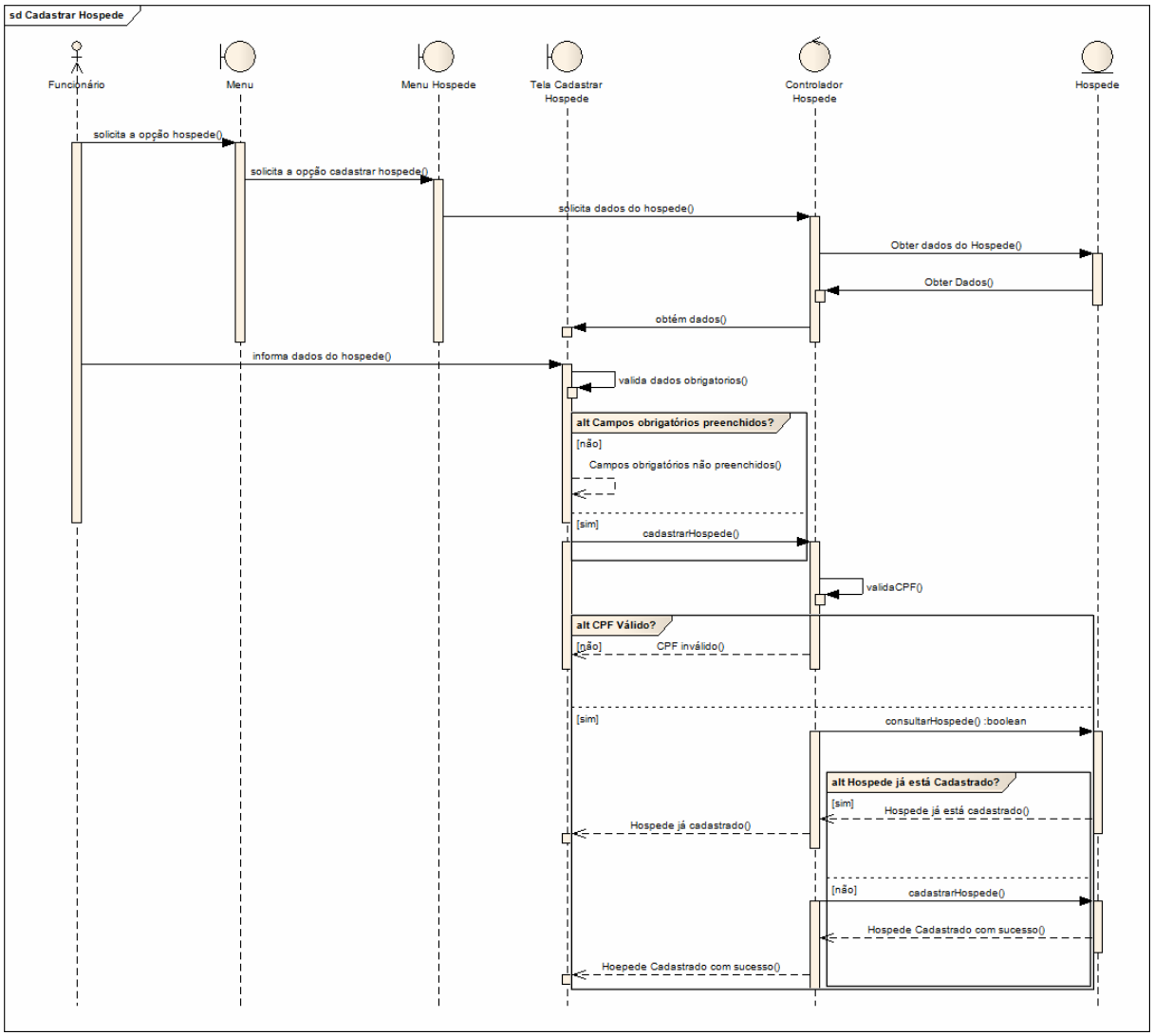

Figura 5 – Diagrama de seqüência Cadastrar Hóspede

O Diagrama de Sequência Cadastrar Apartamento descreve os passos que devem ser seguidos para efetuar o cadastro dos apartamentos do Hotel. Através desse diagrama é que classifica os quartos e suas características.

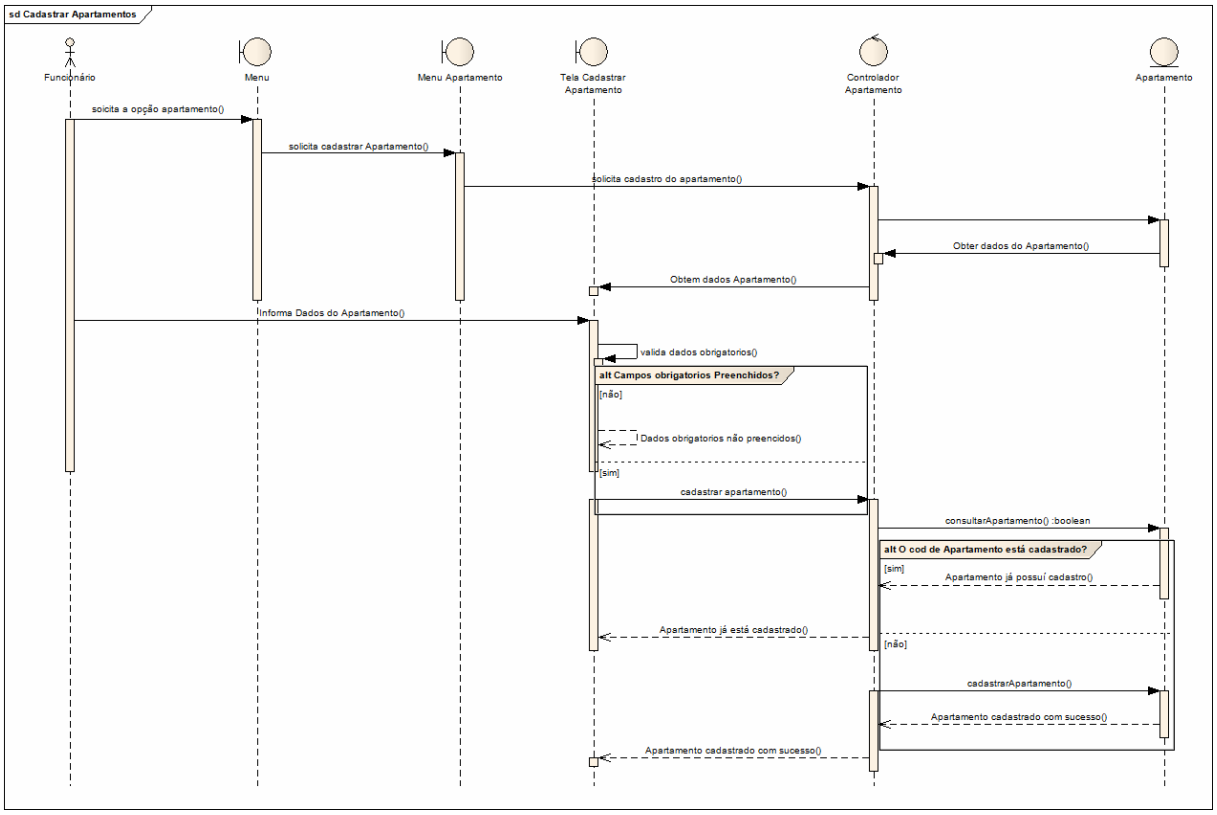

Figura 6 – Diagrama de seqüência Cadastrar Apartamento

O Diagrama de Sequência Manter Funcionário descreve os passos que devem ser seguidos para efetuar a Manutenção de dados do funcionário do Hotel.

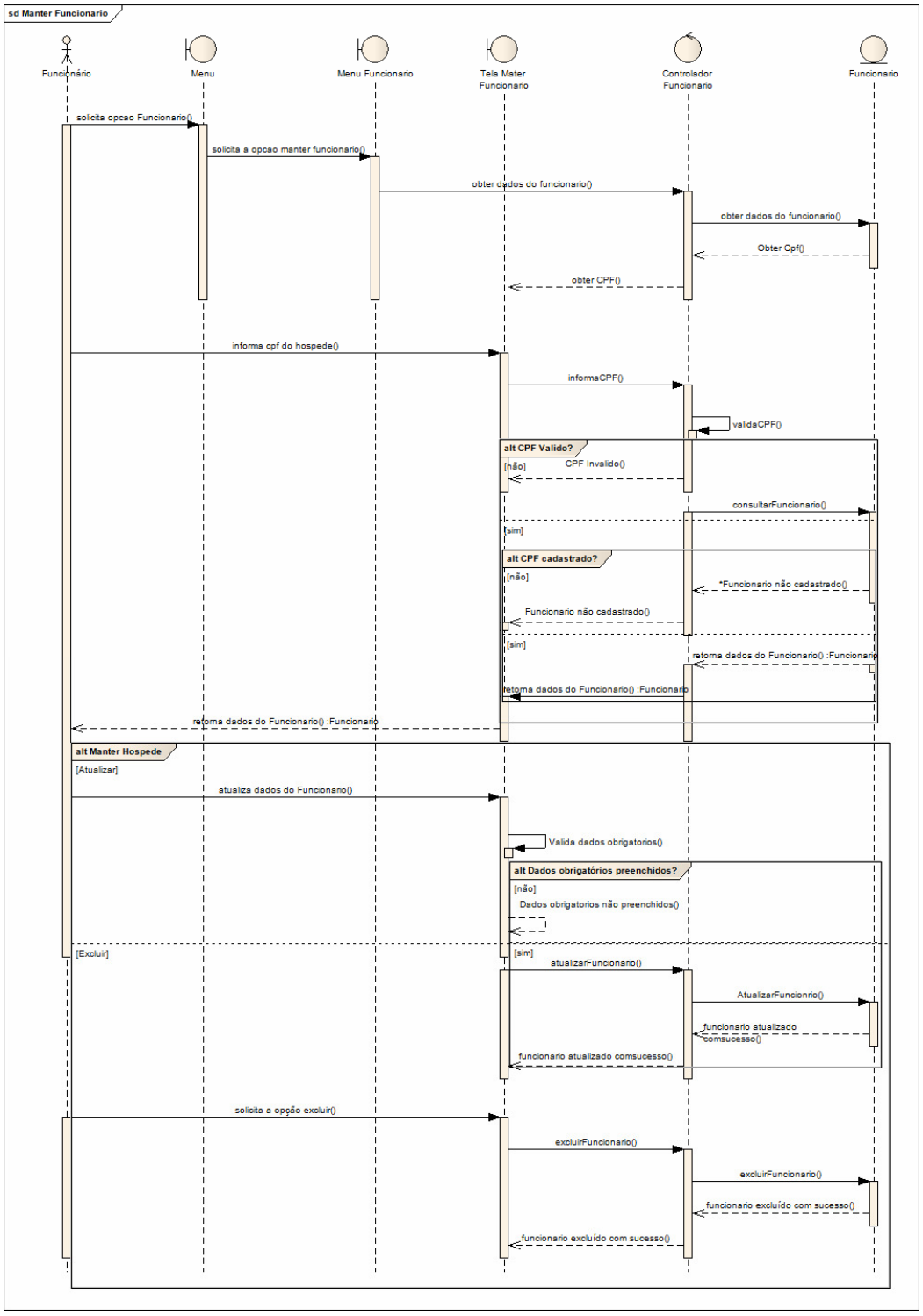

Figura 7 – Diagrama de seqüência Manter Funcionário

O Diagrama de Sequência Manter Hóspede descreve os passos que devem ser seguidos para efetuar o cadastro de hóspedes do Hotel. Através desse diagrama

permiti-se consultar hóspedes já cadastrados e alterar dados necessários.

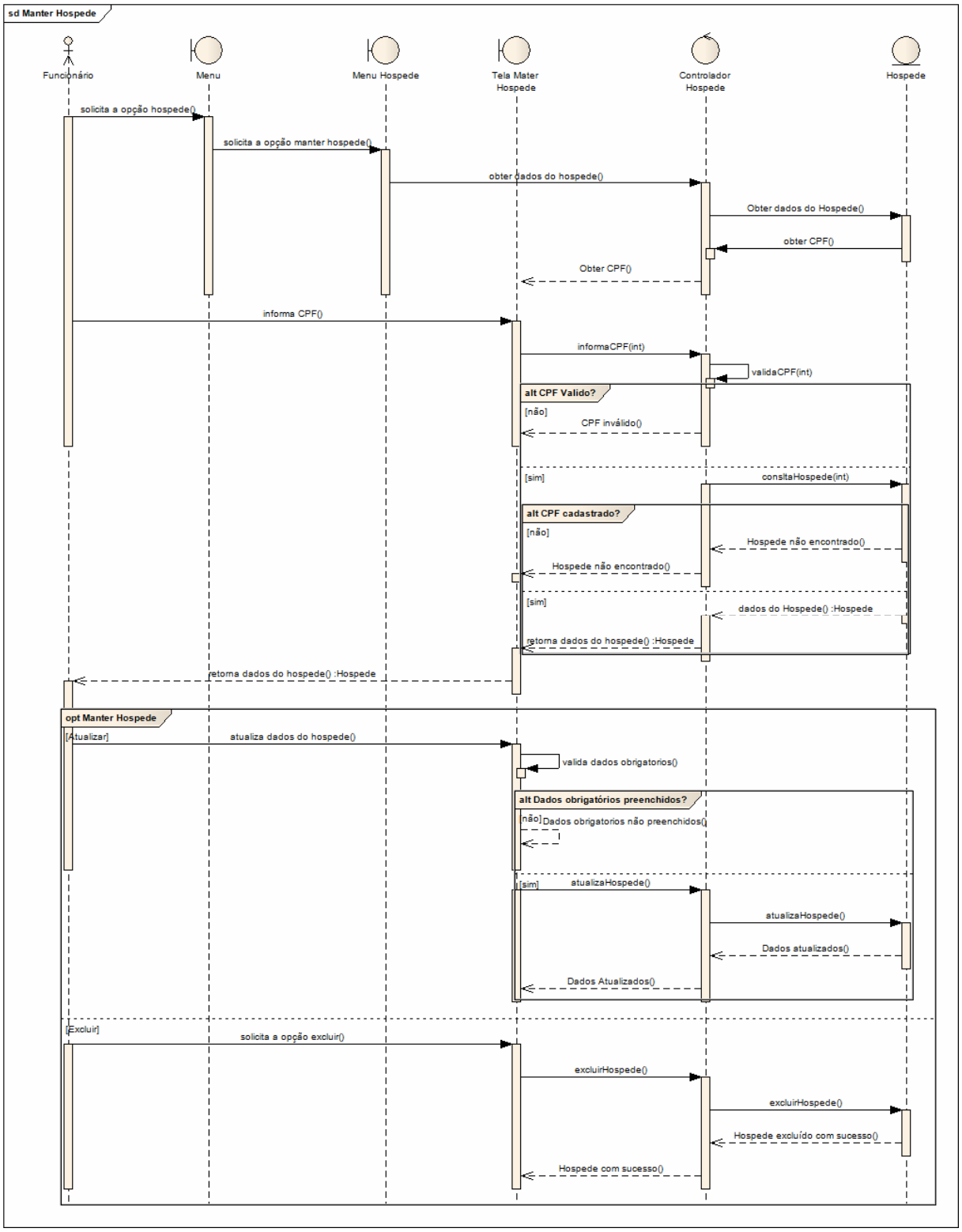

Figura 8 – Diagrama de seqüência Manter Hóspede

O Diagrama de Sequência Manter Apartamento descreve os passos que

devem ser seguidos para efetuar o cadastro dos apartamentos do Hotel. Através desse diagrama permiti-se consultar hóspedes já cadastrados e alterar dados necessários.

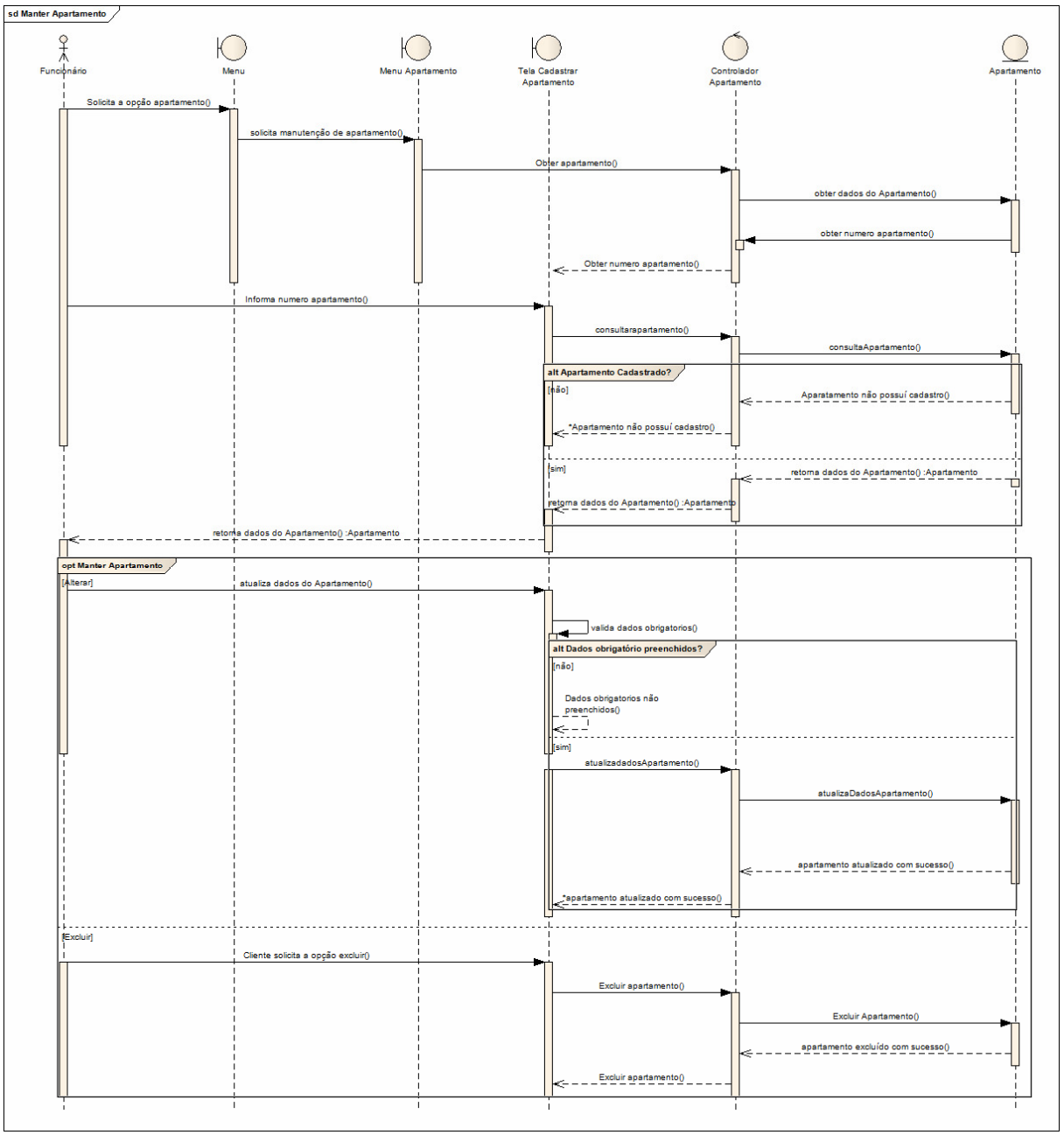

Figura 9 – Diagrama de seqüência Manter Apartamento

O Diagrama de Sequência Registrar Reserva descreve os passos que devem ser seguidos para efetuar o Registro de reservas de apartamentos disponíveis do hotel.

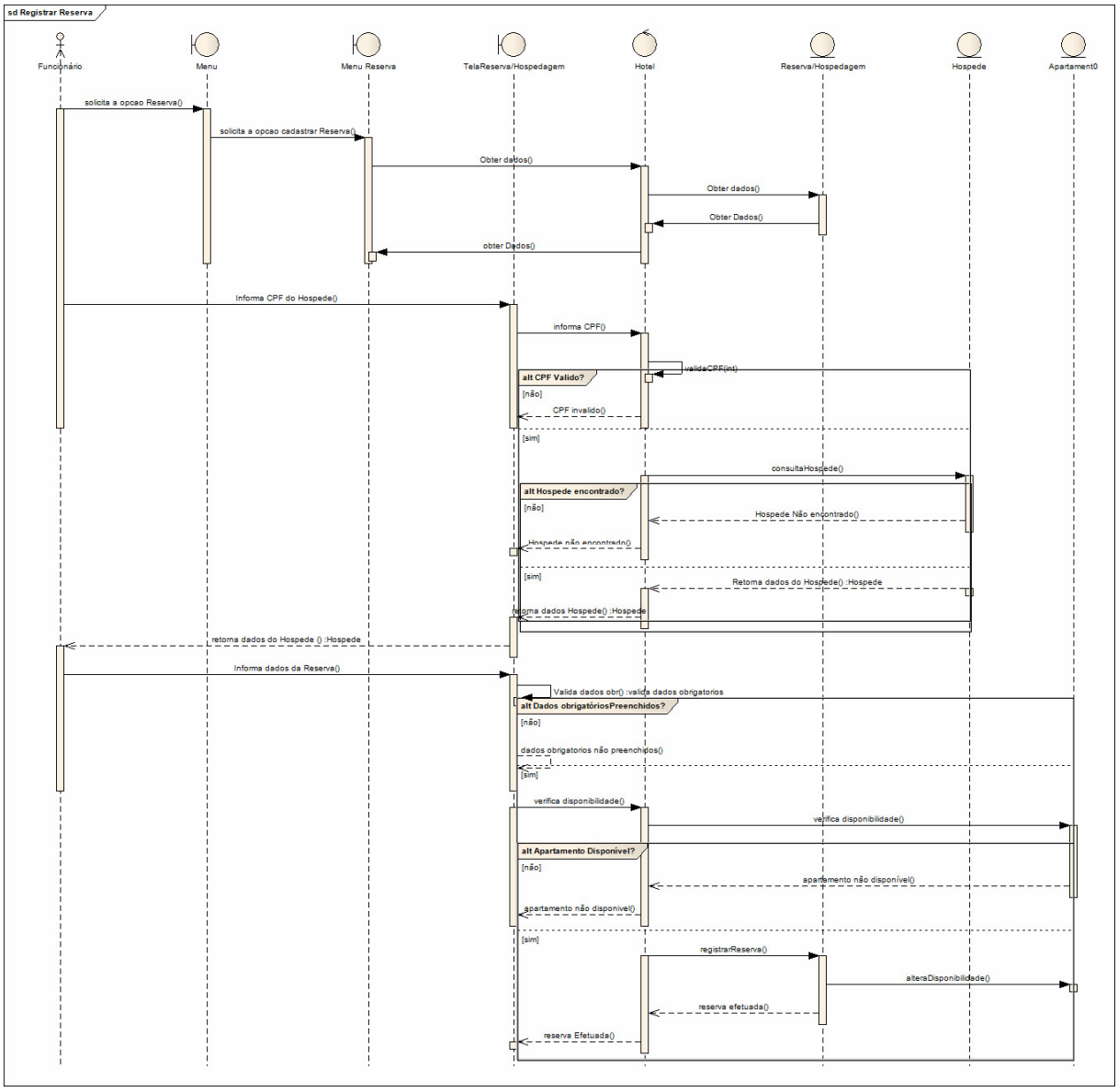

Figura 10 – Diagrama de seqüência Registrar Reserva

O Diagrama de Sequência Consultar Reserva descreve os passos que devem

ser seguidos para consultar os dados de uma determinada reserva feita no hotel.

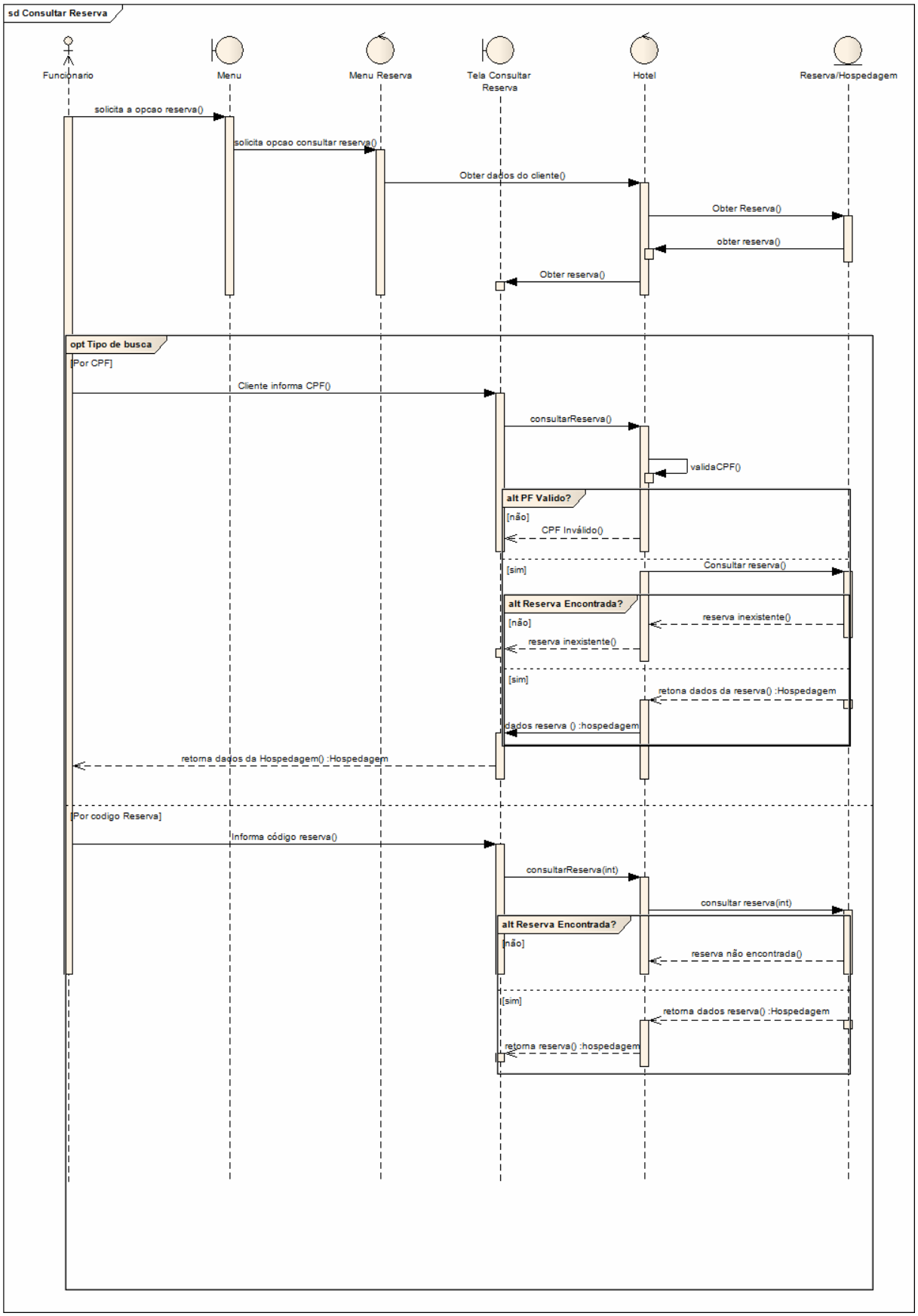

Figura 11 – Diagrama de seqüência Consultar Reserva

O Diagrama de Sequência Registrar Entrada por Reserva descreve os passos que devem ser seguidos para efetuar a entrada do hóspede através de uma reserva efetuada no hotel.

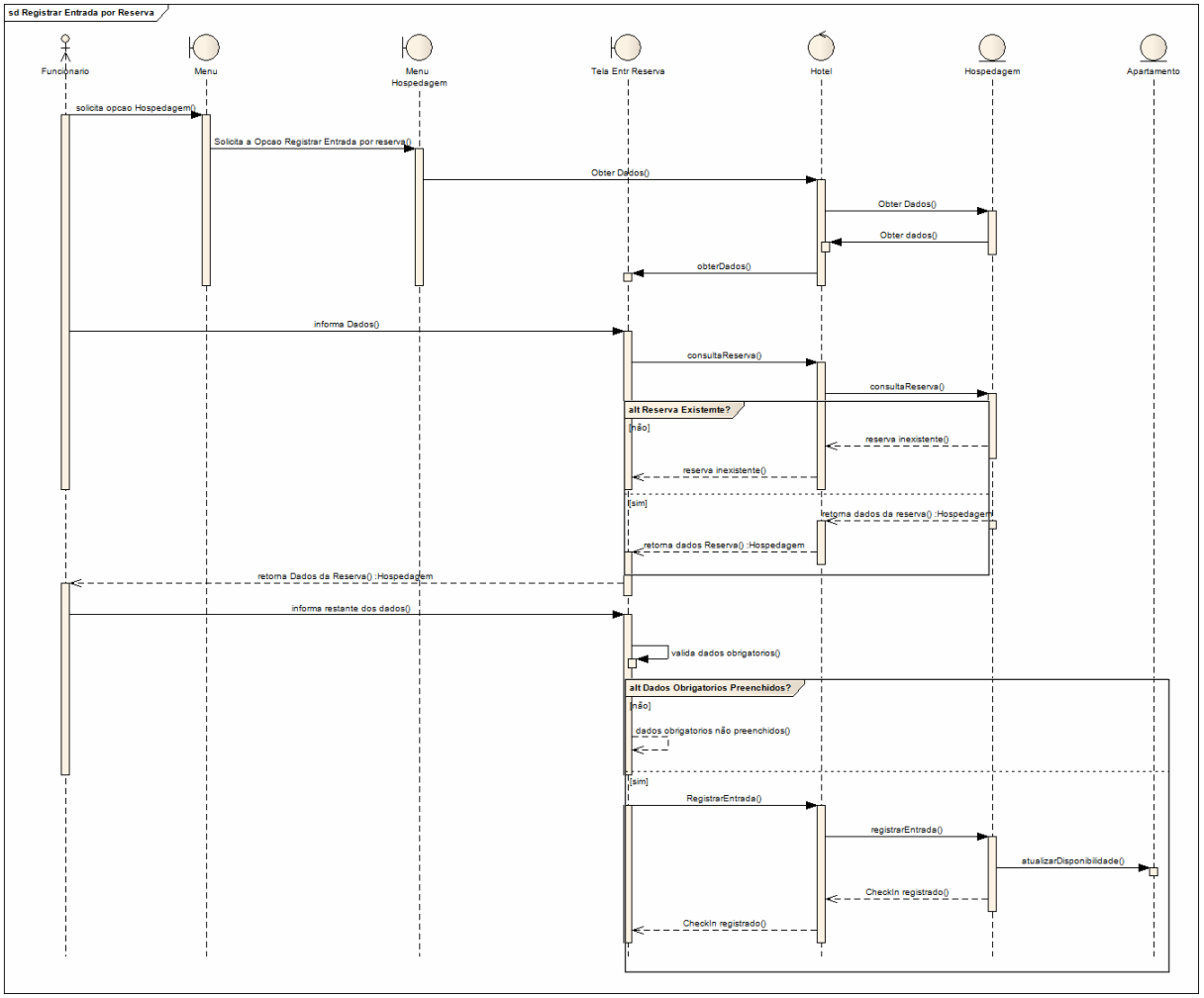

Figura 12 – Diagrama de seqüência Registrar Entrada por Reserva

O Diagrama de Sequência Manter Registrar Entrada descreve os passos que devem ser seguidos para alterar alguam informação referente ao Registro de Hospedagem de um determinado hóspede.

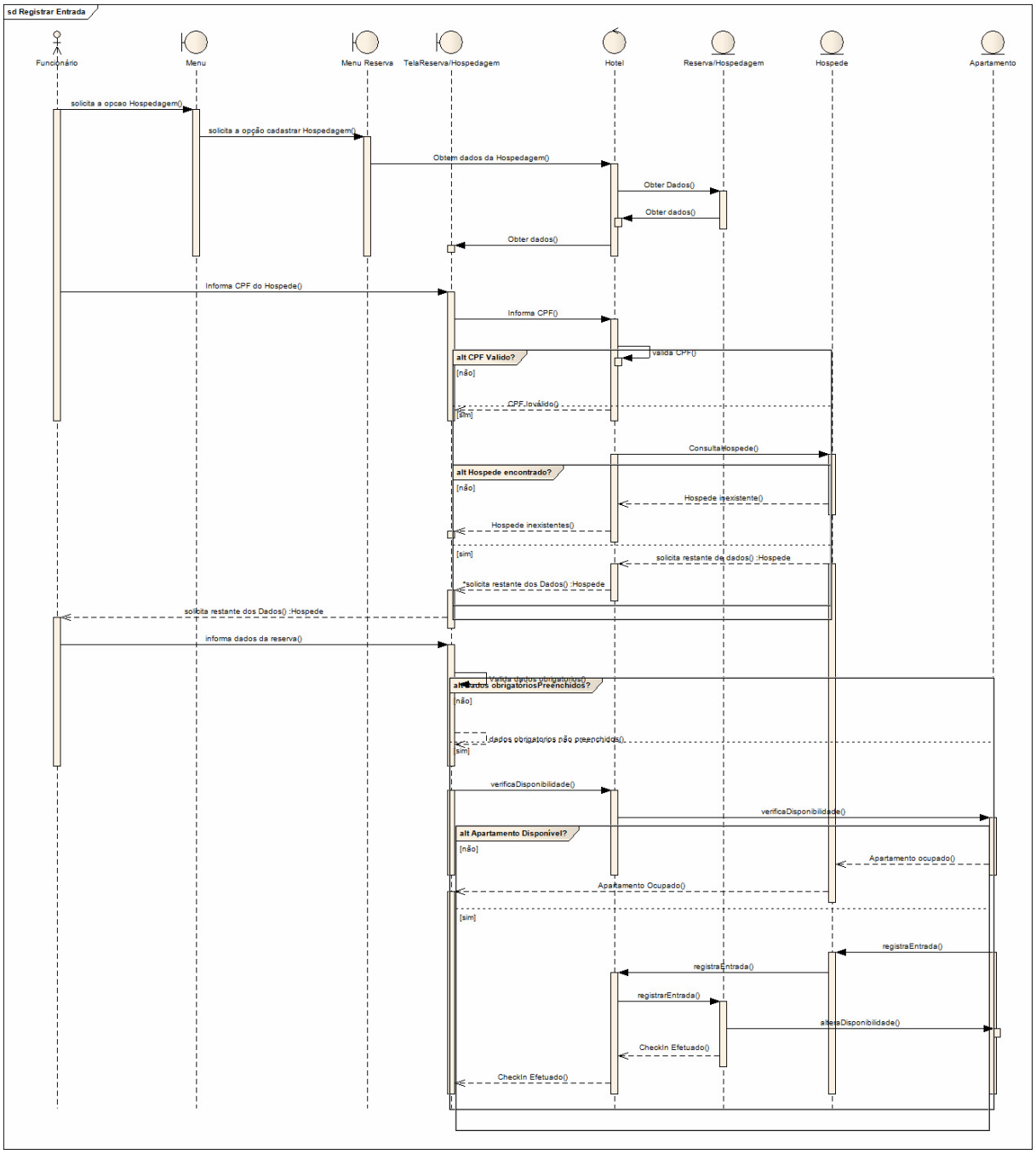

Figura 13 – Diagrama de seqüência Manter Registrar Entrada

O Diagrama de Sequência Registrar Saída descreve os passos que devem ser seguidos para Saída do hóspede no hotel.

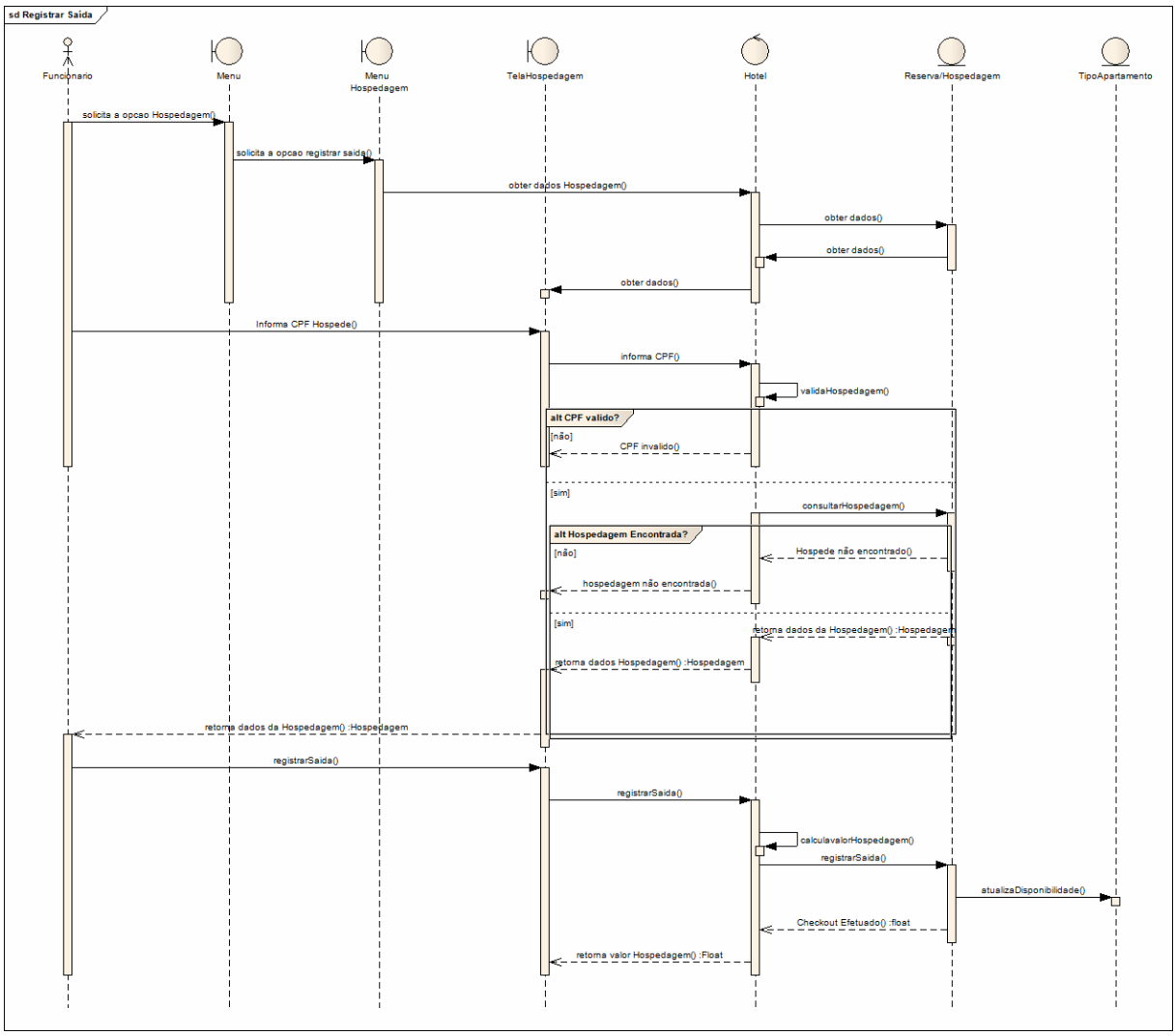

Figura 14 – Diagrama de seqüência Registrar Saída

O Diagrama de Sequência Gerar Mapa de Acomodações descreve os passos

que devem ser seguidos para gerar o relatório contendo os apartamentos disponíveis, reservados e hospedados do hotel.

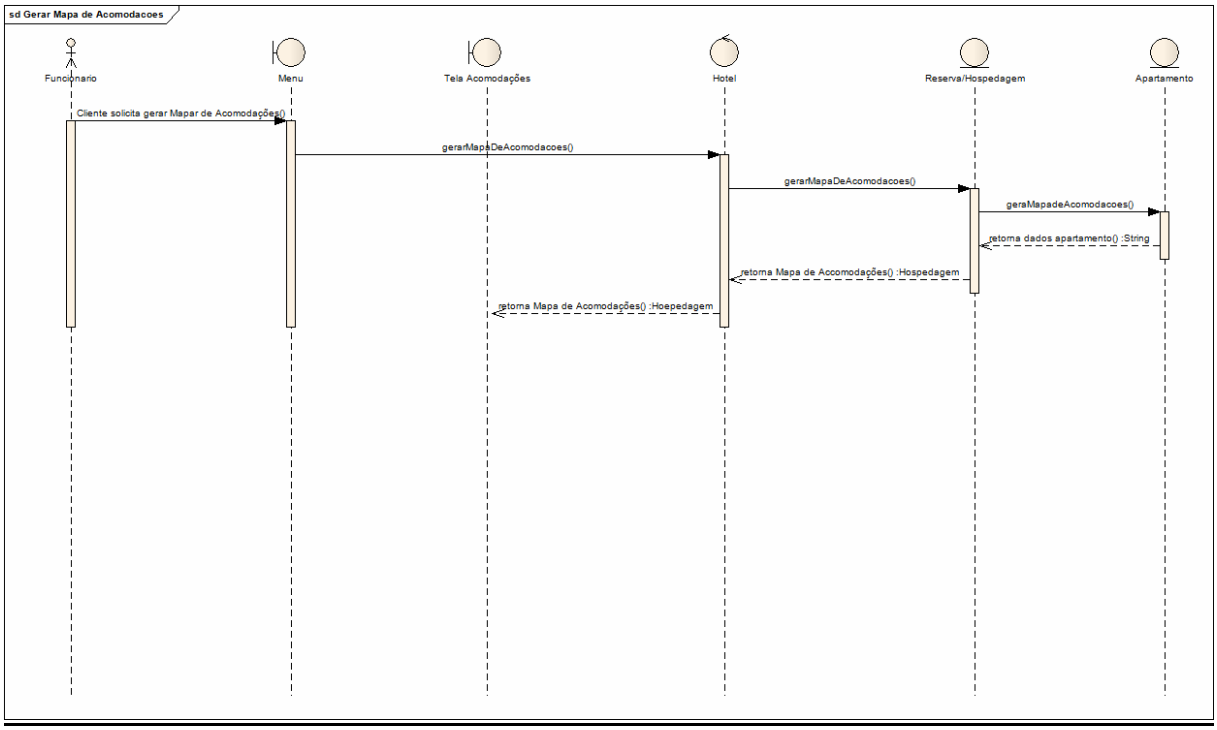

Figura 15 – Diagrama de seqüência Gerar Mapa de Acomodações

O Diagrama de Sequência Efetuar Login descreve os passos que devem ser seguidos para entrar no sistema.

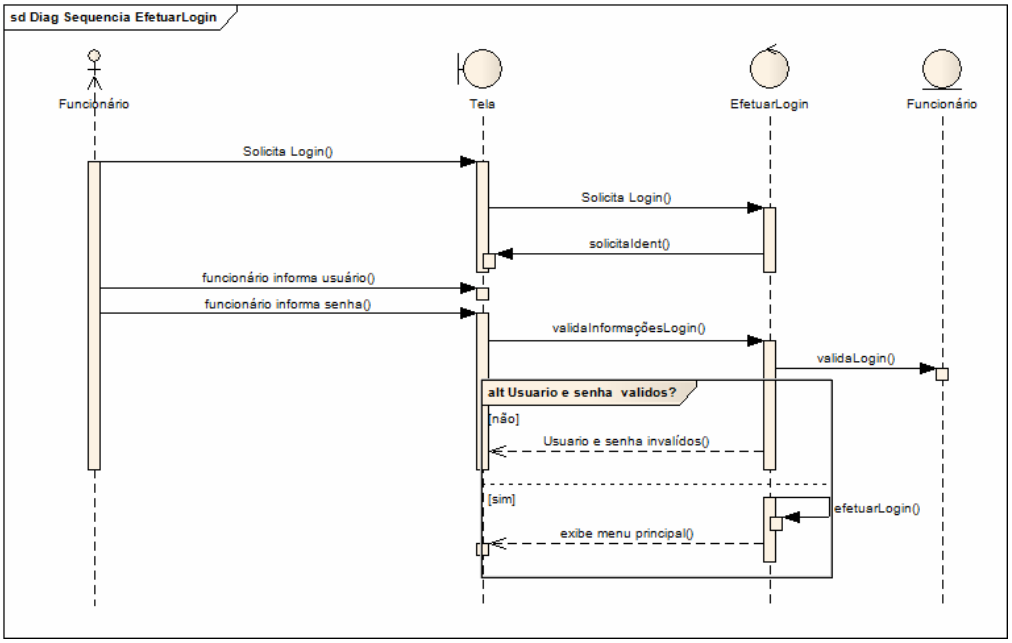

Figura 16 – Diagrama de seqüência EfetuarLogin

3.6 Classes de Análise

As classes de análise são utilizadas na obtenção das principais classes que compõem o projeto desenvolvido, através de uma descrição textual dos Casos de Uso busca–se identificar as classes de análise, as quais refletem as principais abstrações do sistema.

As classes de análise que o sistema é composto dividem–se de acordo com suas responsabilidades em: classes de fronteira, classes de entidade e classes de controle.

3.6.1 Classes de Fronteira

Uma classe de fronteira (*Boundary Class*) representa os resultados de uma interação dos objetos internos, em algo que possa ser entendido pelo ator. Os aspectos considerados são:

- Coordenação do comportamento do atores com os "aspectos internos" do sistema;
- Recebimento de entradas (informações ou requisições) provenientes do ator para o sistema;
- Fornecimento de saídas (informações armazenadas ou resultados derivados) provenientes do sistema para o ator.

Estas classes são responsáveis em fazer a interface com o usuário, ou seja, fazem a interação com os usuários do sistema. No caso do nosso sistema as classes de extensão JSP e as páginas de extensão HTML são as responsáveis por desempenhar essa função. A figura 17 ilustra as classes de fronteira contidas neste projeto

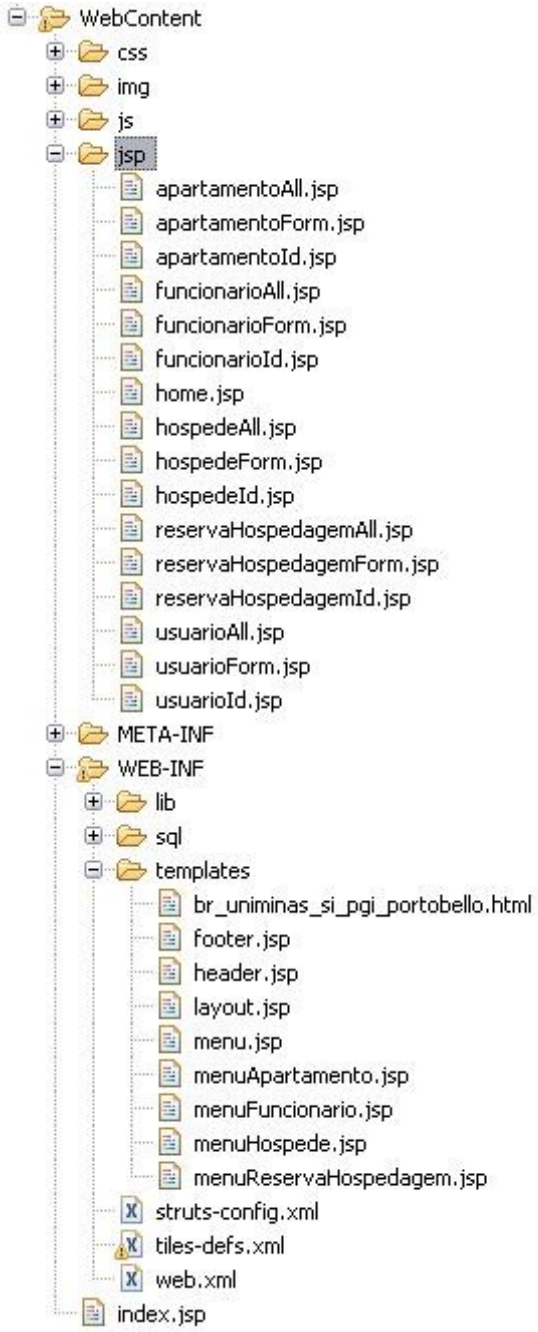

Figura 17 – *Classes de Fronteira*

3.6.2 Classes de Entidade

A classe de entidade representa os conceitos do domínio do negócio cuja informação e os comportamentos associados são, de maneira geral, persistentes através de um arquivo ou banco de dados. As classes de entidade do sistema e

suas responsabilidades são:

- **PessoaBO:** Classe genérica para todas as pessoas cadastradas no sistema, tanto os funcionários quanto aos hóspedes. No banco as tabelas funcionário e cliente serão “filhas” da tabela pessoas que faz ligação à classe Pessoas. As informações pertinentes a essa entidade são: nome da pessoa, registro geral (RG), data de nascimento, endereço, cidade de residência, unidade federativa, cadastro de pessoa física (CPF), telefone de contato e e–mail.
- **FuncionarioBO:** Essa classe é responsável pelos dados dos funcionários, dados estes que serão armazenadas na tabela “funcionario”, e os campos que persistiram além dos referidos na classe Pessoa são: matricula do funcionário, data de admissão, código e série do CTPS, número do PIS, cargo exercido e caso exista data da demissão.
- **HóspedeBO:** Classe responsável pelas informações pertinentes aos hóspedes do hotel que são: numero do cadastro e as preferências do hóspede.
- **UsuarioBO:** Nessa classe serão pertinentes informações referentes ao acesso dos funcionários do hotel ao sistema. Tendo em vista que nem todos os funcionários terão o mesmo nível de acesso ao sistema informações como nome de login, senha, status do usuário e grupo de acesso são necessárias para o controle de acesso.
- **ReservaHospedagemBO:** A classe hospedagem é o “coração” do sistema, pois tudo gira em torno dela. Os Casos de Uso Check In e Check Out juntos fornecem todas a informações necessárias para esta classe, sendo assim nem todas as informações são inseridas na tabela reserva_hospedagem, no ato da inserção no banco. As informações pertinentes são: data de entrada e data de saída (no caso de ser feita uma reserva antes da hospedagem), data do check i e data do check out (em todos os casos, seja uma hospedagem por reserva ou não), valor da hospedagem, valor d o desconto, valor da multa, número do apartamento e status do apartamento.
- **ApartamentoBO:** Na classe apartamento ficam registradas as informações referentes aos quartos existentes no hotel, como: andar do apartamento, descrição do apartamento e tipo do apartamento.

- **TipoApartamentoBO:** Informações referentes aos diferentes tipos de apartamentos como: descrição do tipo de apartamento e o valor da diária desse tipo de apartamento.

3.6.3 Classes de Controle

As classes de controle são responsáveis pela comunicação entre as classes de fronteira e as classes de entidade e representa coordenação, seqüência, transações e controle de outros objetos.

Essas classes são utilizadas para encapsular controle relacionado a um ou mais casos de uso e representam um processamento complexo (cálculos envolvidos na lógica de um processo de negócio) que não seja apropriado a uma classe de entidade em particular. Suas principais características de controle são:

- Criação, ativação e anulação objetos.
- Controla a operação de objetos.
- Controla a concorrência de pedidos de objetos.
- Trabalha somente com sinais (não com dados normais).
- Não sabe como fazer, mas sabe quem deve fazer (qual objeto controlado que faz).

No projeto utilizou–se a tecnologia Struts para o controle da aplicação. As classes do Struts usadas na nossa aplicação foram:

- **ActionServlet:** Essa classe obtém as requisições e as disponibiliza em um ActionForm, e delega para o Action específico.

3.6.4 Diagrama de Classe de Projeto

O diagrama de classes na fase de projeto deve especificar as classes de software e as interfaces da aplicação, diferentemente do diagrama de classes da fase de analise que é considerado um diagrama de classes no modelo conceitual,

onde uma entidade não representa uma classe de software, mas um conceito do domínio do problema.

Para construir um diagrama de classes na fase de projetos deve–se:

- Identificar todas as classes que participam da solução em software, e incluí–las num diagrama de classes.
- Duplicar os atributos a partir das entidades associadas no modelo conceitual.
- Adicionar informação de tipo aos atributos e métodos.
- Adicionar as associações para indicar a direção da visibilidade de atributos.
- Adicionar linhas de relações de dependência para indicar a visibilidade que não seja de atributo.

As associações são necessárias somente para visibilidade de objetos, diferentemente das associações do modelo conceitual que são inseridas para melhorar o entendimento.

Os diagramas de interações são utilizados para descobrir a visibilidade, associações e navegabilidade do diagrama de classe.

Após a definição das classes que vão ser utilizadas no diagrama, são acrescentados alguns padrões para alcançar o nível de organização.

O diagrama da figura 18 mostra uma visão macro de todo o sistema permitindo a visão da organização do mesmo em pacotes:

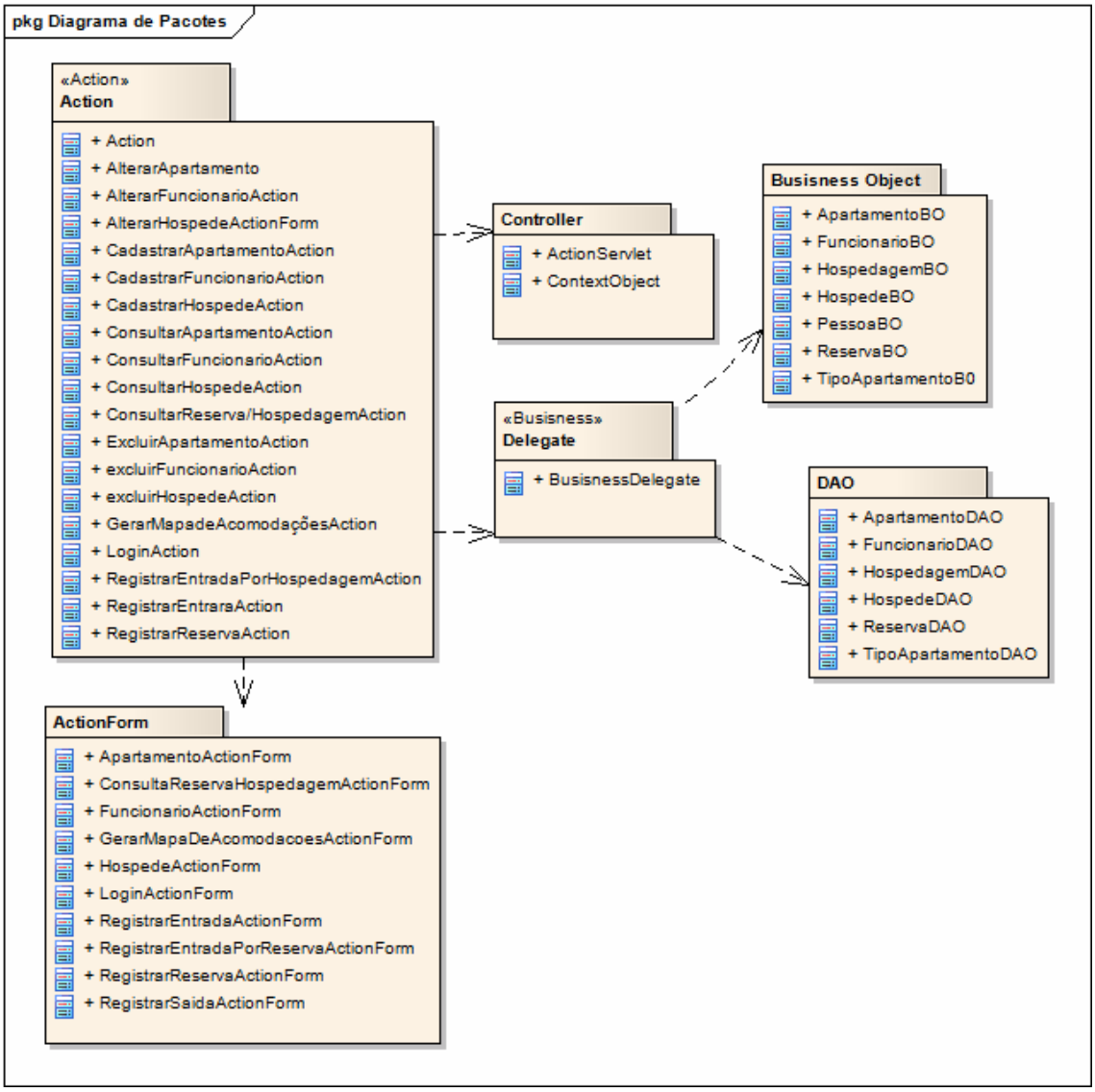

Figura 18 – Diagrama de Pacotes

Abaixo seguem os diagramas de classe na fase de projeto divididos em interações para que uma seqüência lógica seja estabelecida.

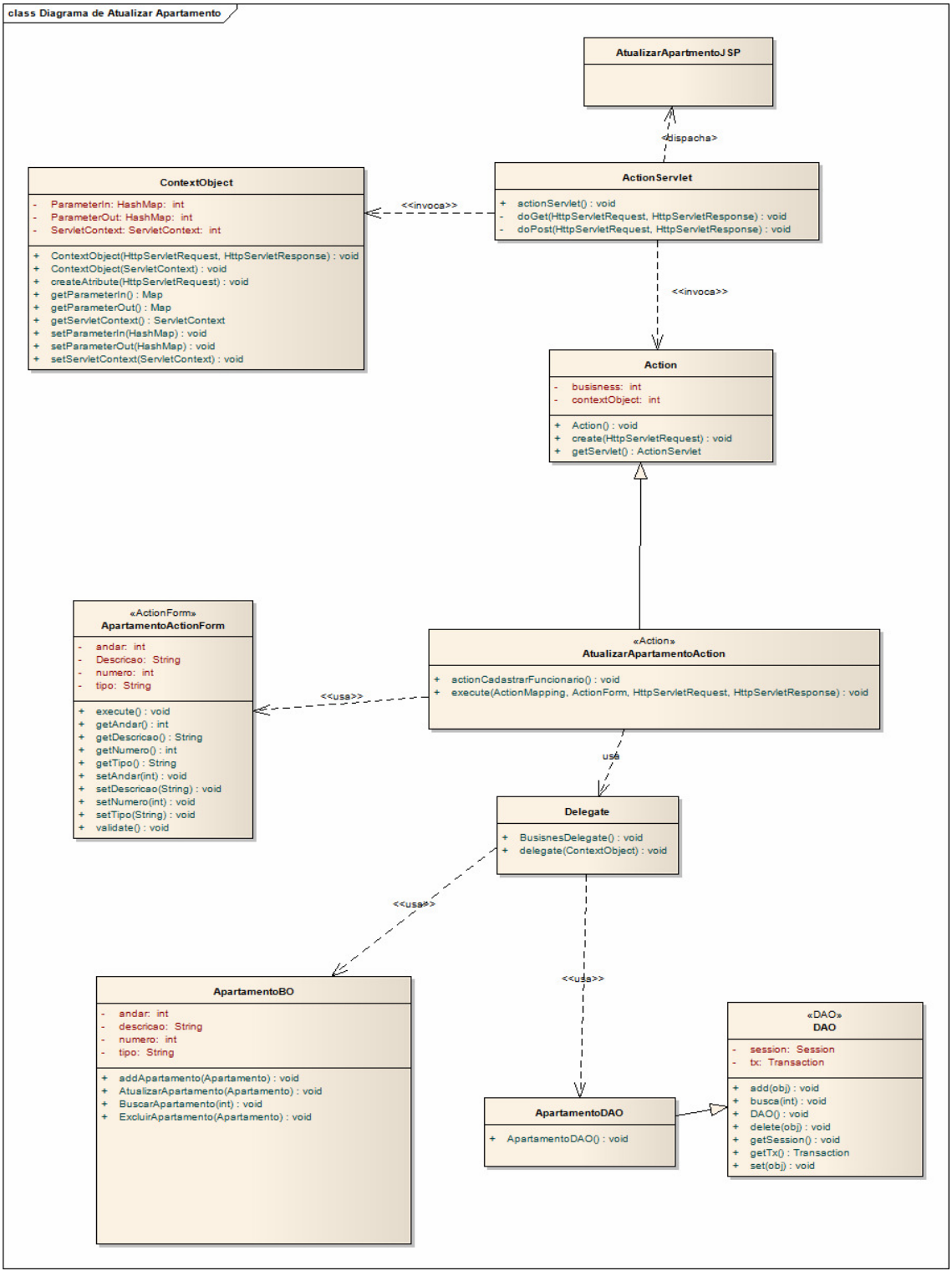

Figura 19 – Diagrama de Classes Atualizar Apartamento

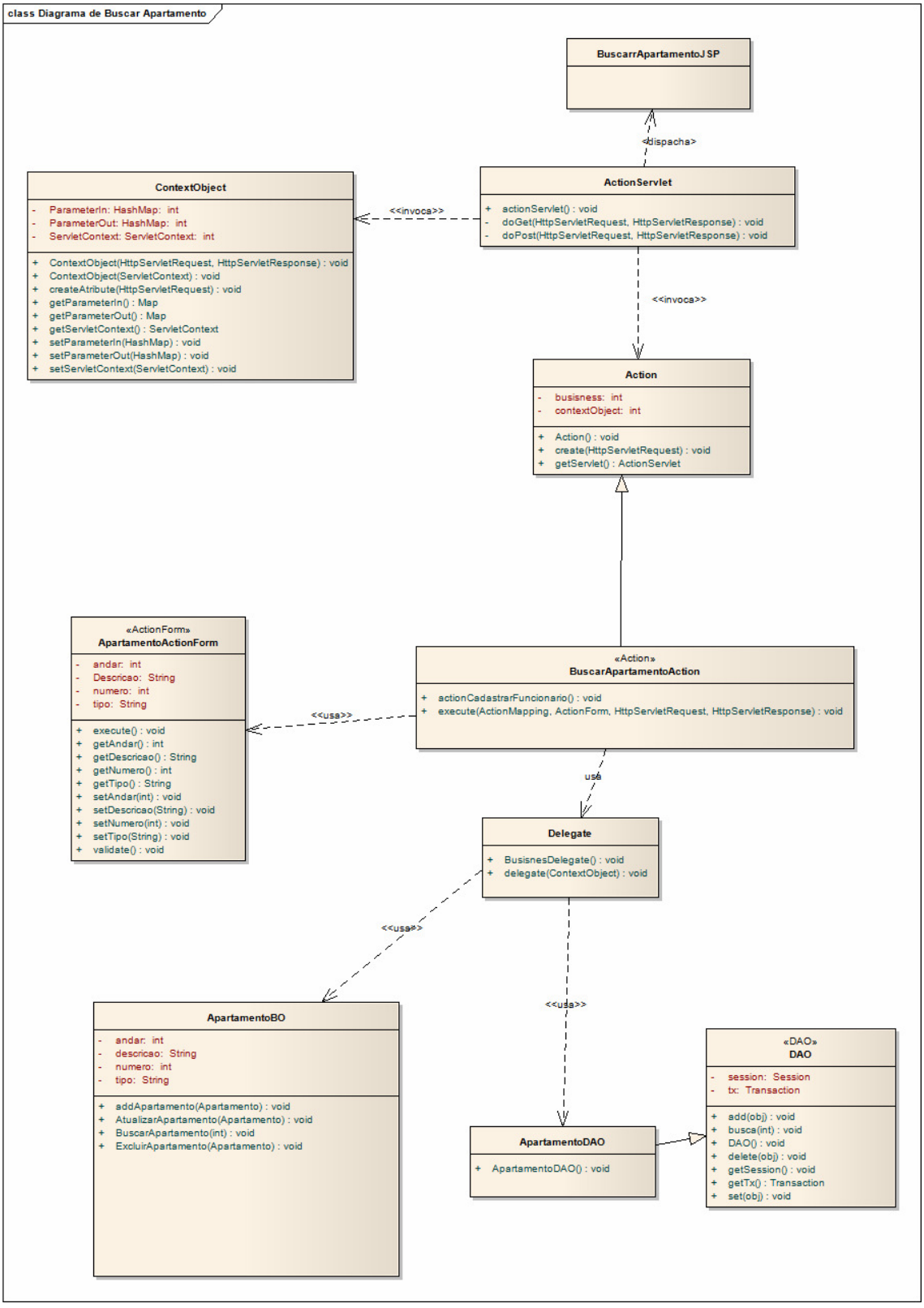

Figura 20 – Diagrama de Classes Buscar Apartamento

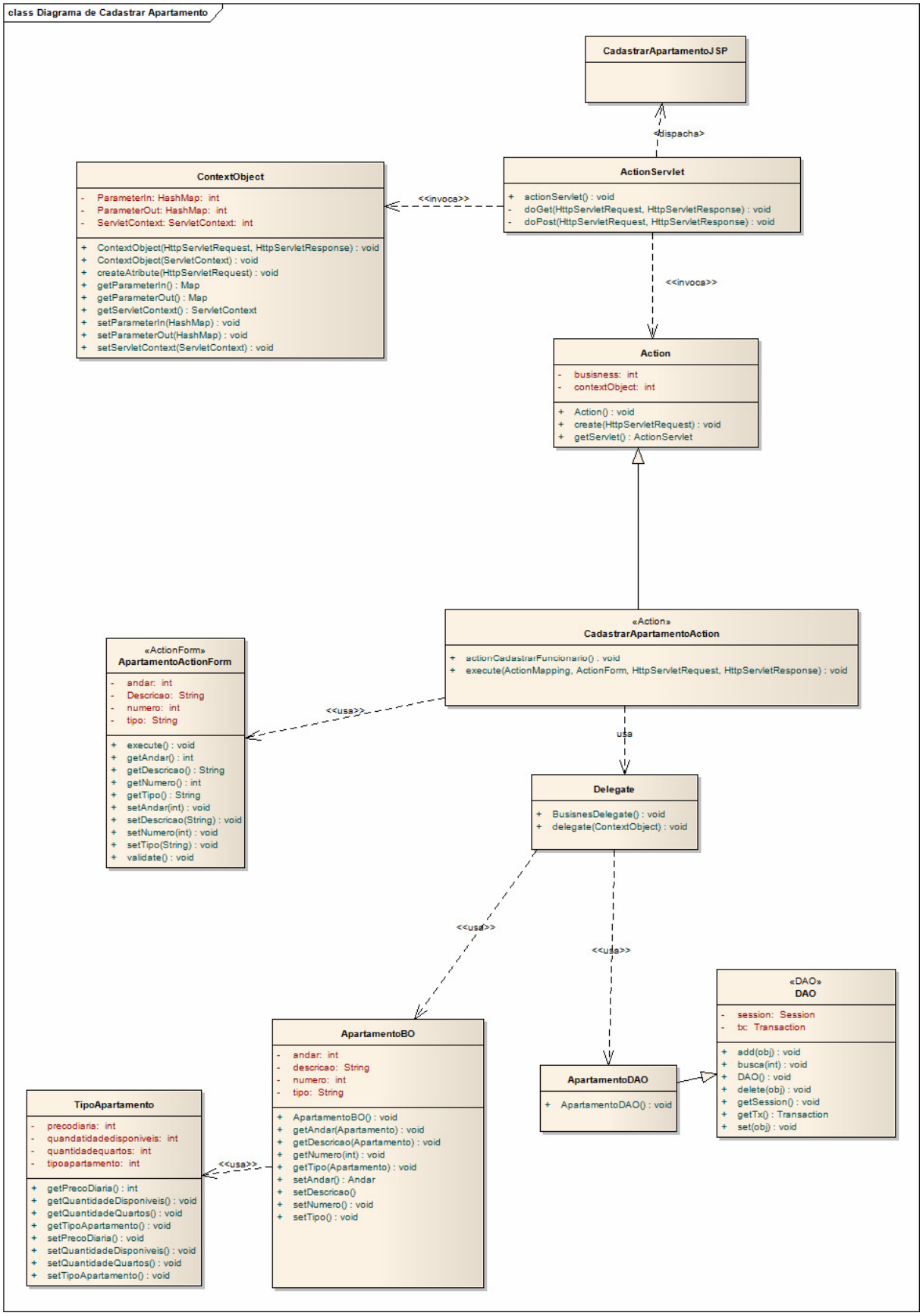

Figura 21– Diagrama de Classes Cadastrar Apartamento

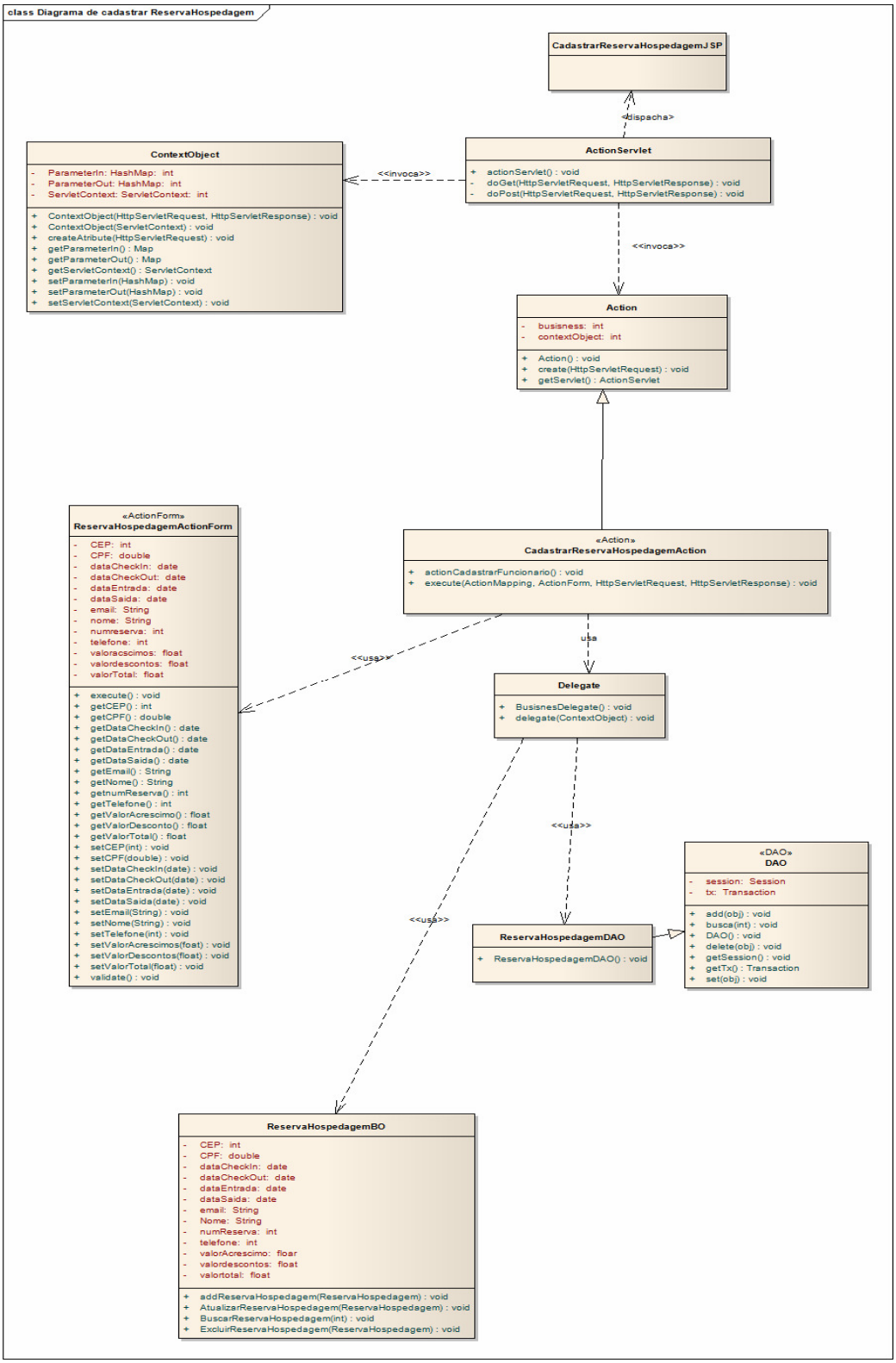

Figura 22 – Diagrama de Classes Cadastar Reserva/Hospedagem

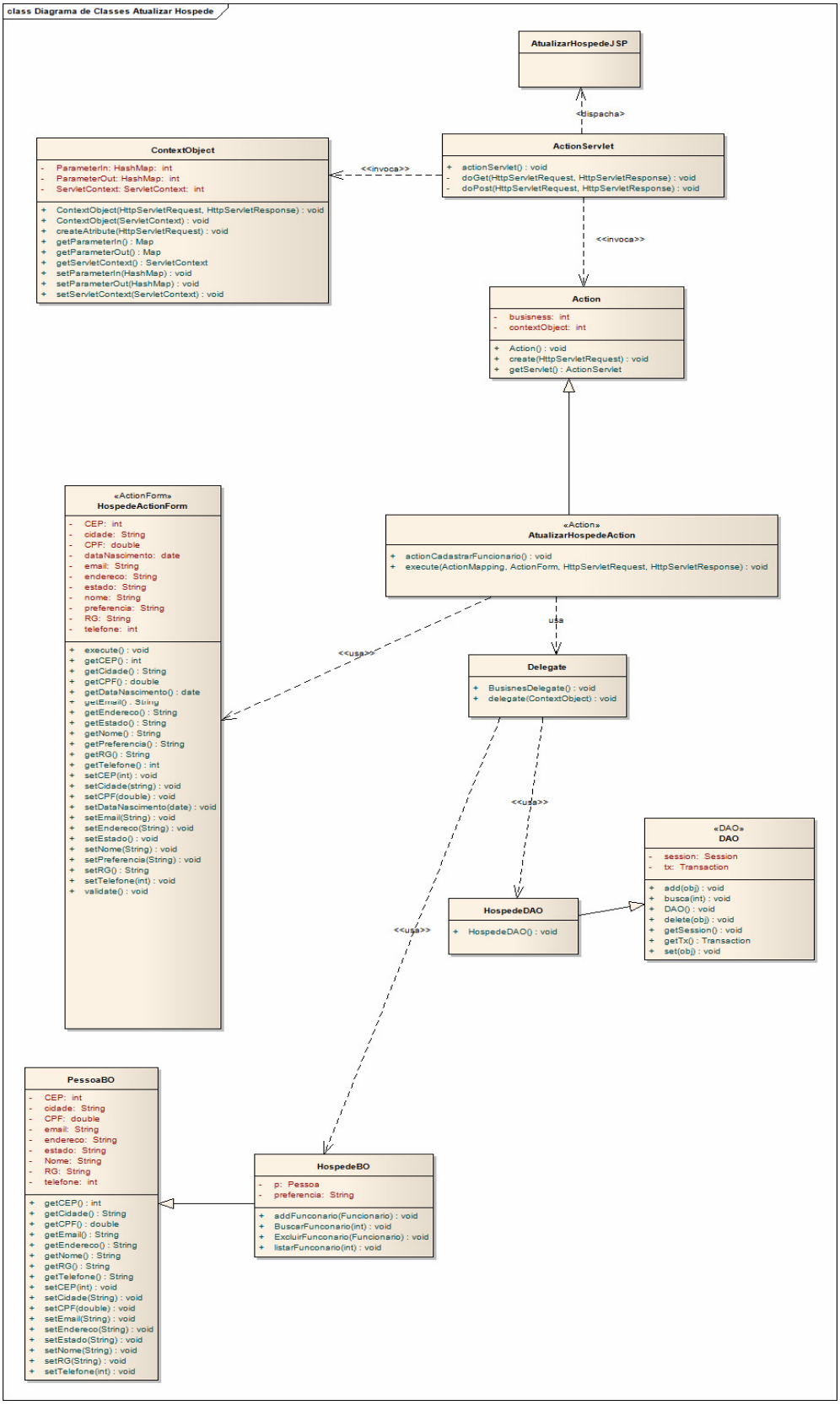

Figura 23 – Diagrama de Classes Atualizar Hospede

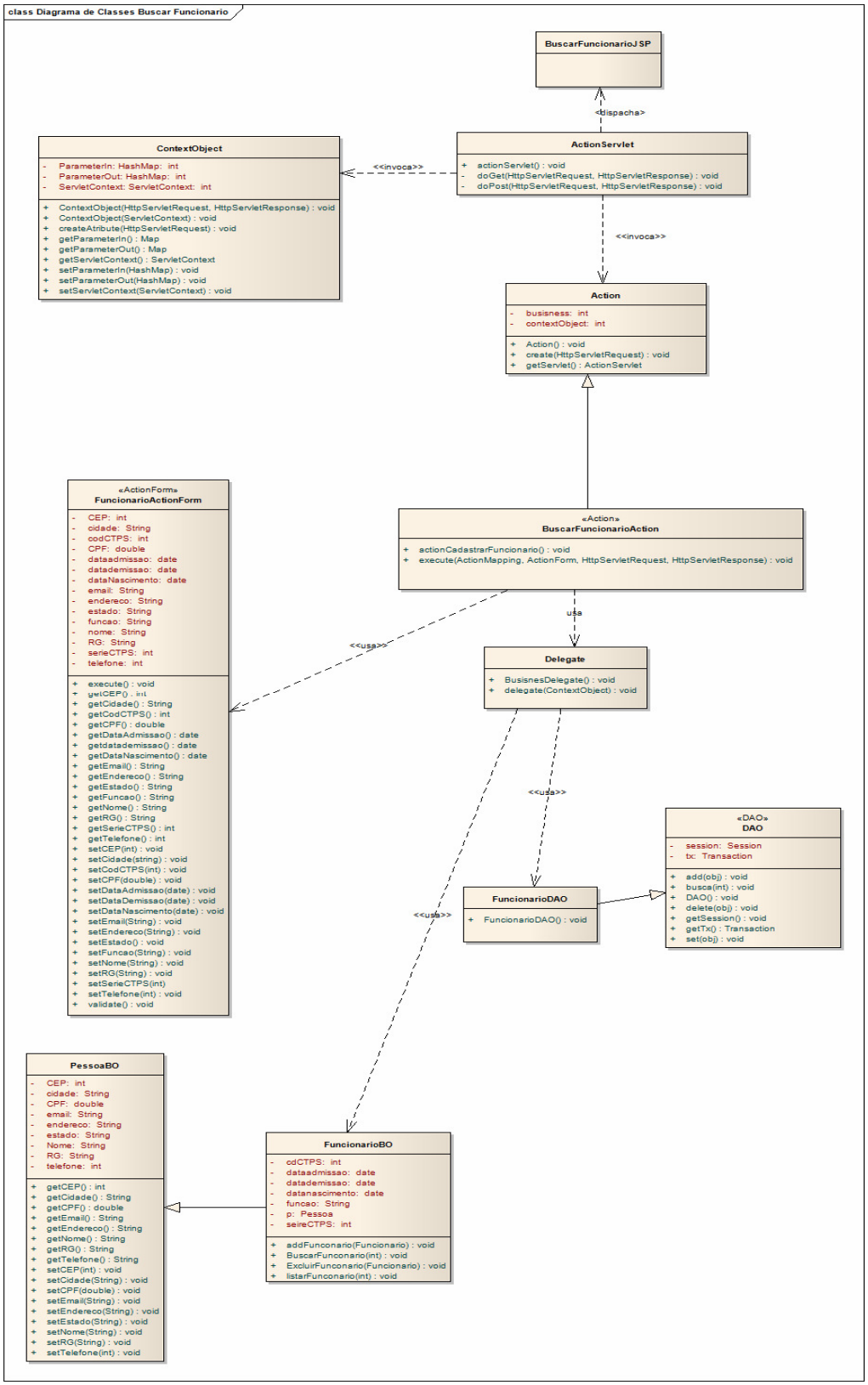

Figura 24 – Diagrama de Classes Buscar Funcionário

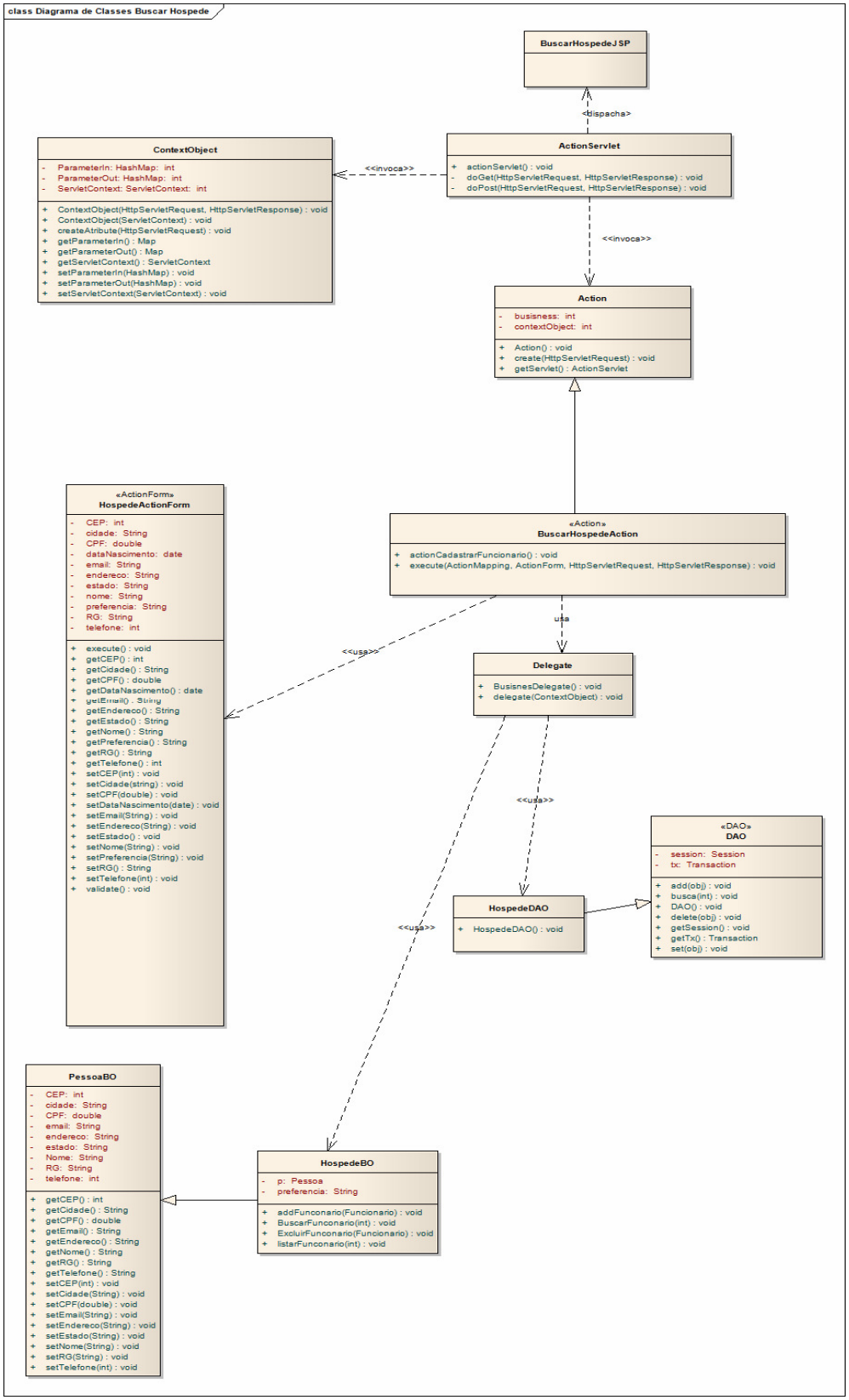

Figura 25 – Diagrama de Classes Buscar Hóspede

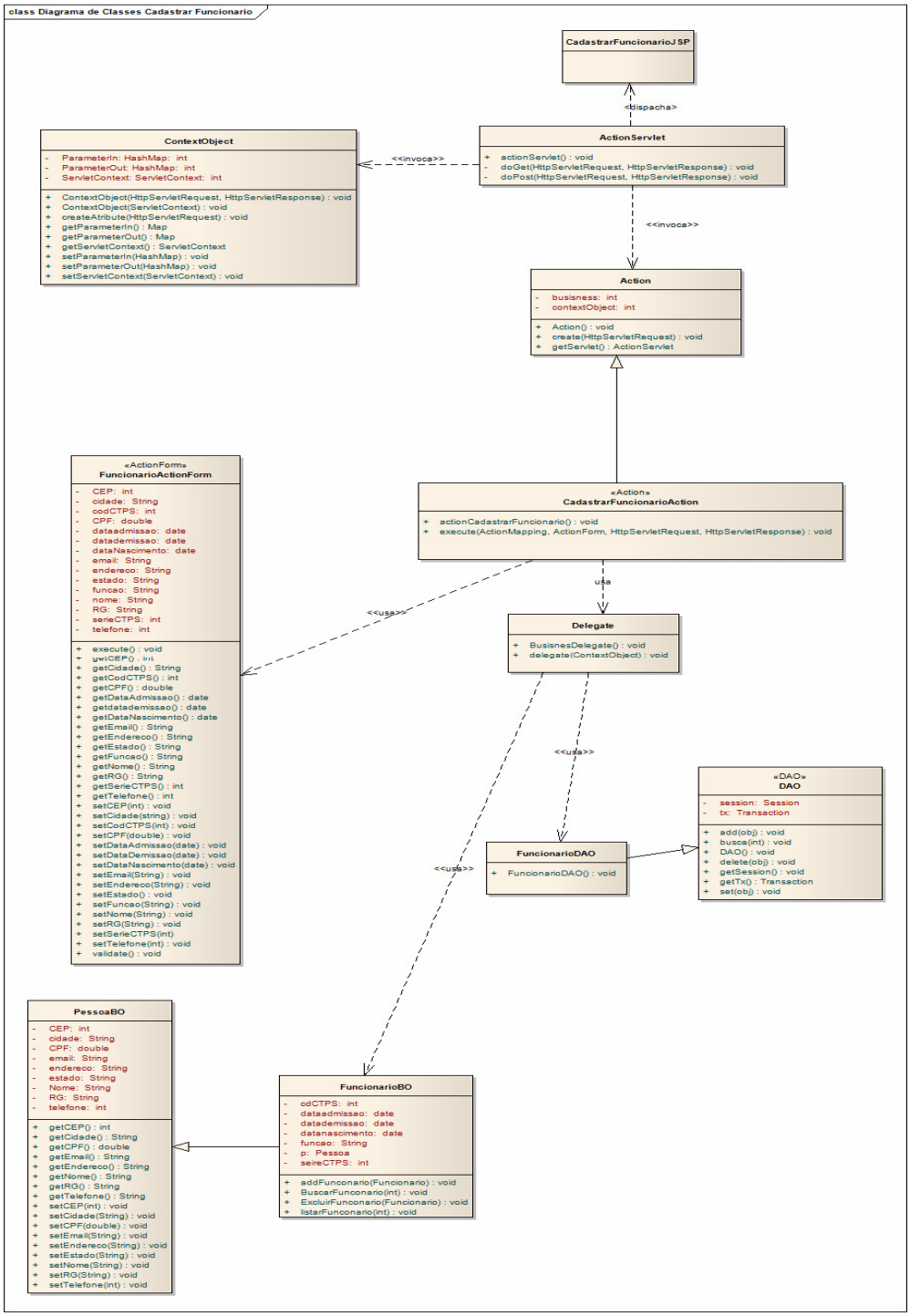

Figura 26 – Diagrama de Classes Cadastrar Funcionário

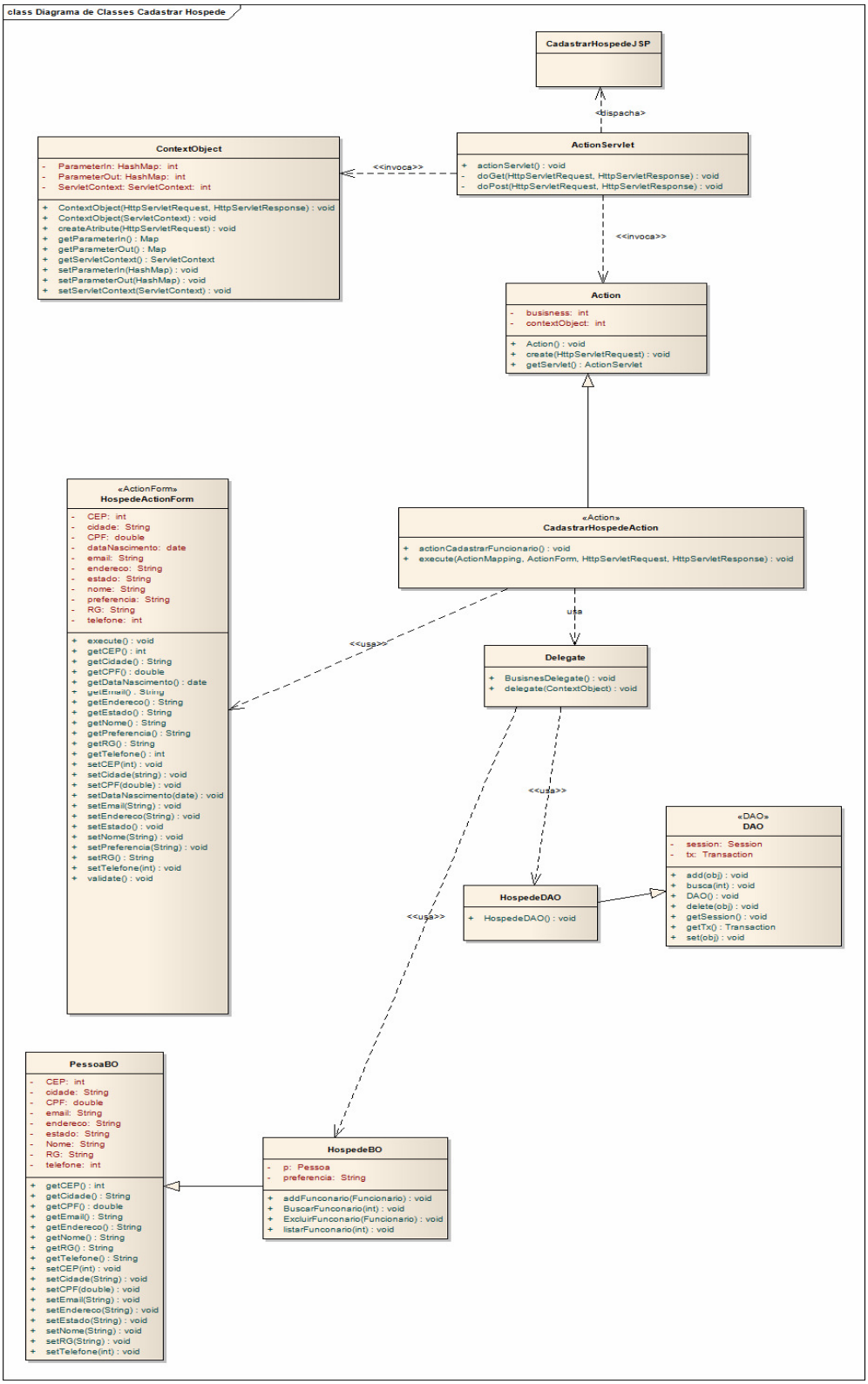

Figura 27 – Diagrama de Classes Cadastrar Hóspede

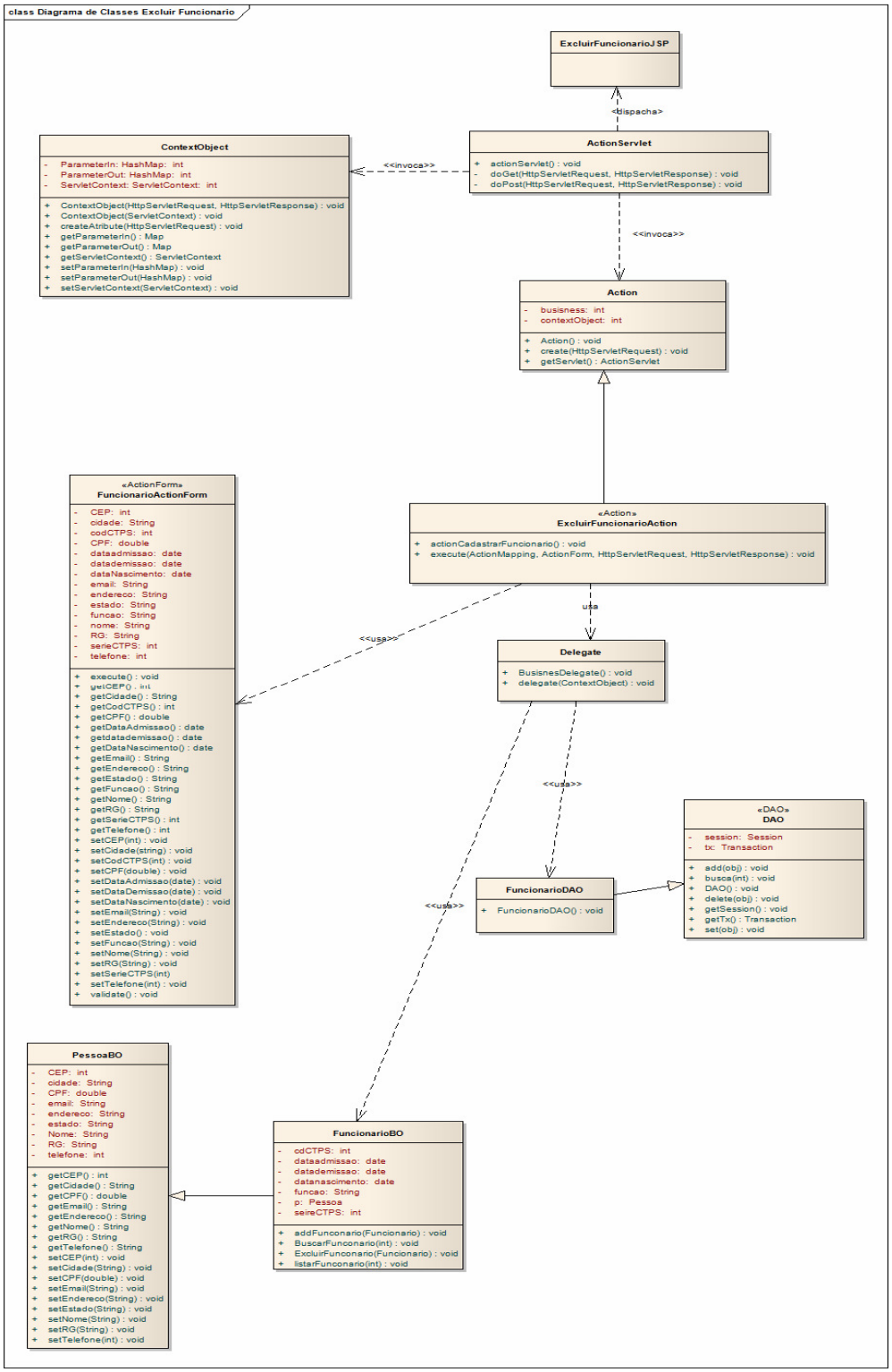

Figura 28 – Diagrama de Classes Excluir Funcionário

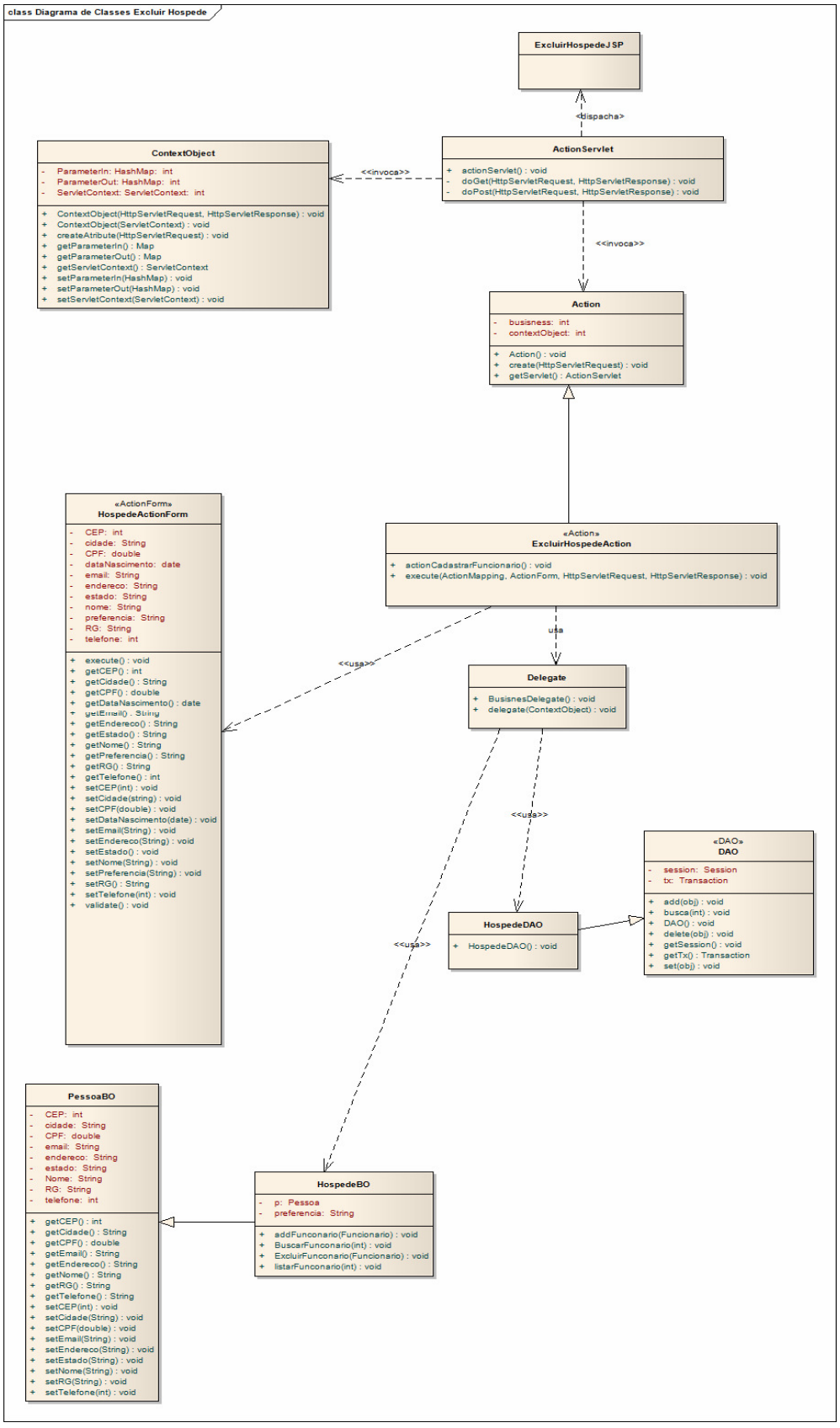

Figura 29 – Diagrama de Classes Excluir Hóspede

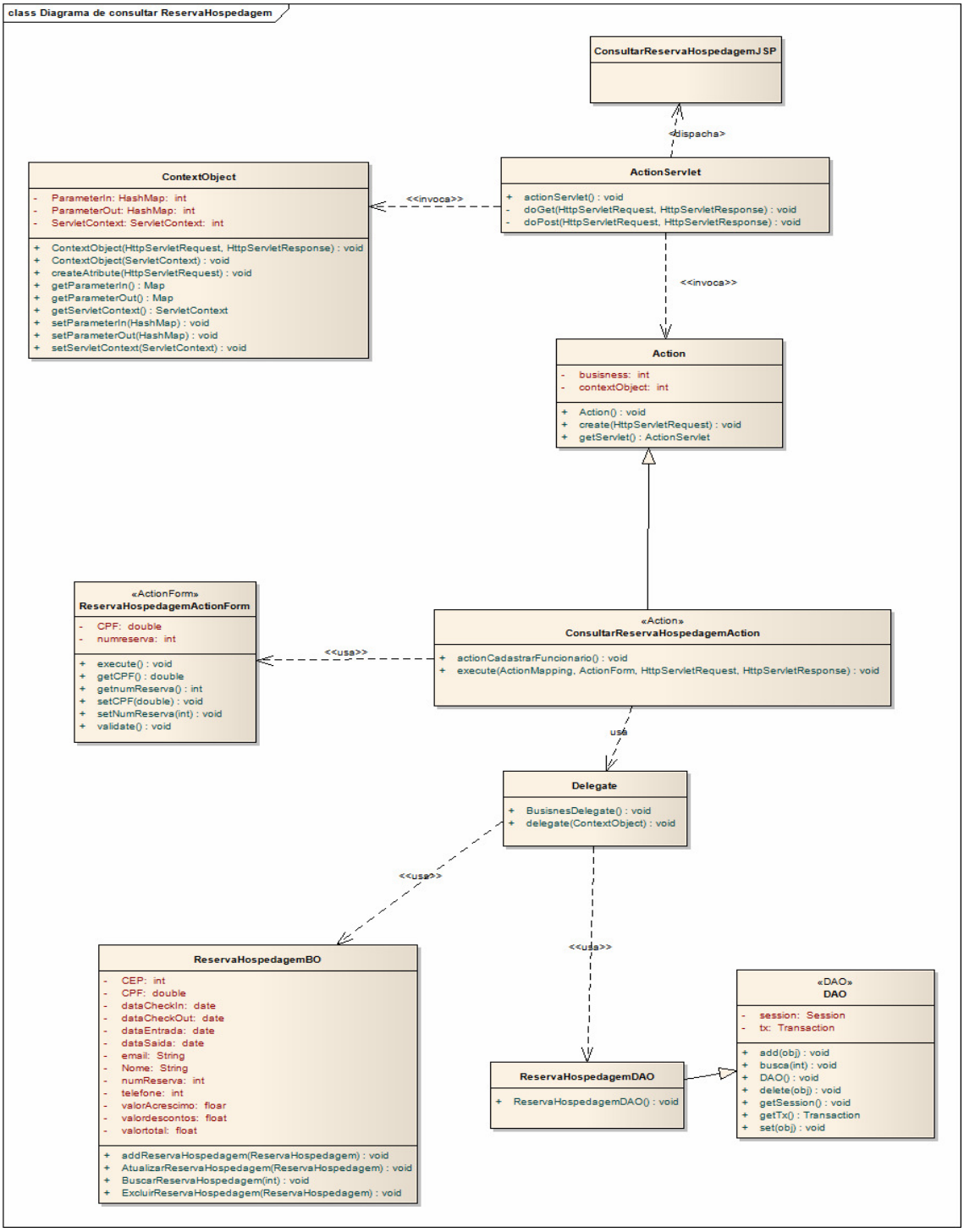

Figura 30 – Diagrama de Classes Consultar Reserva Hospedagem

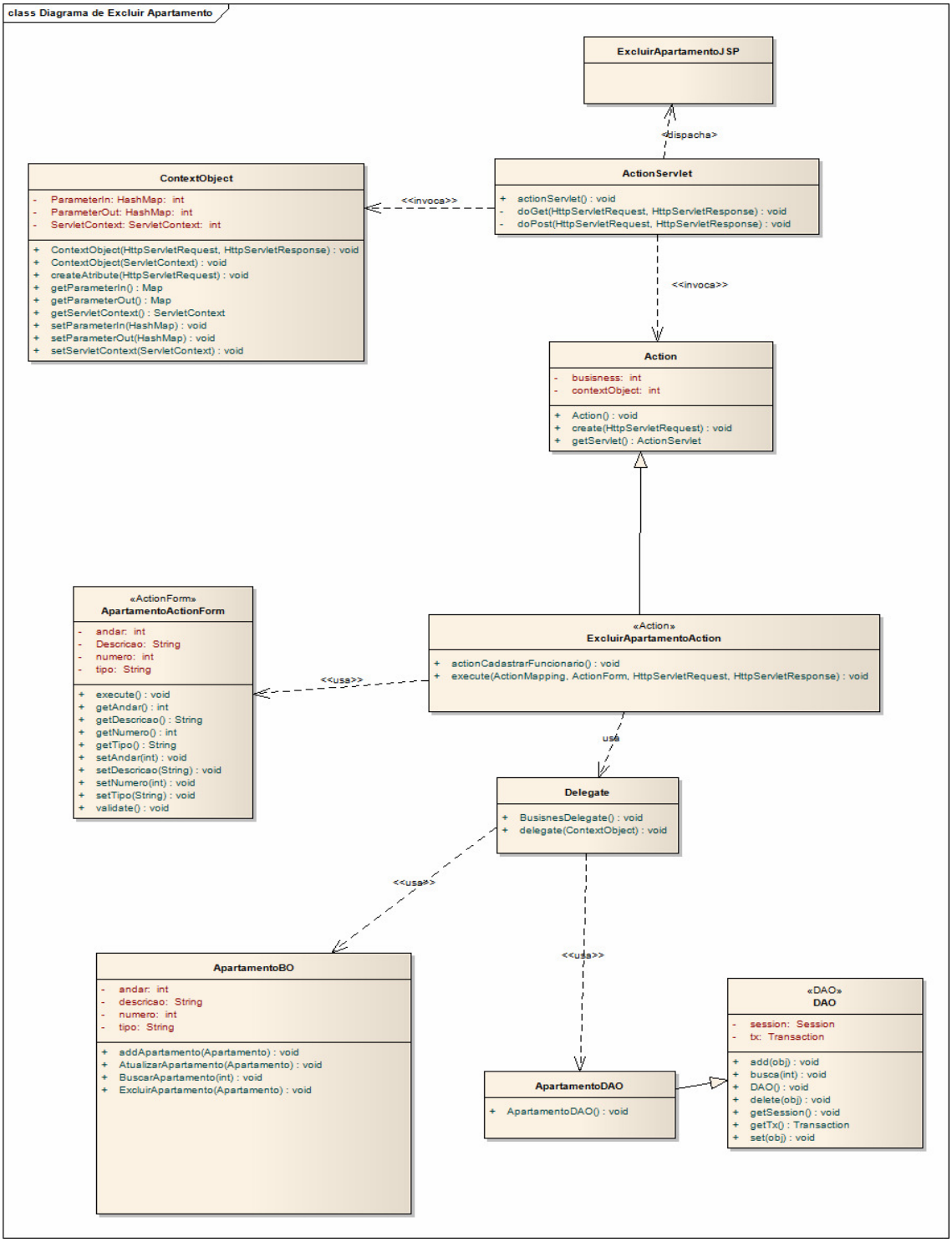

Figura 31 – Diagrama de Classes Excluir Apartamento

4 ESTRUTURA E CÓDIGO

4.1 Arquitetura

Desde sua criação, o desenvolvimento de sistemas tem lidado com problemas associados com a complexidade da informação. Os primeiros problemas de complexidade foram resolvidos pelos desenvolvedores através da escolha da estrutura de dados, do desenvolvimento de algoritmos e pela aplicação de conceitos de separação de escopos.

Embora o significado de arquitetura seja relativamente amplo, os princípios fundamentais deste campo vêm sendo aplicados esporadicamente pela engenharia de software desde o início dos anos 80. As primeiras tentativas de capturar e explicar a arquitetura de software de sistema foram imprecisas e desorganizadas, freqüentemente caracterizadas por um conjunto de diagramas.

> A Arquitetura de Software de um sistema consiste dos componentes de software, propriedades externas, e seus relacionamentos com outros softwares. O termo também se refere à documentação da arquitetura de software do sistema. A documentação facilita a comunicação entre os stakeholders, registra as decisões iniciais acerca do projeto de alto-nível, e permite o reuso do projeto dos componentes e padrões entre projetos. (http://pt.wikipedia.org/wiki/Arquitetura_de_software)

Durante o decorrer da década de 90, houve um esforço concentrado para definir e codificar os aspectos fundamentais sobre arquitetura. Inicialmente, um conjunto de padrões de projeto, estilo, melhores práticas, descrição de linguagens, e lógica formal foi desenvolvido durante este período. Com base nesses estudos feitos, a fim de encontrar uma melhor prática para a elaboração de sistemas e desenvolver um software da forma que o cliente necessita, foram criados padrões de projetos que auxiliam nesse desenvolvimento com a finalidade de simplificar a codificação com base na solicitação do cliente.

Porém, para desenvolver um software com qualidade, é necessário entender o que o cliente quer, de tal forma que sua perspectiva esteja de acordo com a forma no qual a arquitetura foi aplicada. Este caso envolve um paradigma de programação orientada a objetos ou POO (Programação Orientada a Objetos) ou do inglês OOP

(*Object Oriented Programmimg*), que se baseia em identificar os objetos fornecidos no levantamento feito com o cliente e aplicar esses objetos no software a ser criado, mantendo assim um comportamento semelhante e funcional conforme os requisitos dessa aplicação.

O desenvolvimento da aplicação do Porto Bello *Palace* Hotel foi implementado na linguagem JAVA baseado para a web (JAVA EE – *Java Enterprise Edition*), utilizando os *frameworks Struts, Hibernate* e *Tiles*.

4.2 JAVA EE – Java Enterprise Edition

Segundo Alur, Crupi e Malks (2004), a plataforma JAVA EE está designada para desenvolvimento de soluções empresariais para sistemas distribuídos.

> J2EE é uma plataforma para desenvolvimento de aplicativos empresariais distribuídos. A linguagem Java, desde sua implementação, teve uma grande aceitação e crescimento. Mais e mais tecnologias para atender a várias necessidades. Eventualmente, a Sun e um grupo de líderes da indústria, sob o s auspícios do Java Community Process (JCP) aberto, unificaram todos os padrões e APIs relativos a empresas na plataforma J2EE. (ALUR, CRUPI & MALKS, 2004)

Os *Design Patterns*, soluções para problemas cotidianos, foram elaborados por arquitetos de software e orientação a objetos. O objetivo era de tornar mais eficiente o desenvolvimento de um software e assim poupar os demais desenvolvedores de "reinventar a roda" toda vez que se deparassem com um problema.

Entende–se, mediante a obra dos autores Alur, Crupi e Malks (2004), que, com a demanda da criação de aplicações distribuídas, cada vez mais empresas procuravam criar sua própria maneira de resolver esses problemas. Com isso, um grupo de arquitetos pensou nas situações mais relevantes no cenário coorporativo e, baseado na necessidade de segurança/escalabilidade, esses arquitetos fizeram um documento por escrito que se tornou a primeira especificação do JAVA EE.

> Os padrões se referem á comunicação de problemas e soluções. Em outras palavras, os padrões permitem documentar um problema conhecido recorrente e sua solução em um contexto específico e comunicar esse conhecimento para outras pessoas. M dos elementos–chave na declaração

> anterior é a palavra recorrente, visto que a meta do padrão é promover uma reutilização do conceitual ao longo do tempo. (ALUR, CRUPI & MALKS, 2004)

No início, o desenvolvimento na especificação JAVA EE era muito complicado, mas, com o tempo, várias mudanças foram acontecendo e se tornou uma tecnologia difundida e muito utilizada em todo o mundo.

A especificação prega as seguintes características:

- **Alta Disponibilidade**: Aplicações para missões críticas, aplicações que não podem parar, serviços que têm que estar funcionando 24 horas por dia, 7 dias por semana;
- **Segurança**: Os dados devem estar muito bem protegidos, tanto em relação à privacidade dos usuários, quanto à segurança das informações do "negócio";
- **Confiável**: Confiável se enquadra no princípio de ser SEGURA e DISPONÍVEL, ou seja, uma aplicação na qual você pode confiar (pois sempre vai funcionar quando você precisa e suas informações estarão seguras com ela);
- **Escalável**: Uma aplicação que cresce seguindo a demanda do seu negócio. Isso quer dizer que, teoricamente, você poderia começar com uma aplicação "pequena" e, conforme a necessidade fosse surgindo, bastaria você adicionar mais "máquinas/recursos" para suportar a demanda.

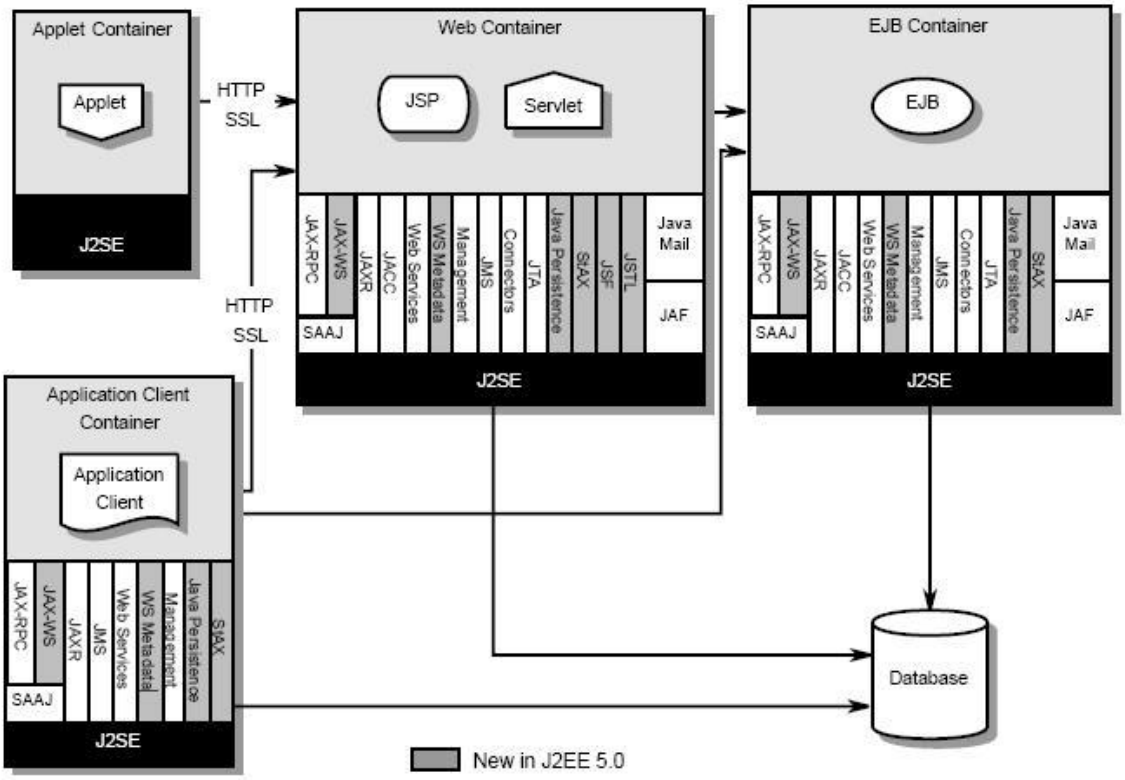

Figura 32 – JAVA EE – Diagrama da Arquitetura

Aprofundando no JAVA EE e observando o diagrama da arquitetura (Figura 63), nota–se que essa especificação foi criada baseada em outra especificação do JAVA, a Java SE (Java Standart Edition), ou seja, o padrão JAVA EE é uma forma de se utilizar o padrão JSE que contempla as APIs de funcionamento básicas do JAVA aplicado para a WEB.

4.3 Camadas e MVC

Para que esta aplicação fosse desenvolvida mantendo sustentação para novas funcionalidades e, ainda, com um código mais organizado e fácil de entender, a arquitetura aplicada no desenvolvimento desse sistema foi o MVC (*Model–view–controller*). De acordo com Macoratti (2003),

A arquitetura MVC (Modelo Visualização Controle) fornece uma maneira de

> dividir a funcionalidade envolvida na manutenção e apresentação dos dados de uma aplicação. A arquitetura MVC não é nova e foi originalmente desenvolvida para mapear as tarefas tradicionais de entrada, processamento e saída para o modelo de interação com o usuário. (MACORATTI, 2003)

Arquitetura em camadas é um padrão de arquitetura de software, que define os componentes da aplicação em camadas (Camada de Negócio, Apresentação e Controle). O conceito de MVC aplica–se às camadas, assim separadas, atribuindo responsabilidades a elas.

A arquitetura de camadas é utilizada em uma aplicação para manter os componentes separados por responsabilidade, diminuindo o acoplamento entre esses componentes, fazendo com que não haja impacto em mudanças decorrentes na aplicação. A figura 64 exemplifica um exemplo de uma estrutura sem separação lógica:

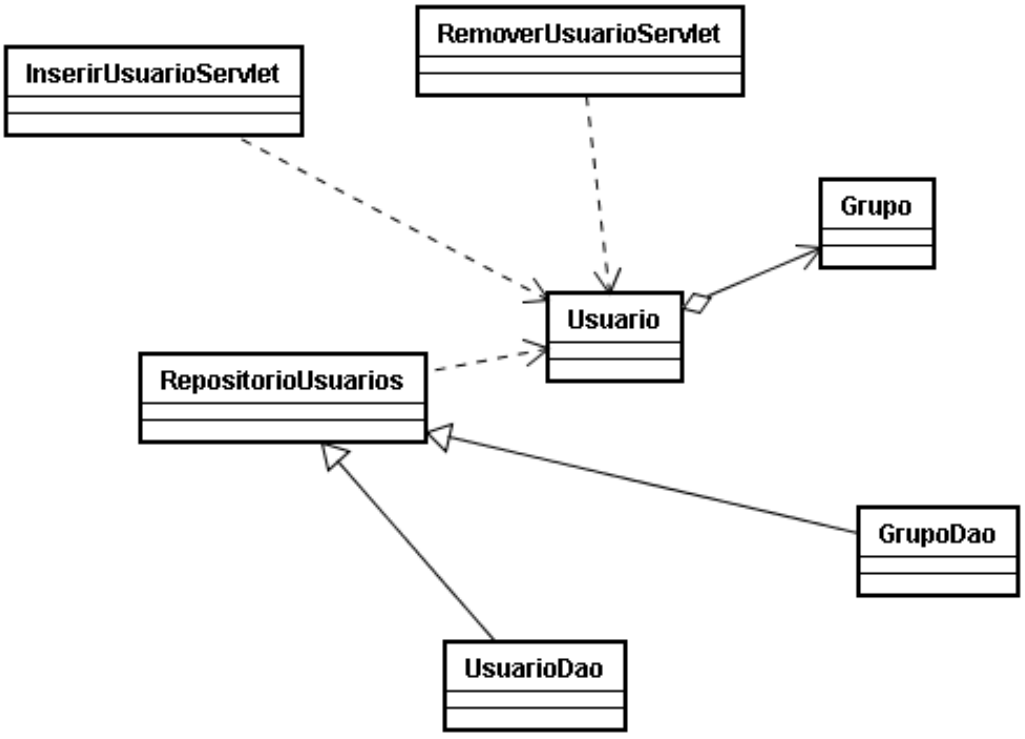

Figura 33 – Diagrama sem nenhuma separação lógica

A figura 65 mostra como ficaria uma estrutura com uma separação lógica definida.

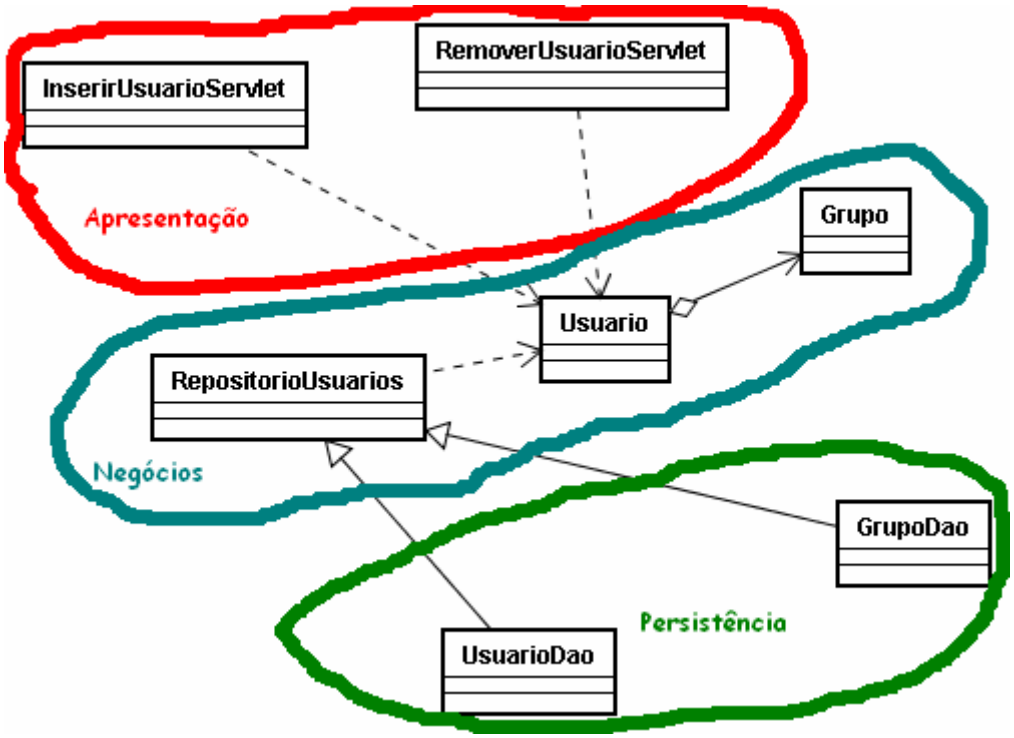

Figura 34 – Componentes agrupados

O padrão MVC divide a aplicação em 3 visões (Modem–View–Controller), no momento em que os componentes da nossa aplicação são organizados, o conceito de MVC é aplicado nesses agrupamentos, no qual a camada de negócios fica como o Model, a camada de apresentação como a View, e a camada de controle fica sendo “definida” na camada de apresentação, agindo como um meio de comunicação com o negócio.

4.4 STRUTS

Dentre os frameworks disponíveis para desenvolvimento de aplicações web, utilizando a linguagem Java, o que se destaca é o Struts. Este framework se aplica fortemente sob a arquitetura JAVA EE.

Struts é uma camada de controle flexível baseada na avançada tecnologia de Java Servlets, JavaBeans, ResourcesBundles e XML, assim como vários pacotes

Apache Commons, como BeanUtils e Chain of Responsibility. O framework ajuda a criar um ambiente de desenvolvimento extensível para sua aplicação, baseado em normas e padrões de projeto.

De acordo com a obra de Husted (2004), percebe–se que o uso desse padrão é fundamental para prevenir o aumento da complexidade da aplicação. Desta forma, os dados e a visão ficam separados de forma que não haja manipulação entre os dados dessas camadas, ou seja, as mudanças que precisarem ser feitas na parte de dados da aplicação podem ser feitas sem a preocupação de ocorrer alterações indevidas nas demais camadas, e vice–versa.

> Ao elevar as estruturas físicas, os engenheiros da construção usam suportes para fornecer sustentação para cada piso de um prédio. Do mesmo modo, os engenheiros do *software* usam o Struts para suportar cada camada de uma aplicação comercial. (HUSTED, 2004)

Utilizando o *Struts* no desenvolvimento da aplicação do Porto Belo *Palace* Hotel, a implementação do sistema foi simplificada com os recursos desse *framework*, pois utiliza, além de *servlets*, outros *design patterns* (*Front Controller, Command, Adapter*) que beneficiam ainda mais o desenvolvedor, tais como:

1. Redirecionar todas as requisições para um único ponto.
2. Extrair e validar informações contidas nas requisições.
3. Mapear as requisições para as suas respectivas atividades.
4. Redirecionar dados e conteúdo para suas respectivas páginas.
5. Tratar eventuais exceções.

O *Struts* é implementado como controle da aplicação, responsável por receber todas as requisições e passar para a camada de negócios, executar as regras e, devido a popularidade das aplicações *web*, requisitos foram adicionados para tornar simples esse mecanismo de *request* e *response*. Na figura 66, o diagrama mostra com clareza este fluxo:

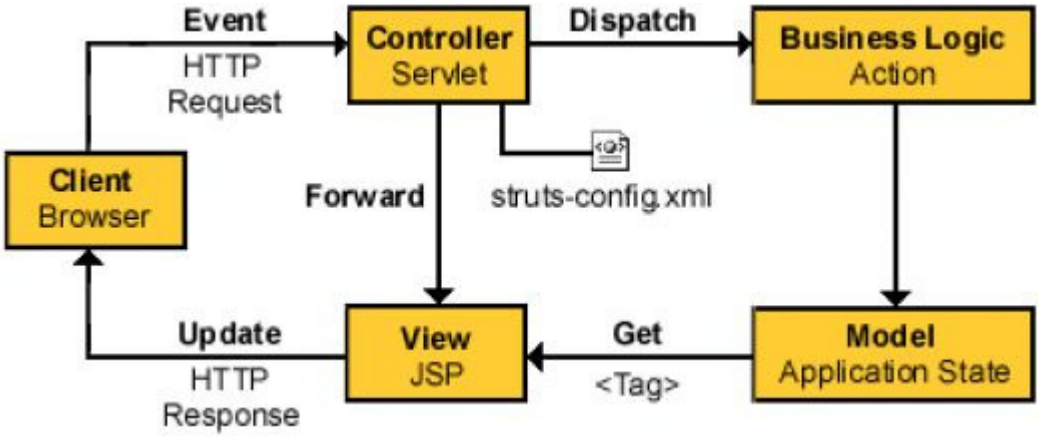

Figura 35 – Como o *Struts* implementa o MVC

A classe *ActionServlet* vai atuar como o *FrontController*, responsável por receber a requisição enviada pelo cliente (http://localhost:8080/projeto/acao.do). Uma leitura do web.xml (quadro 58) será feita, a fim de encontrar a ação específica e invocar a respectiva solicitação.

Para o desenvolvimento, a classe *ActionServlet* atua como *FrontController* do *Struts*, sendo necessário configurá–la no arquivo web.xml do projeto, conforme o quadro 58.

```
<servlet>
	<servlet-name>action</servlet-name>
	<servlet-class>org.apache.struts.action.ActionServlet</servlet-class>

	<!-- Struts Initiate Parameter -->
		<init-param>
		<param-name>config</param-name>
		<param-value>/WEB-INF/struts-config.xml</param-value>
	</init-param>

	<init-param>
		<param-name>chainConfig</param-name>
		<param-value>org/apache/struts/tiles/chain-config.xml</param-value>
	</init-param>

	<load-on-startup>1</load-on-startup>
</servlet>
```

```
<!-- Standard Action Servlet Mapping -->
<servlet-mapping>
       <servlet-name>action</servlet-name>
       <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

Quadro 27 – Arquivo web.xml

O *ActionServlet* funciona recebendo a requisição, tendo como responsabilidade também extrair os parâmetros para instanciar um *ActionForm*.

Para os *domains* (ou *beans*) utilizados no projeto do sistema do Porto Bello, é aplicado o uso do *ActionForm*, que trabalha juntamente com o *ActionServlet* para receber os valores dos respectivos atributos de determinado *domain*.

O termo *domain* significa domínio de uma entidade como, por exemplo, para uma tela de cadastro de cliente, na qual cada campo representa um atributo do mesmo, existe um domínio que irá conter os atributos de um cliente, que será o objeto do negócio. O uso do *ActionServlet* aplicado ao recebimento da requisição e do *ActionForm* aplicado ao *domain*, implica na validação dos atributos, ou seja, o *ActionServlet* irá receber a requisição contendo os valores informados no formulário, e irá extrair os parâmetros da requisição e popular um *domain* aplicado pelo *ActionForm*, formando, assim, o objeto cliente populado com as informações fornecidas na tela.

Para cada requisição feita pelo usuário, uma *Action* é executada, e o processo de criação do objeto por meio do *ActionForm* é executado caso solicitado e o objeto é montado e enviado para continuar o processo de acordo com as regras.

As *Actions* representam as ações que serão feitas na aplicação, ações como salvar, listar, excluir, etc, sendo elas invocadas através de *submits*, nos quais são informadas essas ações, como por exemplo:

- salvarApartamento.do
- listarHospedes.do
- excluirFuncionario.do

A ação de salvar apartamento é demonstrada no diagrama de sequência a

nível de projeto conforme mostra a figura:

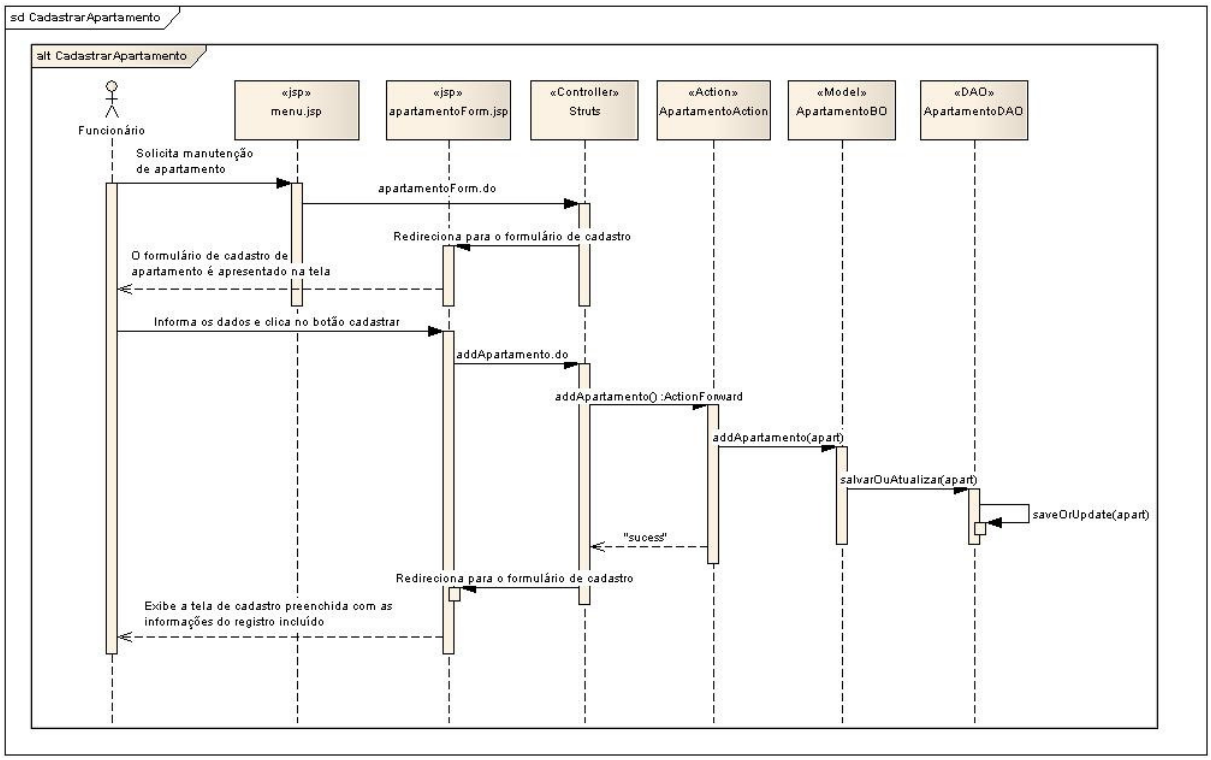

Figura – Diagrama de sequência cadastrar apartamento (projeto)

Após receber a solicitação (exemploAcao.do), o *ActionServlet* irá recuperar essa ação no arquivo de configurações. As informações vindas do cliente serão encaminhadas para a respectiva ação através de um *request* (ou como sendo um *FormBean*). A partir desta ação, o modelo do negócio será acessado e a informação será processada ou salva no banco. Para encerramento desta ação, será feito um *actionForward*, que representlaa o mecanismo (JSP, HTML) que irá apresentar o resultado para o cliente. (Quadro 59):

```
<action path="/jsp/addApartamento"
      type="bello.porto.action.ApartamentoAction"
      name="ApartamentoForm"
      scope="request"
      input="tiles.apartamento.form"
      validate="true"
      parameter="acao" >
      <forward name="cadastraApart" path="tiles.apartamento.form" />
</action>
```

Quadro 28 – Ação para adicionar apartamento

A classe que será a ação informada irá estender a classe *Action* do *Struts*, e possui o método execute (argumentos), que deverá ser sobrescrito (Quadro 60):

```
public ActionForward execute(ActionMapping mapping, ActionForm form,
HttpServletRequest request, HttpServletResponse response) throws Exception {
       Apartamento apart = new Apartamento();
       apart = (Apartamento) form;
       apartB.addApartamento(apart);
       return mapping.findForward("cadastraApart");
}
```

Quadro 29 – Método execute para a ação de cadastrar apartamento

A partir da (função) *Action*, o objeto recebido será manipulado pelas regras da aplicação e persistido no banco, se for o caso.

O mapeamento de todas essas *actions* ficam no arquivo denominado *struts–config.xml*. Dentro desse arquivo de configurações do *Struts* ficam, como principais, os:

1. ActionForms;
2. ActionForwards;
3. ActionMapping.

As requisições passam por esse xml para poder prosseguir com o fluxo. Ver figura 67:

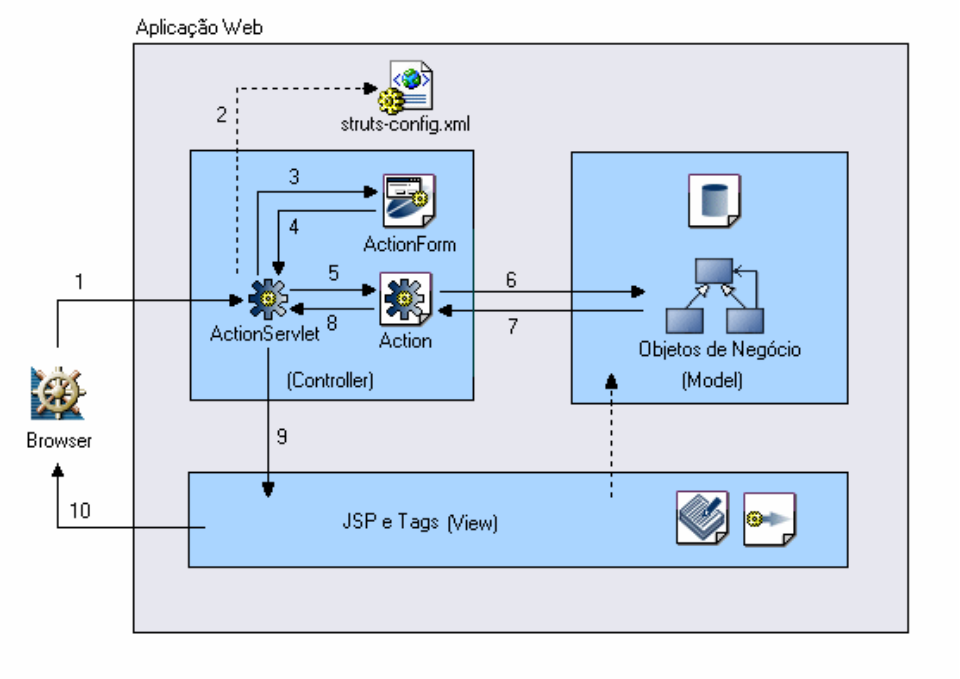

Figura 36 – Fluxo *Struts*

A figura 67 descreve desde o início da requisição, todos os passos, até o seu término.

A princípio, o usuário em seu navegador acessando uma tela de cadastro, seleciona a opção cadastro de apartamento, por exemplo, preenche os campos com as informações solicitadas, e faz um *submit*. Em seguida, o *controller* é acionado e invoca o *ActionServlet*, que chama o arquivo de configuração do *Struts* (*struts–config.xml*), que procura pela ação solicitada pelo usuário. Com isso, as configurações são montadas de acordo com a *action* que foi solicitada na requisição, neste momento o *Struts* já sabe qual classe vai ser acessada, qual *domain* será populado e devolve essas informações para o *ActionServlet* que vai executar. Com isso, o *domain* será populado e enviado para a classe da respectiva ação que terá o método execute que será acessado e executado. A partir desse momento, o objeto, já montado, irá passar pela camada de negócio da aplicação, tendo todas as regras executadas e persistidas no banco.

4.5 HIBERNATE

Ao interagir com um banco de dados relacional, o uso do Hibernate pode facilitar muito no que diz respeito à clareza e objetividade do código. Ao invés de criar trechos em SQL para manipular objetos, o Hibernate trata esses objetos como sendo suas respectivas tabelas, facilitando o manuseio da persistência da aplicação, assim como a manutenção.

> O Hibernate é um framework para o mapeamento objeto–relacional escrito na linguagem Java, mas também é disponível em .Net com o nome NHibernate. Este programa facilita o mapeamento dos atributos entre uma base tradicional de dados relacionais e o modelo objeto de uma aplicação, mediante o uso de arquivos (XML) para estabelecer esta relação. (http://pt.wikipedia.org/wiki/Hibernate)

Persistência é um termo raramente utilizado no contexto de dados, preferencialmente, o termo é banco de dados. A função do sistema na parte de gerenciamento de dados, ou camada de persistência, é permitir o acesso e a atualização simultâneos de dados persistidos, ou seja, a fim de garanti–los a um longo prazo.

Na aplicação do Porto Bello, foi usado na camada de persistência com o padrão de projeto Data Access *Object* (DAO), que se trata de um modelo para persistir dados em banco de dados relacionais. Esse padrão mantém separadas as regras do negócio do Porto Bello das regras de acesso ao banco de dados, no caso dessa aplicação que utiliza o MVC.

Na utilização desse padrão de projeto, foi usado o *Hibernate* como mecanismo de persistência. O uso do *Hibernate* permite persistir dados relacionais de maneira transparente, utilizando–se de arquivos xml (os HBM) para mapear os atributos dos nossos *domains* com os atributos das tabelas no banco de dados.

> Muitas tentativas foram feitas para interligar as tecnologias relacionais e orientadas para objeto, ou para substituir uma pela outra, mas o abismo entre ambas é um dos fatores intrincados da computação corporativa hoje em dia. É este desafio – fornecer uma ponte entre dados relacionais e objetos Java – que o hibernate assume através de sua abordagem do mapeamento objeto/relacional (ORM). O Hibernate enfrenta este desafio de uma maneira pragmática, direta e realista. (CHRISTIAN BAUER, 2005)

A simplicidade do uso do *hibernate* evita a perda de tempo escrevendo SQL e misturar com o código Java, fazendo com que a preocupação seja toda voltada

somente para os objetos do negócio, ou *domains*. Os passos utilizados na utilização do *Hibernate* no sistema do Porto Bello são:

- Criação das tabelas no banco, onde os dados irão persistir.
- Criação dos objetos de negócio, tais objetos que terão seu estado persistido nas suas respectivas tabelas.
- Criação dos arquivos HMB, utilizado para relacionar as propriedades do objeto aos campos da tabela.
- Criação do arquivo de configurações do *hibernate*, responsável por manter os dados da conexão e quais objetos estarão disponíveis no mapeamento.
- Criação da classe DAO que irá conter as propriedades para que o acesso ao banco seja realizado.

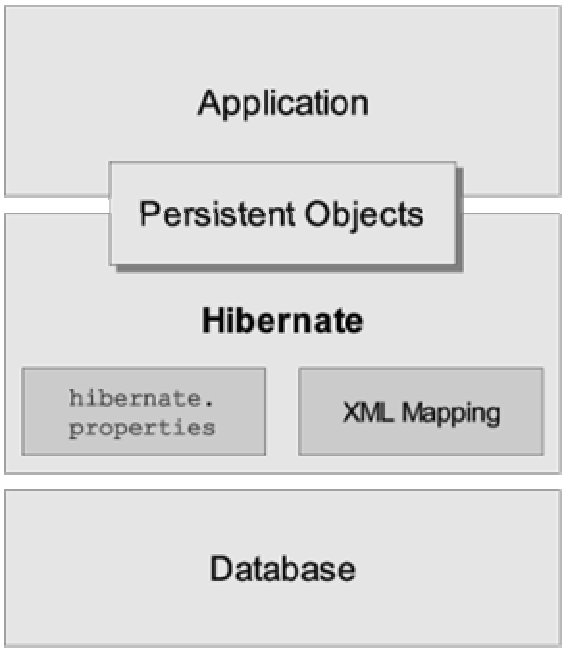

Figura 37 – Diagrama do fluxo do *Hibernate*

A utilização do *Hibernate* consiste em ter um objeto para cada tabela do banco, entretanto, haverá um arquivo que funcionará com mecanismo responsável pelo mapeamento das propriedades do objeto com os campos da tabela. Este arquivo é chamado de HBM (*SeuObjeto.hbm.xml*).

Esse arquivo de mapeamento contém *tags* que interpretam as propriedades do objeto, e define a configuração com base no tipo do dado do objeto e do campo

da tabela. Com todo esse arquivo montado, a persistência do objeto mapeado com sua respectiva tabela, a persistência desse objeto já fica automática, evitando que o resultado de uma busca ou a preparação de uma inserção fosse feita manualmente e sujeita a erros.

O objeto Apartamento possui seus atributos e também um arquivo HBM (*Apartamento.hbm.xml*). O mapeamento foi feito e adicionado nas configurações do *Hibernate* para estar disponível para uso.

O mapeamento HBM define todas as características de uma tabela, desde o ID que é a chave primária, até os relacionamentos existentes. Com esses arquivos criados, o DAO terá apenas que informar qual objeto irá ser persistido e invocar métodos já prontos para executar as ações (salvar, buscar, excluir, etc).

```
package bello.porto.dao;

import java.util.Iterator;
import java.util.List;

import org.hibernate.HibernateException;
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.Transaction;

import bello.porto.beans.Apartamento;

public class ApartamentoDAO {

       protected Session session;
       protected Transaction tx;
       private static List<Apartamento> apartamentos;

    public ApartamentoDAO() throws HibernateException {
        HibernateFactory.buildSessionFactory();
    }

    public Session getSession() {
        return session;
    }
```

```
public Transaction getTx() {
    return tx;
}

public void salvar(Apartamento obj){
    try {
         session = HibernateFactory.openSession();
        tx = session.beginTransaction();
        session.save(obj);
        tx.commit();
    } catch (HibernateException e){
         HibernateFactory.rollback(tx);
    } finally {
        HibernateFactory.close(session);
    }
}

@SuppressWarnings("unchecked")
   public Apartamento localizar(int id){

   Apartamento apart = null;
    Iterator iter = apartamentos.iterator();

    while(iter.hasNext()){
         apart = (Apartamento) iter.next();
         if(apart.getIdApartamentoNum() == id){
               break;
         }
    }
    return apart;
}

@SuppressWarnings("unchecked")
   public List localizarTodos(Class clazz){
   List objs = null;
    try {
         session = HibernateFactory.openSession();
        tx = session.beginTransaction();
        Query q = session.createQuery("from " + clazz.getName());
        objs = q.list();
        tx.commit();
    } catch (HibernateException e) {
         HibernateFactory.rollback(tx);
```

```
        } finally {
            HibernateFactory.close(session);
        }
        return objs;
    }

    @SuppressWarnings("unchecked")
       public void remover(Apartamento obj) {
        try {
             session = HibernateFactory.openSession();
            tx = session.beginTransaction();
            session.delete(obj);
            tx.commit();
        } catch (HibernateException e) {
             HibernateFactory.rollback(tx);
        } finally {
            HibernateFactory.close(session);
        }
    }

    public void salvarOuAtualizar(Apartamento obj) {
        try {
             session = HibernateFactory.openSession();
            tx = session.beginTransaction();
            session.saveOrUpdate(obj);
            tx.commit();
        } catch (HibernateException e) {
             HibernateFactory.rollback(tx);
        } finally {
            HibernateFactory.close(session);
        }
    }

    public void handleException(HibernateException e){
       HibernateFactory.rollback(tx);
       return;
    }
}
```

Quadro 30 – Classe DAO ApartamentoDAO.java

Os métodos da classe (Quadro 61), no caso da aplicação do Porto Bello, são

específicos para o objeto referente ao apartamento. Cada método recebe um objeto do tipo Apartamento e invoca os métodos do *Hibernate* para continuar com a persistência.

Esse código abre a sessão com o banco e inicia uma transação, onde irá ser feita a ação específica (Quadro 62). São eles:

```
session = HibernateFactory.openSession();
tx = session.beginTransaction();
```

Quadro 31 – Abre sessão com o banco de dados

Em seguida, é através desses métodos que o objeto é manipulado, que pode ser:

a) Salvar:

```
    session.save(obj);
```

Quadro 32 – Salva determinado objeto no banco

b) Remover:

```
    session.delete(obj);
```

Quadro 33 – Remove determinado objeto do banco

c) Atualizar (neste caso, salvar ou atualizar):

```
    session.saveOrUpdate(obj);
```

Quadro 34 – Salva ou atualiza determinado objeto no banco de dados

d) Somente atualizar:

```
    session.update(obj);
```

Quadro 35 – Apenas atualiza determinado objeto no banco de dados

Ou criando uma *query* que retorna todos os valores e gerando uma lista de objetos:

```
Query q = session.createQuery("from " + clazz.getName());
objs = q.list();
```

Quadro 36 – Retorna vários objetos do banco numa lista

Agora, para que esse mapeamento fique disponível para uso, devemos configurar o arquivo (Quadro 51) de configurações (hibernate.cfg.xml):

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
            "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
            "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property
name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
        <property name="hibernate.connection.password">lucas</property>
        <property
name="hibernate.connection.url">jdbc:mysql://localhost:3306/portobello</property>
        <property name="hibernate.connection.username">root</property>
        <property
name="hibernate.dialect">org.hibernate.dialect.DB2Dialect</property>

        <mapping resource="bello/porto/dao/Funcionario.hbm.xml" />
        <mapping resource="bello/porto/dao/Grupo.hbm.xml" />
        <mapping resource="bello/porto/dao/Hospede.hbm.xml" />
        <mapping resource="bello/porto/dao/Usuario.hbm.xml" />
        <mapping resource="bello/porto/dao/ReservaHospedagem.hbm.xml" />
        <mapping resource="bello/porto/dao/Apartamento.hbm.xml" />
        <mapping resource="bello/porto/dao/Pessoa.hbm.xml" />
        <mapping resource="bello/porto/dao/TipoApartamento.hbm.xml" />

    </session-factory>
</hibernate-configuration>
```

Quadro 37 – Arquivo de configurações hibernate.cfg.xml

Quando o mapeamento é criado, ele deve ser adicionado nesse arquivo para que possa ser usado na nossa camada de persistência por um DAO.

A aplicação se tornou mais fácil de desenvolver com o uso dessas tecnologias (*frameworks*), mantendo um código com lógica separada e simplicidade na codificação, resultando em uma manutenibilidade mais fácil.

5 PROJETO DE INTERFACES

5.1 Regra do Click

A *web* é o ambiente no qual o poder do cliente se manifesta no mais alto grau seguindo a regra do click: quem faz o click faz as regras. Na economia de rede, o *website* torna–se a principal interface da empresa com o cliente. Na verdade, para empresas de comércio eletrônico, o site é a empresa. Dr. Jakob Nielsen – importante engenheiro da *Sun Microsystems* (até 1998), guru de usabilidade na *Web* e detentor de 38 patentes nos Estados Unidos (principalmente de formas de facilitar o uso na internet) – argumenta que:

> [...] A interface com o usuário torna–se os materiais de marketing, a vitrine, o interior da loja, a equipe de vendas e o suporte pós–venda, tudo em um só pacote. Em muitos casos, o site torna–se até mesmo o produto em si. Portanto ter uma má usabilidade é como ter uma loja que fica no décimo–sétimo andar de um prédio (e, portanto, ninguém pode ir) e não tem nada além de vendedores mal–humorados que não falam com os clientes (e, portanto, as pessoas não compram muito) (NIELSEN, 2000).

Joel Spolsky, ex–programador e gerente na Microsoft, na Viacom e no Juno e fundador da *Fog Creek Software* (Nova York) confirma a teoria com o axioma de maior importância de todos os projetistas de interfaces com usuários: “uma interface de usuário é bem projetada quando o programa se comporta exatamente como o usuário pensa que ele se comportaria” (SPOLSKY, 2000). Geralmente os usuários julgam um sistema pela sua interface ao invés de sua funcionalidade, um projeto de interface mal elaborado pode levar o usuário a cometer erros catastróficos. Um projeto de interface pobre é uma das razões pela qual muitos sistemas de software nunca foram utilizados.

Um dos maiores desafios hoje é projetar interfaces efetivas para sistemas de software com o objetivo de ajudar o usuário a obter acesso rápido ao conteúdo de sistemas complexos. Uma *web* com padrões resolve estes problemas.

5.2 Toda história tem um começo

> [...] Onde devo começar, por favor vossa majestade? Comece do começo," disse bravo o rei, "e vá até chegar ao fim: então pare.[...] (Lewis Caroll – Alice no país das maravilhas)

5.2.1 O nascimento da Internet

No final de outubro de 1957 a União Soviética chocou o planeta e principalmente os Estados Unidos ao lançar com sucesso o primeiro satélite na órbita da Terra chamado *Sputnik 1*. Os Estados Unidos tinha seu próprio programa de lançamento de satélites, mas ainda não havia lançado. Este evento levou diretamente à criação da *ARPA* (Agência de Projetos de Pesquisa Avançada) do Departamento de Defesa dos Estados Unidos.

Em 1960, o psicólogo e cientista de computação Joseph Licklider publicou um documento intitulado "Relação Homem–Computador", que articulava a idéia de computadores em rede fornecendo armazenamento e consulta de informações. Em 1962, ao mesmo tempo em que trabalhava para a *ARPA* como o chefe do escritório de processamento de informação, ele formou um grupo para futuramente fazer pesquisas com computadores, mas deixou o grupo antes que qualquer trabalho sobre a idéia tivesse sido feito. O plano para esta rede de computadores (chamada "*ARPANET*") foi apresentado em outubro de 1967, e em dezembro de 1969 a primeira rede de quatro computadores estava pronta e funcionando. O grande problema em criar uma rede era como conectar redes físicas separadas sem que as ligações aumentassem os recursos de rede para *links* constantes. A técnica que solucionou este problema é conhecida como troca de pacotes e envolve requisições de dados sendo divididos em pequenos pedaços ("pacotes"), que podem ser processados rapidamente sem bloquear a comunicação de outras partes – esta técnica é usada para o funcionamento da Internet hoje. Este conceito recebeu grande adoção, com o surgimento de várias outras redes usando a mesma técnica de troca de pacotes.

A proliferação de diferentes protocolos de rede logo se tornou um problema,

quando se tentava fazer todas as redes separadas se comunicarem. Robert Kahn, enquanto trabalhava em um projeto de pacotes de rede de satélite para a *ARPA*, começou a definir algumas regras para uma arquitetura de rede mais aberta para substituir o protocolo atual usado na ARPANET. Mais tarde, com a chegada de Vinton Cerf da Universidade de *Stanford*, os dois criaram um sistema que mascara a diferença entre os protocolos de rede usando um novo padrão. Na publicação do rascunho da especificação em dezembro de 1974, eles o chamaram de *ITCP – Internet Transmission Control Program* (Programa de Controle de Transmissão Entreredes).

Esta especificação reduziu o papel da rede e transferiu a responsabilidade de manter a integridade da transmissão para o computador servidor possibilitando o acesso com facilidade a quase todas as redes simultaneamente. A *ARPA* financiou o desenvolvimento do software e em 1977 foi conduzida uma demonstração de uma comunicação entre três redes diferentes. Em 1981, a especificação foi finalizada, publicada e adotada e em 1982 as conexões da *ARPANET* fora dos EUA foram convertidas para usar um novo protocolo chamado TCP/IP. Assim nasceu a Internet.

5.2.2 A criação da World Wide Web

No início dos anos 90 surgiu um sistema (criado pela Universidade de Minnesota) de recuperação de informação chamado *Gopher* que oferecia um método de entrega de menus de *links* para arquivos, recursos de computadores e outros menus. Estes menus puderam atravessar as fronteiras da computação da época e usar a Internet para obter menus de outros sistemas. Eles foram muito populares com universidades que buscavam oferecer informação dentro do campus e grandes empresas procurando centralizar o armazenamento e gerenciamento de documentos. No início do ano de 1993 muitas empresas começaram a buscar alternativas ao *Gopher* devido início de cobrança de taxas de licença pelo uso de sua referência de implementação do servidor *Gopher*.

A CERN (Organização Europeia para Investigação Nuclear), na Suiça, tinha uma alternativa através de Tim Berners–Lee que estava trabalhando em um sistema

de gerenciamento de informação, no qual um texto poderia conter *links* e referências para outros trabalhos, permitindo o leitor a pular rapidamente de um documento para outro. Ele havia criado um servidor para publicar este tipo de documento (chamado hipertexto) bem como um programa para lê–lo, que ele havia chamado de "*WorldWideWeb*". Este software foi lançado em 1991, entretanto, ele teve dois eventos que causaram sua explosão em popularidade e a eventual substituição do *Gopher*.

Em 20 de Abril de 1993 o *CERN* lançou o código–fonte do *WorldWideWeb* em domínio público, então qualquer um poderia usar ou construir algo sobre o *software* sem nenhuma taxa.

Então, mais tarde no mesmo ano, o *NCSA* (Centro Nacional de Aplicações de Supercomputação) lançou um programa que combinava um navegador *web* e um cliente *Gopher*, chamado *Mosaic*. Ele estava disponível originalmente apenas para máquinas *Unix* e em forma de código–fonte, mas em dezembro de 1993 saiu uma nova versão do *Mosaic* que podia ser instalada em *Apple Macintosh* e *Microsoft Windows*. O *Mosaic* viu sua popularidade aumentar rapidamente, e junto com ele a *Web*. Também o número de navegadores disponíveis aumentou drasticamente, muitos criados por projetos de pesquisa em universidades e empresas.

5.2.3 A "Guerra dos Browsers"

A popularização da *web* trouxe interesses comerciais. Marc Andreessen deixou a *NCSA* e junto com Jim Clark fundou a *Mosaic Communications*, depois renomeada para *Netscape Communications Corporation*, e começou a trabalhar no que viria a ser o *Netscape Navigator*. A versão 1.0 do software foi lançada em dezembro de 1994. Neste mesmo ano a *Telenor* (uma empresa de comunicações Noroeguesa) criou a primeira versão do navegador *Opera*. A Spyglass Inc. (o braço comercial da *NCSA*) licenciou sua tecnologia do *Mosaic* para a *Microsoft* formar a base do *Internet Explorer*. A versão 1.0 foi lançada em agosto de 1995. Em uma rápida escalada, a *Netscape* e a *Microsoft* tentaram cada qual obter uma margem competitiva em termos de recursos suportados, a fim de atraírem desenvolvedores. Isto ficou reconhecida como a Guerra dos *Browsers*.

5.2.4 A web sem padrões

No período da guerra dos navegadores, *Microsoft* e *Netscape* estiveram focadas em implementar novas funções em vez de consertar os problemas com as funções já suportadas, e adicionaram funções proprietárias e criaram funções que competiam diretamente com funções existentes no outro navegador, mas implementadas de uma forma incompatível. Como conseqüência os desenvolvedores foram forçados a lidar com o crescente aumento do nível de confusão enquanto tentavam criar *web sites*, as vezes chegando ao ponto de construir dois sites diferentes para os navegadores principais, e outras vezes escolhendo suportar apenas um navegador, e bloqueando os outros de usarem seus sites.

Com o crescimento do comércio na *Web* (*e–comerce*) veio a necessidade de publicação de conteúdo diagramado, como jornais, revistas, catálogos e assim por diante, a linguagem *HTML* (por não possui estes recursos) foi adaptada, infelizmente nesta fase foi esquecido a qualidade do código e somente houve preocupação unicamente visual.

Por volta de 1996 surgiram editores *HTML WYSIWYG* como *Go Live, Front Page e Dreamweaver*. Editores *HTML WYSIWYG* (do inglês "*What You See Is What You Get*" ou em português “O que você vê é o que você tem") são programas que permitem ter a visualização final de um documento, enquanto o mesmo é editado. A edição do arquivo pode ser feito por código ou por *design*. No modo *design*, uma pessoa com pouco experiência (ou nenhuma) em programação pode criar uma página ou mesmo um site. Entretanto o código gerado pelo editor visual é *blackcode* (código sujo) com *scripts* complexos (várias linhas de código) para funções simples de pouca codificação.

No mercado apareceram vários cursos e profissionais desatualizados e sem experiência, como conseqüência as páginas geradas eram um incompreensível emaranhado de *tags* com tabelas, formatações, scripts e assim por diante e tudo bem maior que o necessário. E o problema aumentou quando a codificação da apresentação da página foi desenvolvida com tabelas com bordas ocultas (*border* = 0), imagens invisíveis (*spacers.gif*) e misturado o código da apresentação com o código da estrutura. Na necessidade de alteração no site era preciso alterar todos os

arquivos, isto um a um. O resultado de tudo isso foi uma *web* sem padrões.

5.2.5 A formação da W3C

Em 1994, Tim Berners–Lee fundou o *World Wide Web Consortium* (W3C), no Instituto de Tecnologia de *Massachusetts*, com suporte do *CERN*, *DARPA* (como foi renomeada a *ARPA*) e da Comissão Europeia. A visão da *W3C* era a de padronizar os protocolos e tecnologias usados para criar a web de modo que o conteúdo possa ser acessado largamente pela população mundial tanto quanto o possível. Nos anos seguintes o *W3C* publicou várias especificações (chamadas "recomendações") incluindo o *HTML*, o formato de imagens *PNG*, e as Folhas de Estilo em Cascata (*CSS* 1 *e CSS* 2).

Apesar dos principais *browsers* terem sido envolvidos na criação de padões *web* desde a formação do *W3C*, durante muitos anos o seguimento dessas regras não era obrigatório, os fabricantes adotavam os documentos da *W3C* apenas se eles quizessem etiquetar que seus produtos como complacentes com a *W3C*. Ao lançar *browsers* que não suportavam os *standards* uniformemente, os fabricantes fragmentaram desnecessariamente a *web*, prejudicando de igual forma *designers*, programadores, utilizadores e empresas.

A falta de suporte uniforme para os *standards* do *W3C* deixou os consumidores frustrados: se usassem o *browser* "errado", muitos não conseguiriam ter acesso ao conteúdo ou executar as transações pretendidas. Entre aqueles mais frequentemente afetados estavam as pessoas com incapacidades ou necessidades especiais.

Na época as recomendações *W3C* não tinham muito valor no mercado devido a falta de conhecimento, importância e interesse dos usuários e programadores, em conseqüência disto, a "Guerra dos *Browsers*" continuou inabalável.

5.2.6 A formação da WaSP

Um grupo de desenvolvedores *web* e *web designers* profissionais se uniu e auto–denominou como Projeto Padrões *Web* ou *Web Standards Project* (WaSP). A

idéia era mudar o nome dos documentos da *W3C* de recomendações para padrões e assim poder *Microsoft* e a *Netscape* a suportá–los.

Em 2000, a *Microsoft* lançou o *Internet Explorer 5* para *Macintosh*. Este foi um marco muito importante, ele se tornou o navegador padrão instalado no *MacOS* nesse momento, e teve um razoável suporte às recomendações da *W3C* também. Juntamente com o suporte decente do *Opera* para *CSS* e *HTML*, isto contribuiu para um momento geral positivo, em que desenvolvedores e *web designers* se sentiram confortáveis projetando sites usando padrões *web* pela primeira vez.

A *WaSP* persuadiu a *Netscape* a adiar o lançamento da versão 5.0 do *Netscape Nativagor* até que ele estivesse mais complacente com os padrões (este trabalho formou a base do que é hoje o *Firefox*, um navegador muito popular). A *WaSP* também criou uma "força–tarefa para o *Dreamweaver*", para encorajar a *Macromedia* a mudar sua popular ferramenta de desenvolvimento para estimular e suportar a criação de sites complacentes.

5.2.7 Web Standard

No evento Campus *Party* 2009, Eduardo Santos em sua palestra sobre *Web Standard* argumentou:

> Web Standards ou padrões Web como um conjunto de normas, diretrizes, recomendações, notas, artigos, tutoriais e afins de caráter técnico, produzidos pela W3C. É destinado a orientar fabricantes, desenvolvedores, projetistas para o uso de práticas que possibilitem a criação de uma Web acessível a todos. [...] na comunidade de desenvolvedores web profissionais os padrões web se tornaram obrigatórios" (SANTOS, E. 2009).

A Figura 38 mostra um paralelo entre um site com e sem padrões web.

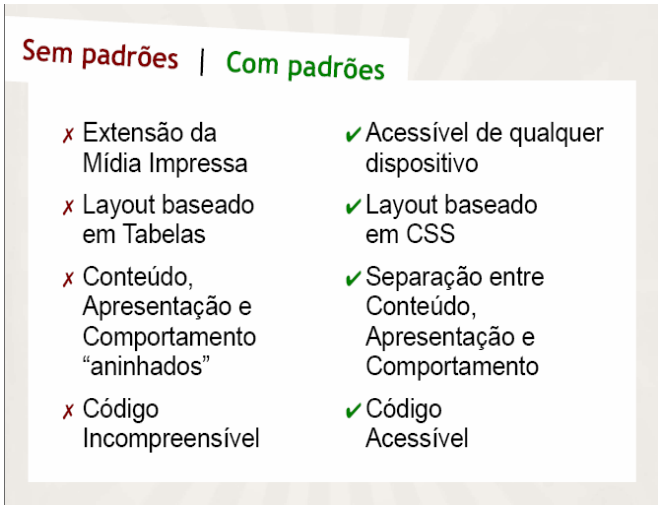

Figura 38 – Sem padrões x Com padrões.
(Fonte: http://www.agni.art.br/palestras/WebStandards.pdf)

Construir sites sem os padrões *W3C* as páginas precisam de extensão da mídia impressa, seus *layouts* são baseados em tabelas, na codificação o conteúdo, apresentação e comportamento são misturados resultando um código difícil de compreender. Enquanto que com padrões as páginas são acessíveis de qualquer dispositivo, seus *layouts* são baseados em *CSS*, há separação na codificação de conteúdo, apresentação e comportamento resultando um código acessível e de fácil compreensão.

Para ter um projeto de sucesso com os padrões *web* é necessário uma mudança de conceito dos *designers* e dos desenvolvedores. A Figura 33 indica como os padrões *web* resolvem os problemas discutidos.

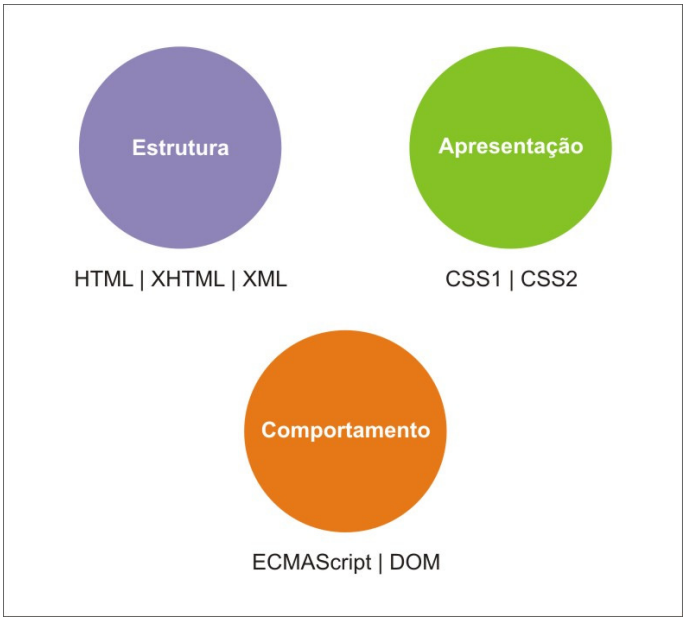

Figura 39 – Estilo de interação por linguagem de comando

A técnica web *standard* é dividir qualquer página *web* em três componentes separados: estrutura, apresentação e comportamento. As páginas devem ser desenvolvidas de forma independente, porém uma complementando a outra.

5.3 Componente: Estrutura

A funcionalidade da *web* é baseada em três padrões:

- URI, um sistema que especifica como cada página de informação recebe um "endereço" único onde pode ser encontrada.
- Protocolo de Transferência de *Hipertexto* (*HTTP*), um protocolo que especifica como o navegador e servidor web comunicam entre si.
- Linguagem de Marcação de *Hipertexto* (*HTML*), uma linguagem de marcação para codificar a informação de modo que possa ser exibida em uma grande quantidade de dispositivos.

A linguagem de marcação que é uma das funcionalidades da *web* é também um dos componentes *W3C* da divisão da *web*. Linguagem de marcação é também um conjunto de códigos aplicados a um texto ou a dados, com o fim de adicionar informações particulares sobre esse texto ou dado, ou sobre trechos específicos.

5.3.1 HTML

A *HTML* é uma linguagem de marcação de texto. A *W3SCHOOLS*, uma empresa de treinamento e certificações *web* define HTML como:

> HTML é uma linguagem para descrever as páginas web. HTML significa Hyper Text Markup Language. HTML não é uma linguagem de programação, é uma linguagem de marcação. Uma linguagem de marcação é um conjunto de tags de marcação. HTML usa etiquetas de marcação para descrever as páginas da web. (Tradução nossa).

De acordo com os padrões da *W3C* o código *HTML* deve ser semântico. Cada marcação é utilizada para o que ela realmente foi criada um código semântico está sendo construído. Semântica refere ao estudo do significado, ao codificar é preciso pensar na estrutura da Informação e fazer uma marcação com significado, por exemplo:

- Usar `<table></table>` apenas para dados tabulares e não para *layout* de páginas;
- Usar `<h1></h1>` até `<h6></h6>` para títulos;
- Usar `<ol></ol>` para listas ordenadas e `<ul></ul>` para listas não ordenadas, seguidos do elemento `<li></li>`;
- Usar `<em></em>` para ênfase;
- Usar `<strong></strong>` para enfase forte;
- Usar `<label></label>` para identificar campos em formulários;
- Assim por diante.

5.3.2 XML

A *XML* é uma linguagem de marcação de dados. A *W3SCHOOLS* (2009) define XML como:

> XML significa Extensible Markup Language. O XML é uma linguagem de marcação muito semelhante ao HTML. O XML foi projetado para transportar dados, para não exibir dados. Tags XML não são predefinidas. Você deve definir suas próprias marcas. O XML é projetado para ser auto–descritivo. XML é uma recomendação do W3C. (Tradução nossa).

A *XML* é simplesmente um texto e não tem nada de especial. É apenas texto simples e o *software* que pode lidar com texto puro também pode manipular XML. Não é um substituta para o *HTML* elas foram projetadas com diferentes objetivos: A *XML* para o transporte e armazenamento de dados e informações e a *HTML* para exibir dados e informações.

Com *XML* você inventar suas próprias *tags* (*tags* personalizadas) como por exemplo a *tag* recado que poderia ser utilizado em um sistema de recados:

```
<recado>
   <de>João</de>
   <para>Maria</para>
   <texto>Você quer sair comigo?</texto>
</recado>
```

Quadro 38 – *Tag* recado.

As *tags* no exemplo acima (como `<de>` e `<para>`) não são definidos em qualquer padrão *XML*. Essas marcas são "inventadas" pelo desenvolvedor do documento *XML*. Isso é possível porque a linguagem *XML* não possui *tags* pré–definidas.

A *XML* é agora tão importante para a *Web* como *HTML* foi à fundação da *web*, tornou–se uma recomendação do *W3C* e está em toda parte, é a ferramenta mais comum para as transmissões de dados entre todos os tipos de aplicações, e está se tornando cada vez mais popular na área de armazenamento e descrever a informação.

5.3.3 XHTML

Uma linguagem de marcação amplamente usada para texto é a *HTML*, mas que vem perdendo espaço para a sua evolução, o *XHTML* por conta desta ser mais

eficiente para separação entre a estrutura e o conteúdo de uma página de forma mais organizada e eficiente e é compatível com *HTML* 4.01 e todos os *browsers* suportam. Em *XHTML* 26 de janeiro de 2000 tornou–se uma recomendação W3C

Segundo a *W3SCHOOLS* (2009), *XHTML* significa:

> XHTML significa Extensible HyperText Markup Language. XHTML é quase idêntico ao do HTML 4,01. XHTML é uma versão mais rigorosas e de limpeza de HTML. XHTML é HTML definido como uma aplicação XML. XHTML é uma recomendação do W3C (Tradução nossa).

Quando criada corretamente (sem erros, sem *tags* ilegais e sem atributos) a marcação *XHTML* é completamente portátil. Ele funciona em navegadores *web*, leitoras de tela, navegadores de texto, dispositivos sem fio, entre outros.

5.3.4 JSP

Segundo Deitel (2005), *JavaServer Pages* (JSP) é:

> [...] uma extensão da tecnologia de servlet que separa a apresentação da lógica do negócio. Isso permite que os programadores em Java e designers de site Web concentrem suas formas – em escrever código Java e desenhar páginas Web, respectivamente [...].

Por ser baseada na linguagem de programação Java, a *JSP* tem a vantagem da portabilidade de plataforma, que permite a sua execução em diversos sistemas operacionais, como o *Windows* e o *linux*. Esta tecnologia permite ao desenvolvedor *web* produzir aplicações que acessem o banco de dados e manipular arquivos no formato texto, capturar informações a partir de formulários e captar informações sobre o visitante e sobre o servidor.

A *JSP* é similar às tecnologias *Ative Server Pages* (ASP) da *Microsoft* e *PHP* e permite a mistura de *HTML* estático com conteúdo gerado dinamicamente. Esses comandos dinâmicos são processados pelo servidor *web* antes da página *HTML* ser enviada. No lugar do comando é enviado apenas o resultado deste comando no formato *HTML*. Em resumo *JSP* é uma página *HTML* comum que contém código

Java.

Uma página *JSP* contém dois tipos de elementos:

- Elementos *template* que são a parte estática de uma página *web* que pode ser escrito em *HTML*, *XML* ou *XHTML*. Em linhas gerais é tudo que existe na página e que não é elemento JSP.
- Elementos *JSP* são os diretivas, scripts, ações padrão e ações personalizadas (*tag extension*).

Através destes componentes–chave a *JSP* auxilia não somente na estrutura da página *web*, mas também na apresentação e no comportamento.

5.3.4.1. Diretivas JSP

De acordo com Deitel (2005) diretivas são:

> [...] mensagens para o contêiner de JSP – o componente de servidor que executa JSPs – que permitem ao programador especificar configurações de página, incluindo conteúdo de outros recursos, e especificar bibliotecas de tag personalizada para utilização em uma JSP [...].

As diretivas são limitas pela por `<%@` e `%>`, são processadas em tempo de tradução e não produzem nenhuma saída imediata porque são processadas antes de a *JSP* aceitar qualquer solicitação.

5.3.4.2. Elementos Scripts JSP

Segundo Deitel (2005), os elementos scripts:

> [...] permitem aos programadores inserir código Java que interaja com componentes em um JSP (e possivelmente outros componentes de aplicativo Web) para realizar o processamento de solicitação. Scriptlets, um tipo de elemento script, contém fragmentos de código que descrevem a ação a ser realizada em resposta a uma solicitação de usuário. [...].

Os *scriptlets* são blocos de códigos delimitados por `<%` e `%>` e contém instruções Java que o contêiner coloca no método `_jspService` em tempo de tradução.

5.3.4.3. Tag Libraries

Segundo DEITEL (2005) as ações:

> [...] encapsulam funcionalidades em tags predefinidas que programadores podem incorporar em uma JSP. As ações freqüentemente são realizadas com base nas informações enviadas para o servidor como parte de uma solicitação particular de cliente. Elas também podem criar objetos Java para utilização em scriplets de JSP [...].

De acordo com Deitel a *JSP* possibilita embutir *scriplets* e *JavaBeans* em linha com conteúdo HTML, então porque precisamos de *Tags Libraries*? Segundo SHACHOR (2002) precisamos porque:

> [...] as tags nunca foram destinadas a oferecer mais funcionalidades do que scriptlets, apenas uma melhor embalagem. As tags JSP formam criadas para melhorar a separação de lógica de apresentação e lógica do programa, especificamente para abstrair sintaxe Java de HTML [...]

Os contêineres de *JSP* processam as *tags* em tempo de execução. Estas *tags* são de dois tipos: padrão e personalizadas.

5.3.4.4. Tag Padrão

As ações padrão facilitam a utilização da *JPS*, pois o código das páginas fica centralizado em marcadores *(tags*) e o código Java fica separado do código existente na página (elemento *template*).

Essas ações fornecem implementadores de *JSP* com acesso às tarefas mais comuns realizadas em um *JSP*, como incluir o conteúdo de outros recursos, encaminhar solicitações para outros recursos e interagir com componentes de

software *JavaBean*. As *tags* padrão são delimitadas por `<jsp:ação>` e `</jsp:ação>`. O Quadro 28 exemplifica algumas *tags* padrão e suas funções.

Tag Padrão	Função
<jsp:include>	Inclui dinamicamente outro recurso em um JSP.
<jsp:forward>	Encaminha processamento de solicitação para outro JSP, servlet ou página estática.
<jsp:useBean>	Especifica que o JSP utiliza um objeto (instância) de JavaBean seu escopo e ID para sua manipulação.
<jsp:setProperty>	Configura uma propriedade na instância JavaBean específica.
<jsp:getProperty>	Obtém uma propriedade na instância de JavaBean específica e converte o resultado em uma string para saída na resposta.

Quadro 39 – Exemplo de *Tag* padrão da JSP e suas funções**.** FONTE: adaptada de DEITEL (2005).

5.3.4.5. *Tags* Personalizadas

O conjunto de ações padrão disponível pela especificação *JSP* é insuficiente, mesmo para tarefas pequenas. Então a especificação proveu a criação de novas *tags* possibilitando associar uma série de funcionalidades ás páginas *JSP*. Estas novas *tags* são agrupadas e conhecidas como *Tag Libraries*.

5.3.4.6. Evolução da estrutura de página web com JSP

De acordo com ANGOTI (2009) houve uma “evolução da estrutura de página web e está dividida em três gerações:

Na primeira geração as páginas *JSP* contém *tags HTML* embutidas. Com este método técnica *web standard* é dividir qualquer página *web* em três componentes

separados é contrariada, porque não há separação entre estrutura e apresentação.

Segunda geração: páginas *JSP* contém diretivas `include` estáticas ou `include` *JSP* dinâmico. O reuso de código é utilizado estrutura e limitado na apresentação, pois cada página *JSP* ainda contém informações sobre apresentação.

Terceira geração: Há total separação entre estrutura e apresentação, ambos páginas *JSP* e *layouts* são reusáveis.

5.3.5 Ferramentas

5.3.5.1. Adobe *Dreamweaver*

Macromedia Dreamweaver é um editor de texto para projeto, codificação, páginas e aplicações web. Seu ambiente de edição desfruta do controle da codificação manual ou trabalho visual. O *Dreamweaver* fornece ferramentas úteis para melhorar a sua experiência de criação web.

Os recursos de edição visual no *Dreamweaver* permitem criar rapidamente páginas sem escrever uma linha de código e ver todos os elementos do site e arrastá–los do painel diretamente em um documento. É possível otimizar o fluxo de trabalho de desenvolvimento, criando e editando imagens em um aplicativo gráfico, então importá–los diretamente para o *Dreamweaver*, ou adicionando o *Macromedia Flash* objetos.

Segundo o *Dreamweaver Help*, o aplicativo

> [...] fornece um ambiente completo de recursos de codificação que inclui código de ferramentas de edição (tais como a coloração de código e marca conclusão) e material de referência sobre a linguagem Cascading Style Sheets (CSS), JavaScript e ColdFusion Markup Language (CFML), entre outros. (Tradução nossa).

5.3.5.2. Eclipse

Eclipse é um ambiente multi–linguagem de desenvolvimento de software que inclui um *IDE* e um *plugin* do sistema para estendê–lo. É escrito em Java e pode ser usado para desenvolver aplicações em Java e, por meio de diversos *plugins*, em outras línguas, bem como, incluindo *C*, *C++, COBOL*, *Python*, *Perl*, *PHP* e outros. Os usuários podem estender as suas capacidades através da instalação de *plugins* escritos para a estrutura de software *Eclipse*, como *kits* de ferramentas de desenvolvimento para outras linguagens de programação, e podem escrever e contribuir com seu próprio *plugin modules*. Os pacotes de idiomas fornecer traduções em mais de uma dezena de línguas naturais. O Software disponibilizado sob os termos da *Eclipse Public License* e é livre e aberto.

De acordo com a *Wiki Eclipse* (2004) o Eclipse

> [...] não surgiu do nada, mas evoluíram a partir de uma linha de produtos longos de ambientes de desenvolvimento, dos quais os anteriores são a IBM VisualAge para Smalltalk e VisualAge IBM Java. Ambos os produtos foram escritos em Smalltalk. A IBM VisualAge Micro Edition produto foi a primeira experiência séria e realmente muito bem–sucedida com a escrita todo o IDE em Java [...]. No entanto, por conta de terceiros, foi difícil estender o produto com componentes novos, principalmente por duas razões: (1) não foi projetado com um modelo de componentes em mente, e (2), foi essencialmente um monolítico, fechado, produto de origem.

O projeto *Eclipse* foi iniciado na *IBM* que desenvolveu a primeira versão do produto e doou–o como software livre para a comunidade. O gasto inicial da *IBM* no produto foi de mais de 40 milhões de dólares.

O *Eclipse Web Tools Platform* (WTP) projeto amplia a plataforma *Eclipse* com ferramentas para desenvolvimento de aplicações *web* e Java EE. Inclui fonte e editores gráficos para uma variedade de línguas, assistentes e aplicações integradas para simplificar o desenvolvimento e as ferramentas e *APIs* para apoiar a implantação, execução e aplicações de testes.

5.4 Componente: Apresentação

5.4.1 Interface

O termo *interface* é aplicado normalmente àquilo que interliga dois sistemas.

> Uma interface humano–computador é uma parte do sistema computacional, composta por uma coleção de dispositivos, por meio dos quais o usuário pode trocar informações com o sistema. Esses dispositivos são sensíveis às ações do usuário e são capazes de estimular a sua percepção para que ele possa avaliar e controlar o funcionamento do sistema (LEITE, 1998).

A *interface* possui componentes de *hardware* e *software*. Os componentes de *hardware* compreendem os dispositivos com os quais o usuário realiza as atividades motoras e perceptivas. Entre eles estão a tela, o teclado, o mouse e vários outros.

> Os componentes de software são responsáveis por controlar os dispositivos de hardware, construir os dispositivos virtuais, com os quais o usuário também pode interagir, gerar os diversos símbolos e mensagens que representam as informações do sistema e interpretar os comandos do usuário (SOUZA ET AL., 1999).

Na Figura 34, observa–se que a *interface* é a parte do sistema responsável por traduzir ações do usuário em ativações das funcionalidades (aplicação) e pela apresentação dos resultados produzidos.

> A interação é um processo de comunicação entre o usuário e o sistema, que engloba as ações do usuário sobre a interface e as suas interpretações sobre as respostas reveladas por essa interface (LEITE, 1998).

Portanto, a *interface* é o combinado de *hardware* e *software* que viabiliza a interação.

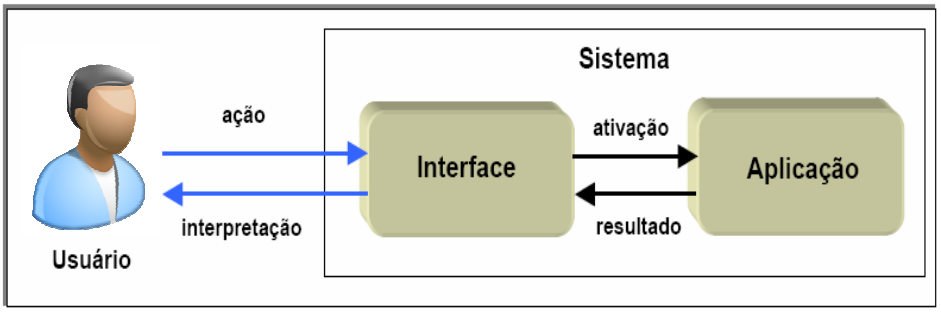

Figura 40 – Interação usuário–sistema.

Segundo SPOLSKY (2009), IU (Interface de Usuário) é importante porque afeta os sentimentos, as emoções, e o humor de seus usuários. Se a IU está

incorreta e o usuário sente como se eles não pudessem controlar o seu software, eles literalmente não serão felizes e eles amaldiçoarão isto no seu software. Se o IU é esperto e as coisas funcionam da maneira que o usuário espera que elas funcionem, eles estarão felizes como se eles controlassem o sucesso de pequenos objetivos.

Conforme relatado anteriormente sobre o conceito de Spolsky em relação ao axioma de maior importância de todos os projetistas de interfaces com usuários: Uma interface de usuário é bem projetada quando o programa se comporta exatamente como o usuário pensa que ele se comportaria. Steve Jobs, *Apple CEO*, reforça o conceito afirmando que "*Design* não é o que um produto parece ser, *design* é como o produto funciona".

Sobre a apresentação de um produto concebido por uma atividade de *Design*, Leite (1998) afirma:

> Design é a atividade intelectual de conceber e descrever um produto a partir dos requisitos de seus potenciais usuários. Esta atividade requer técnicas e ferramentas adequadas, aliadas à criatividade, ao talento e à experiência do designer. O produto concebido em uma atividade de design precisa ser apresentado através de um protótipo e/ou de uma especificação. A prototipação consiste na descrição do que foi concebido, utilizando materiais mais baratos e dimensões reduzidas. O objetivo é poder fazer uma avaliação. A especificação consiste na descrição abstrata, rigorosa, idealmente correta e completa do produto, utilizando uma notação ou linguagem adequadas. A especificação e a prototipação permitem visões e formas de avaliação complementares do produto concebido. A especificação permite uma descrição e avaliação a partir de técnicas associadas às notações utilizadas. Já a prototipação permite uma descrição e avaliação mais concreta do produto no contexto de utilização.O design de interfaces de usuário é uma atividade que requer análise dos requisitos dos usuários, concepção, especificação e prototipação da interface, e avaliação da utilização do protótipo pelos usuários. Diversos modelos do desenvolvimento de software baseados em prototipação argumentam que este processo deve ser realizado de forma cíclica, isto é, a avaliação deve levar a um novo design e ser posteriomente avaliado. (LEITE, 1998).

Dias (2002), recomenda o uso dos sete princípios de *design* definidos pelo Centro para *Design* Universal, da Universidade Estadual da Carolina do Norte (1995), nos Estados Unidos. Foram definidos a partir de um esforço cooperativo para estabelecer parâmetros para as disciplinas de *design* de produtos, ambientes e comunicações:

- Uso equitativo – o *design* é passível de utilização por qualquer grupo de usuários e oferece as mesmas formas de uso (idênticas ou equivalentes) a todos, sem segregar ou estigmatizar qualquer usuário.
- Flexibilidade no uso – o *design* acomoda uma vasta variedade de preferências e habilidades individuais, oferecendo mais de uma opção de uso ao usuário.
- Uso simples e intuitivo – o *design* é de fácil compreensão indeprendente da experiência, conhecimento ou habilidades verbais do usuário. Elementos complexos desnecessários devem ser eliminados.
- Informação perceptível – o *design* consegue comunicar, com eficácia, a informação necessária ao usuário, independentemente das condições ambientais ou habilidades sensoriais do usuário.
- Tolerância a falhas – o *design* minimiza erros e ações adversas originadas por atos não intencionais ou acidentais do usuário, provendo mensagens elucidativas e alternativas para solucionar falhas.
- Baixo esforço físico – o *design* pode ser usado de forma eficiente e agradável, gerando o mínimo de fadiga ao usuário, minimizando tarefas repetitivas, manipulações complexas e posições desconfortáveis.
- Tamanho e espaço para aproximação e uso – o *design* tem tamanho e espaço apropriado para aproximação, alcance, manipulação e uso, independentemente da mobilidade, postura ou tamanho do corpo do usuário.

Além desses princípios de *design* universal, outros fatores são importantes para um bom design, tais como estética, custo, segurança, adequação cultural e de gênero.

5.4.1.1. Componentes de interface web

Avelareduarte (2009) define componentes de interface como “elementos comuns à maioria dos sites *web* que influenciam o deslocamento, a ação e a experiência do usuário de acordo com soluções amplamente testadas e aceitas”.

5.4.1.1.1. Barras de navegação ou menus

Segundo Linwood (2003) a navegação é apenas um segmento da arquitetura de um site de informação, mas é o segmento mais visível para o usuário final e as barras de navegação é a forma mais prevalente de navegação.

A maioria dos usuários espera que as barras de navegação permaneçam constantes durante todo o período de visita no site. Cada nível pode ter um esquema de navegação de três ou quatro níveis. Mais longe do que isso, é possível prejudicar o espaço disponível o conteúdo do site, especialmente com os locais que disponibilizam *banners*.

É possível usar duas abordagens para criar barras de navegação: com imagens ou texto. As imagens foram mais populares antes da maioria dos navegadores darem suporte a *HTML* dinâmico (*DHTML*) e *Cascading Style Sheets* (*CSS*), porque era mais fácil de mudar uma imagem quando o ponteiro do mouse rolava sobre ela, para dar ao usuário final um efeito visual. O texto é mais rápido para carregar e mais fácil de gerar a partir da estrutura do seu site, e permite mudar a cores dos links com efeitos *DHTML*.

Jakob Nielsen (2008) baseando nos estudos de *eyetracking* (caminho do escaneamento – uma série resultante das fixações e movimentações oculares) em sites recomenda algumas diretrizes para o design de menus verticais:

Não alinhar à direita o menu, porque dificulta a leitura (ver Figura 35 – imagem à esquerda).

Alinhar à esquerda o menu, de modo que os olhos do usuário pode se mover em linha reta e não ter de voltar a adquirir no início de cada nova linha (ver Figura 35 – imagem à direita).

Evitar usar as mesmas palavras para iniciar alguns itens da lista, porque isso torna mais difícil a verificação.

Figura 41 – Design de menu vertical

5.4.1.1.2. Barras de rolagem

De acordo com Avelareduarte (2009) Barras de rolagem são as barras laterais à janela do *browser*, que sinalizam a existência de conteúdo que ultrapassa a área visível da página e permitem o deslocamento da janela para alcançá–lo.

Como muitos usuários relutam em utilizar estas barras, é importante que o conteúdo mais importante de cada página fique localizado na parte visível da página logo que esta carrega no *browser*.

Segundo King (2006), apenas 23% dos usuários que visitam um site pela primeira vez usam as barras de rolagem na página principal e este número diminui ainda mais nas visitas seguintes. As barras de rolagem podem ser horizontais mas são pouco indicadas (pois exigem a rolagem horizontal) e são usadas por apenas 0,4% dos usuários.

De acordo com Avelareduarte (2009), os usuários estão habituados a encontrar nas barras de rolagem os seguintes elementos:

- Setas acionáveis no alto e embaixo, sensíveis ao toque prolongado do *mouse*.
- Deslizador" arrastável, normalmente em cor diferente do fundo, que sinalizar o movimento em direção à parte de cima ou de baixo da página e, pelo seu tamanho, indica a extensão da janela.

Para ajudar o usuário a identificar e imediatamente entender o uso das barras de rolagem é importante que o projetista de *interface* utilize um *layout* semelhante ao das barras de rolagem do *browser* e dos programas que usuário está acostumado a usar.

Quando as barras são muito diferentes da interface do *browser*, sua funcionalidade fica difícil de identificar, especialmente pelos usuários que usam a *internet* há pouco tempo.

Figura 42 – Barra de rolagem em forma de fleha ou remo

Na Figura 36 a barra de rolagem é apresentada em forma de flecha, ou remo. Visualmente é interessante e chama a atenção do público infantil, mas muitas crianças podem passar pela página sem saber que há mais texto na parte de baixo, que pode ser visto tocando–se a parte inferior e superior da imagem

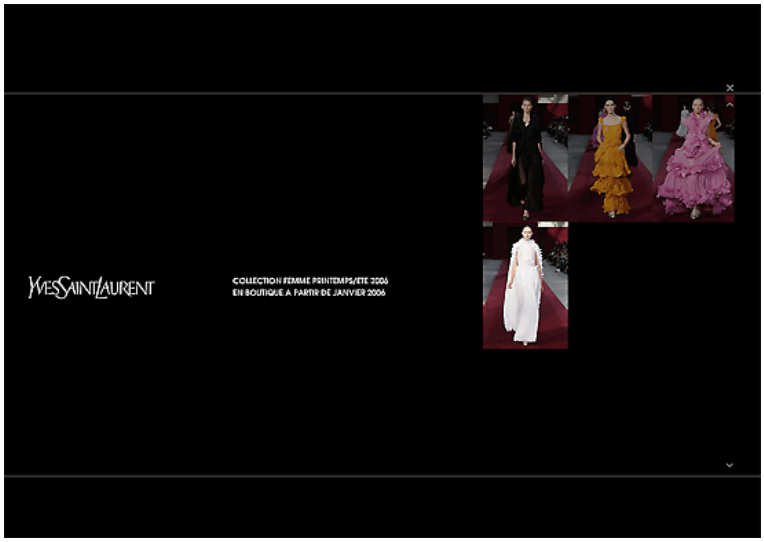

Figura 43 – Barra de rolagem discreta

Na Figura 37, as barras de rolagem muito discretas dificultam a identificação pelo usuário, que deixa de ter acesso a informações que pode estar procurando. São duas setas localizadas à direita, uma apontando para cima e outra apontando para baixo. Alguns usuários podem não encontrá–las facilmente.

Nielsen (2005) recomenda cinco diretrizes de usabilidade indispensável para rolagem e barras de rolagem:

- Oferecer uma barra de rolagem, se tem uma área de rolagem de conteúdo. Não confie na rolagem automática ou arrastando, que as pessoas não podem notar.
- Ocultar todas as barras de rolagem se o conteúdo é visível. Se as pessoas vêem uma barra de rolagem, eles assumem que há conteúdo adicional e será frustrado se não pode rolar.
- Cumprir as normas de uso da GUI e barras de rolagem que se parecem com barras de rolagem.
- Evitar a rolagem horizontal em páginas da Web e minimizá–la em outro lugar.
- Mostrar todas as informações importantes acima da dobra. Usuários muitas vezes decidir ficar ou sair com base no que eles podem ver sem se mover.

5.4.1.1.3. Breadcrumbs

Segundo Avelareduarte (2009), *breadcrumbs* são recursos de navegação que mostram a trilha de acesso a uma página em relação à estrutura do site em que está situada. O nome em inglês é uma referência à história de João e Maria, que, para não se perder na floresta, jogam pedaços de pão para formar uma trilha que sinalize o caminho de volta.

Jacob Nielsen (2007) tem recomendado o uso de *breadcrumb* desde 1995 por alguns motivos que ele considera simples:

- Mostram às pessoas a sua localização atual em relação ao ensino de conceitos de nível.
- Ajuda a compreender onde o usuário está em relação ao resto do site.
- Permite acesso de um clique para níveis mais elevados do site.
- Nunca causam problemas nos testes de usuário.

Os *breadcrumbs* devem mostrar a hierarquia do site, e não uma história do usuário. Quase sempre são implementadas da mesma maneira, com uma linha horizontal que:

- progride do nível mais alto ao mais baixo, a um passo de cada vez;
- começa com a *homepage* e termina com a página atual;
- tem um *link* de texto simples para cada nível (exceto para a página atual, porque você nunca deve ter uma ligação que não faz nada), e
- tem um simples, um separador de caráter entre os níveis (geralmente ">").

A Figura 38 exemplifica um *breadcrumb* no site UseIT.com.

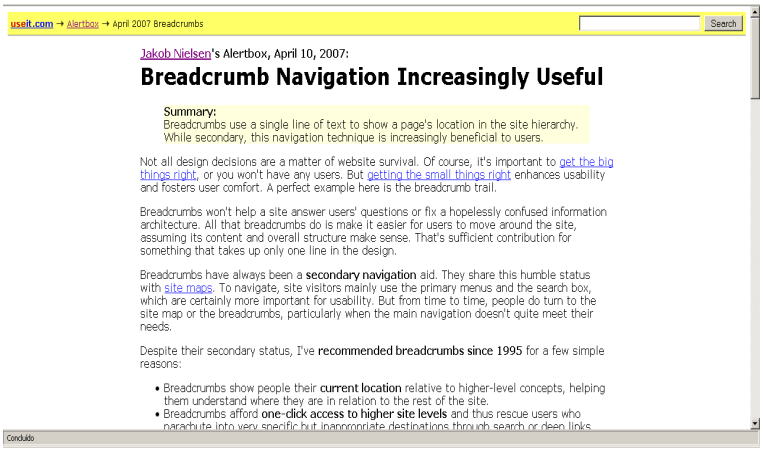

Figura 44 – Exemplo de Breadcrumb

5.4.1.1.4. Formulários

Segundo Avelareduarte (2009), os formulários permitem que o usuário inclua, a partir de um *web* site, diversos tipos de informações e as envie para destinatários definidos, como no caso de pagamento de impostos, compra de mercadorias, contrato de serviços, registro de acesso a um site. As informações encaminhadas podem ser incluídas em bancos de dados e gerar respostas automáticas – como a compra de uma mercadoria, por exemplo. Podem também ser enviadas por email para uma pessoa, responsável pela resposta ou pelo seu encaminhamento a terceiros.

5.4.1.1.5. Formulários HTML/XHTML

Este tópico (4.4.1.1.5) é uma adaptação tradução feita por (SILVA, 2006) do título original *Accessible HTML/XHTML forms* de Ian Lloyd (LLOID) disponibilizado no site da *Web Standard Project*.

Normalmente, para novatos, os formulários se constituem em um aspecto complicado do desenvolvimento *web*, porque sua implementação vai além de incluir informações em elementos da marcação, para serem vistos pelos visitantes do site. Formulários coletam informações e dados a serem processados. E da mesma forma

como deve ser difícil ao novato entender a mecânica de coletar e processar dados, também é difícil o entendimento dos aspectos de acessibilidade relacionados aos formulários.

Conteúdos estáticos do site serão facilmente visualizados e entendidos por usuários de leitores de tela ou até mesmo, talvez, por um computador Braille se cosiderarmos que você usou uma marcação semântica e bem estruturada (tal como h1 – h6 para cabeçalhos). Ler ou deixar de ler um conteúdo estático no site é uma atitute passiva do visitante, mas ao encontrar um formulário o usuário *quer* interagir com ele e de repente muda–se um campo – é uma via de duplo sentido. Para formulários, a marcação torna–se crucial para a correta captura de informações do usuário.

Os elementos *HTML* usados para marcação de formulários são:

- O elemento `<form>` (o "container" para os elementos a seguir);
- `<input>` – podem ser de variados tipos como: `text`, `submit`, `button`, `radio button` e `checkbox`;
- `<textarea>` – para entrada de textos multi–linhas, tais como comentários;
- e o elemento `<select>` – também conhecido como *drop–down–list*, ou *pull–down* menu (à sua preferência);

Existe também o elemento `<button>`, que vem sendo pouco usado ultimamente.

Estes elementos são renderizados e têm um comportamento semelhante em diferentes *browsers* (*GUI–based*). Pode haver algumas diferenças de estilização, mas essencialmente um campo de texto se parece com um campo de texto e uma *drop–down–list* cumpre a finalidade de *uma drop–down–list*. Teoricamente a interação com formulários deveria ser fácil para qualquer um, independente do dispositivo ou *browser* usado.

Quando você lê o título de um artigo no jornal, fica sabendo que o texto que se segue diz respeito àquele título, assim como, o texto que aparece embaixo de uma foto, é a descrição da foto. Por vezes um layout de página não segue padrões e em conseqüência você encontra dificuldade em entender o que se passa – duas

fotos sendo uma ao lado da outra com um texto entre elas, OK você é inteligente e vai acabar descobrindo a que foto se refere o texto.

Para formulários existem alguns consagrados princípios CHI (*Computer–Human Interaction* – Interação Computador–Humano) que funcionam de modo análogo ao jornal, isto é, formulários serão bem mais fáceis de se usar se projetados conforme determinados princípios de uso. Por isto tecnologias assistivas, tais como leitores de tela, seguem àqueles princípios quando interpretam formulários em páginas *web*. Assim, ao encontrar um campo "`text input`", os leitores de tela esperam que o texto descritivo daquele "`text input`" esteja antes do campo e é ali, antes do campo que o leitor vai procurar a descrição! Então, se o leitor de tela encontra algo como o Quadro 28, ele vai ter dificuldade em interpretar o formulário:

```
<input type="text" name="nome" id="nome" /> - Entre seu
primeiro nome<br />
<input type="text" name="sobrenome" id="sobrenome" /> -
Entre seu sobrenome <br />
```

Quadro 40 – Código de text input

Dificuldade por quê? Porque o leitor de tela *poderá* associar incorretamente o segundo input com a primeira linha de texto como mostrado no Quadro 29:

```
<input type="text" name="sobrenome" id="sobrenome" /><--
Entre seu sobrenome<br />
```

Quadro 41 – Texto após o input text

Figura 45 – Exemplo input antes do campo

Esta é a receita para o desastre, colocar o texto depois do campo. Para alguns tipos de input isto é absolutamente correto (Quadro 30):

```
<input type="checkbox" name="athome" id="athome" />
Please call me at home<br />

<input type="checkbox" name="atwork" id="atwork" />
Please call me at work<br />
```

Quadro 42 – Codigo de input checkbox

Considere uma série de *checkboxes*. Observe no *layout* a seguir, tudo bem alinhado (Quadro 43).

```
        [  ] Please call me at hom
        [  ] Please call me at my workplace
        [  ] Please call me at the shack
```

Quadro 43 – Representação da posição do input *checkbox* a esquerda do *label*

Mas se for invertido o alinhamento é perdido:

```
        Please call me at home [  ]
Please call me at my workplace [  ]
Please call me at the shack [  ]
```

Quadro 44 – Representação da posição do input *checkbox* a direita do *label*

Siga as recomendações abaixo e os seus formulários serão automaticamente mais acessíveis, pois é desta forma que os dispositivos adaptados esperam que eles sejam (Quadro 44):

Elemento e tipo HTML	Sequência no layout	Exemplo
input type="text"	Rótulo descritivo, Elemento HTML	`Your First Name  <input type="text" name="txtFirstName" />`
input type="password"	Rótulo descritivo, Elemento HTML	`Your Passnumber  <input type="password" name="txtPassword" />`
input type="button"	Não se aplica (usar atributo)	`<input type="button" name="cmdChkAvail" value="Check Availability" />`
input type="submit"	Não se aplica (usar atributo)	`<input type="submit" name="cmdBookNow" value="Place Booking" />`
input type="radio"	Elemento HTML, Rótulo descritivo	`<input type="radio" name="radMarried" value="Yes" /> Yes, I am married  <input type="radio" name="radMarried" value="No" /> No, I am single `
input type="checkbox"	Elemento HTML, Rótulo descritivo	`<input type="checkbox" name="chkSubscribe" value="Subscribe" /> Subscribe to the newsletter`
select	Rótulo descritivo, Elemento HTML	`Title  <select name="ddlTitle"> <option>Mr</option> ... </select>`
textarea	Rótulo descritivo, Elemento HTML	`Your comments  <textarea name="txtComments"></textarea>`
button	Não se aplica (usar atributo)	`<button name="cmdBigButton">Go on, click me!</button>`

Quadro 45 – Layout para elementos de formulário – CHI Boas técnicas

Seguindo as diretrizes do Quadro 33 e construindo formulários de acordo com os princípios CHI, eles serão mais acessíveis e cumprirão melhor sua finalidade. E, com o propósito de manter as coisas simples, deve-se considerar como usar (ou não usar) scripts para construir formulários mais acessíveis.

Considere que scripts estão desabilitados no lado do usuário. Um formulário deve cumprir sua finalidade independentemente de *scripts* estarem ou não habilitados no *browser* (ainda que eles suportem *scripts* – lembre–se que alguns dispositivos adaptados não suportam *scripts*). Validação e manipulação de dados no lado do servidor possibilita o funcionamento de formulários independentemente de

scripts no lado do usuário. Muitos desenvolvedores ainda se utilizam largamente de scripts no lado do cliente para validação, ou mesmo disparadores de eventos (observe no Quadro 34 um formulário, cujo funcionamento é totalmente dependente de scripts estarem habilitados no lado do usuário:

```
<input type="button" value="Place order"
onclick="document.forms('main'.submit();" />
```

Quadro 46 – Código de formulário com dependência de script no lado do usuário

e uma maneira alternativa não dependente de scripts que cumpre a mesma finalidade:

```
<input type="submit" value="Place order" />
```

Quadro 47 – Código de formulário sem dependência de script no lado do usuário

Assegure–se de que seu formulário funciona com scripts desabilitados – entregue a tarefa ao servidor!

Quais as implicações de se usar *JavaScript* que desencadeia no formulário uma ação fora do controle do usuário? O exemplo clássico ocorre com um elemento de formulário do tipo "*drop–down–list*" usado como ferramenta para navegação. Muitos sites utilizam este recurso que consiste em ao usuário selecionar um item do menu, automaticamente abre–se a página selecionada. Para usuários impossibilitados de usar *mouse* (pessoas cegas navegando com leitores de tela, pessoas com restrições motoras e outros casos mais) o uso do menu torna–se muito difícil (ou mesmo impossível).

Usar o teclado para tentar selecionar as opções da lista não terá efeito, pois resultará sempre em seleção e ativação do primeiro item da lista. A tentativa de encontrar uma solução inteligente resultou em uma armadilha para a acessibilidade. Mantenha as coisas simples – use uma lista "*drop–down–list*" sempre que assim desejar, mas forneça um botão "*Go*" (Ir) para o usuário ativar sua seleção, conforme

mostrado na Figura 47.

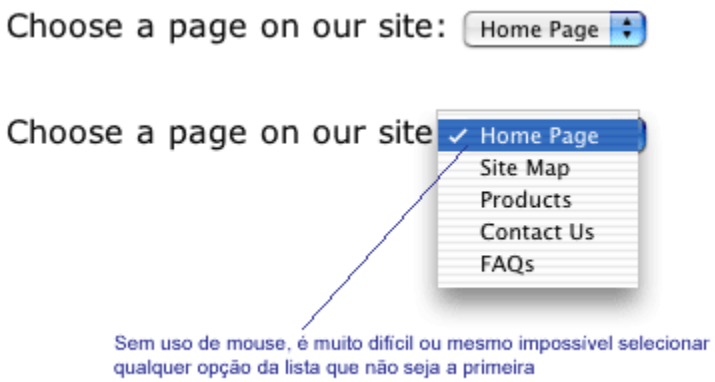

Figura 46 – "*Drop–down–list*" como ferramenta para navegação

Uma outra questão envolvendo a acessibilidade em formulários, diz respeito ao uso de tabelas para se conseguir um *layout* alinhado e certinho. Esta prática não implica necessariamente em um problema de acessibilidade, mas poderá vir a causar problemas – um *layout* de formulário baseado em tabela é irrelevante para um leitor de tela, o conteúdo é lido na ordem em que foi escrito no código fonte. Se você dispor seu *layout* de formulário, agrupando os rótulos descritivos e os controles do formulário você "quebrará" o relacionamento entre eles. Isto pelo fato de que leitores de tela ' linearizam' as tabelas e interpretarão erroneamente o formulário, conforme exemplo mostrado na Figura 41:

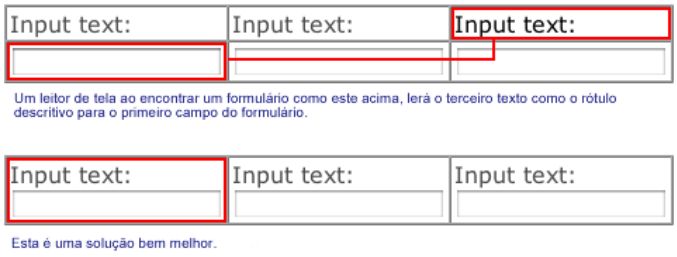

Figura 47 – Layout de formulário

Estas são algumas regras básicas relacionadas ao *layout* e ao posicionamento de elementos de formulários e seus rótulos descritivos, que servirão como um bom ponto de partida para projetar formulários.

5.4.1.1.6. Ícones

A escrita ocidental é baseada em um alfabeto composto de letras (unidades mínimas) que devem ser associadas para formar palavras, que agrupadas formam frases. Há situações em que a leitura e compreenção de uma mensagem comprida deve ser instantânea. Frases compridas como: "Há uma lombada adiante; diminua a velocidade" podem causar acidentes pela demora na leitura. Para superar esta e outras situações surgiram os ícones.

De acordo com Instituto Brasileiro de Amigabilidade e Usabilidade – IBRAU (2006), um ícone é um símbolo gráfico cuja visualização recupera, da memória de curto ou longo prazo, lembranças relacionadas a vários fatores: perigos, alertas, opções, ações, etc. (ver na Figura 42 alguns exemplos de ícones). Avelareduarte (2009) relata que em sites com usuários regulares, ícones podem ajudar a simplificar a navegação, através da rápida visualização de *links* e informações a eles associadas.

Figura 48 – Exemplos de ícones

Os ícones para serem melhores compreendidos seu *layout* deve estar associado a elementos conhecidos no contexto sócio–cultural do usuário. A Figura 43 lembra um controle de um aparelho de som, ou de vídeo, ou de um *DVD*.

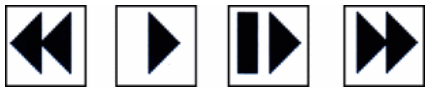

Figura 49 – Ícone de multimedia

Segundo Norman, apud. IBRAU (2006) há três tipos de ícones (Quadro 36):

	Os que representam os **objetos** que serão manipulados

	Os que representam as **operações** ou os operadores
	Os que representam **operadores** atuando sobre objetos

Quadro 48 – Tipos de ícones

Na sistemas *desktop* ou *web* os ícones podem ser utilizados para diversas finalidades como: ativação de menus, iniciação a execução de ações, manipulação de janelas, representação de arquivos, diretórios e estruturas, entre outros.

Dentre várias vantagens dos ícones se destacam:

- Um ícone, desde que corretamente projetado, dispensa leitura, análise, reconhecimento ou tradução.
- É compreensível até por pessoas não alfabetizadas.
- É compreendido rapidamente: “Estudos na compreensão de sinalização rodoviária demonstram que um ícone pode ser reconhecido ao dobro de distância e na metade do tempo que um sinal escrito” .
- Contribui à facilidade de (re)aprendizado.
- Projetados adequadamente, contribuem à optimização de espaço na tela.
- Há extensas bibliotecas com ícones prontos de acesso gratuito na internet.

Os ícones são um grande aliado do projetista. Sua capacidade de compreensão imediata torna–os extremamente úteis para destacar ações, sinalizar eventos e representar estados. Na decisão de seu uso, deve–se tomar o cuidado utilizá–los com moderação para não caursar poluição visual nem aumentar o tempo de carregamento das páginas em sistemas web.

5.4.1.1.7. Título da página

Segundo AVELAREDUARTE (2009) um título da página identifica o assunto da página. É o texto registrado por default nos favoritos dos *browsers* e é um

lembrete muito útil sobre o conteúdo da página para os usuários dos programas.

É uma informação que merece atenção especial, pois algumas ferramentas de busca mostram o título da página na lista de resultados, associada à sua URL.

Cada página de um site deve ter um título diferente, para identificar seu conteúdo e sua especificidade. Os títulos ou cabeçalhos das páginas devem corresponder exatamente aos termos utilizados nos links que apontam para essas páginas.

Para ajudar a indexação dos sites de busca e o rastreamento da navegação pelo usuário, cada página deve ter um título específico, que explique o conteúdo da página e faça sentido fora do contexto do site. Se o usuário lê o título no resultado de uma ferramenta de busca, deve poder entendê–lo para que possa selecioná–lo.

5.4.1.2. Resolução do monitor e tamanho do *website*

A resolução de um monitor indica a quantidade de pixels (pontos) que formam a imagem que aparece em uma tela de dispositivo digital. A resolução de um monitor com 800 x 600, por exemplo, mostra 800 pixels em cada uma das 600 linhas do monitor, totalizando 480.000 pixels.

A qualidade da definição de uma imagem depende da relação entre o número de pixels por polegada quadrada (dpi, dots per inch) com que a tela está configurada, sua resolução, e o tamanho do monitor. Quanto maior o número de pontos que forma uma imagem, maior o monitor, maior será a definição da imagem que aparecerá para os usuários.

Se, no entanto, um monitor grande é configurado com baixa resolução de pontos, mostra imagens indefinidas e embaçadas, são criados pixels "falsos" a partir da imagem de menor tamanho para ocupação de toda a sua área na tela.

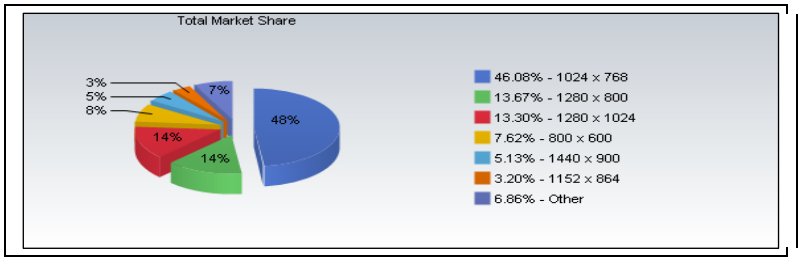

Figura 50 – Resoluções de tela utilizadas em novembro de 2007

O site Market Share disponibiliza um resultado de pesquisa de resoluções de telas utilizadas para visualização de páginas web. Na Figura 44 é possível visualizar que em novembro de 2007 a resolução 1024x768 era utilizada por 46,08% dos internautas e 800x600 por 7,62%. O cenário muda em novembro de 2008 (Figura 45) com a resolução 1024x768 em queda para 37,59% e 800x600 para 4,69%. Na Figura 0012 mostra que em novembro de 2009 a resolução 1024x768 está com 30,25% e 800x600 com 3,58%. De acordo com as Figuras 44, 45 e 45 a perda de percentual foi devido ao aumento de uso de telas com resolução maior que 1024x768.

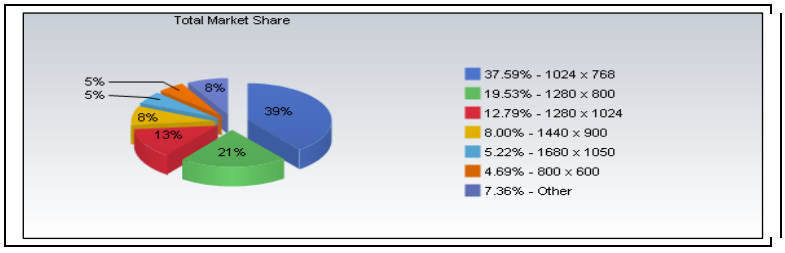

Figura 51 – Resoluções de tela utilizadas em novembro de 2008

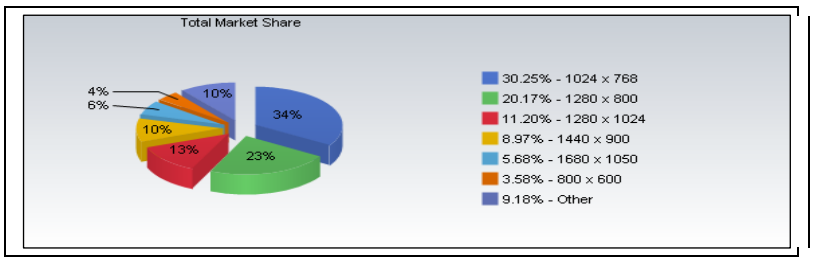

Figura 52 – Resoluções de tela utilizadas em novembro de 2009

5.4.1.3. *Layout* líquido

Por apresentar compatibilidade com os monitores mais antigos, muitos desenvolvedores e *designers* ainda consideram a resolução de 800x600 como base, com a largura–referência da interface de 750px, que leva em conta as dimensões do

browser e de barra de rolagem laterais.

Especialistas da *web* como Jakob Nielsen recomenda o desenvolvimento de interfaces baseadas no chamado "*layout* líquido", em que os limites laterais das páginas são definidos em percentuais em vez de pixels. Essa técnica é uma otimização que faz com que uma página criada no tamanho 1024x768 funcione bem em outros tamanhos com um bom aspecto.

Existem vários sites utilizando esta otimização, a Organização W3C é um exemplo, na Figura 47 é possível visualizar a página na resolução 1024x768. A visualização 800x600 é visto na Figura 48 e Figura 49 o mesmo site é apresentado em resolução 1280x800. Independente da resolução o site ficou com um ótimo aspecto.

Figura 53 – Site W3C visualizado na resolução 1024x768

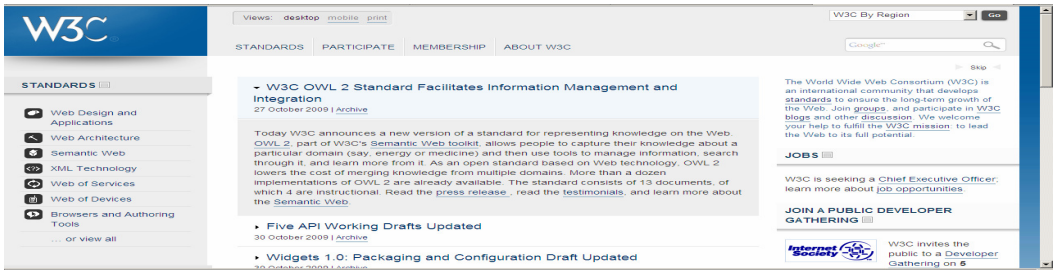

Figura 54 – Site W3C visualizado na resolução 1280x800

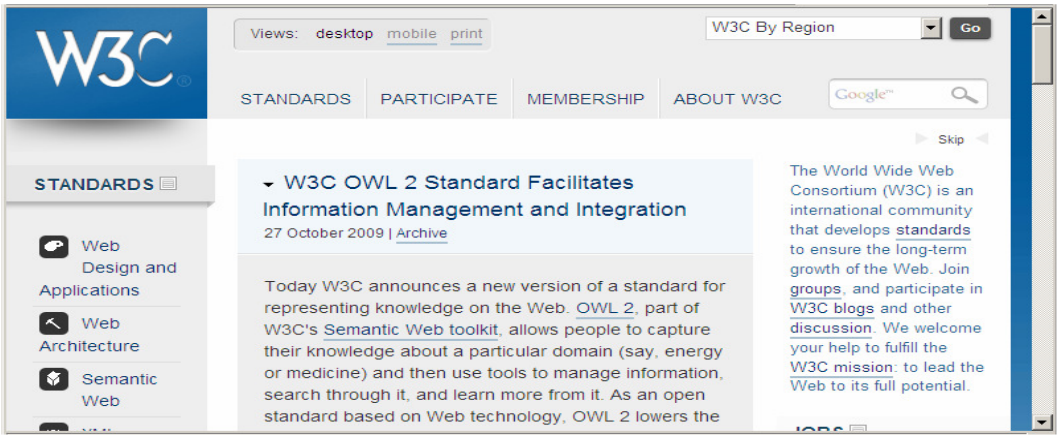

Figura 55 – Site W3C visualizado na resolução 800x600

Nielsen (2009) recomenda também três critérios a considerar na adaptação da interface à resolução de tela:

- Visibilidade inicial: Informações mais importantes da página localizadas no alto, para que o usuário possa tomar conhecimento do assunto geral sem precisar fazer rolagem vertical.
- Estética: Aparência da página e composição dos elementos dispostos de maneira harmônica em diferentes resoluções de tela.
- Legibilidade: Textos fáceis de ler em páginas com colunas de diversas larguras, especialmente colunas nas mais estreitas.
- O *layout* líquido por não ser uma solução perfeita, apresenta algumas desvantagens:
- Dificuldade na leitura em janelas pequenas, em que a largura da mancha que contém os textos fica muito estreita, especialmente em telas de baixa resolução. Este problema pode ser atenuado com a configuração de larguras mínimas de colunas de textos.
- Se o *layout* ocupa toda a largura da janela do *browser* numa tela de alta resolução, as linhas de textos podem ficar muito longas e difíceis de ler, o que pode ser atenuado com a configuração da largura para apenas 85% da página.

5.4.1.4. Espaço do *browser* na tela

Os componentes de um navegador ou *browser* (menu, outros atalhos, barras de rolagem vertical e horizontal) ocupam espaço na tela e seus tamanhos mudam em tipo de *browser* e também de acordo com a resolução da tela utilizada. Para criar *layouts* adequados a estas telas, é necessário considerar a largura em *pixels* e subtrair o espaço ocupado pelo *browser* da largura total.

Segundo MCCLURG (2006),[...] não podemos usar os números "brutos" para determinar como um grande projeto deve ser. Temos de considerar o espaço ocupado pelo próprio navegador, e subtrair que a partir do tamanho da tela global para obter uma imagem mais precisa.

Assim, quanto espaço um navegador ocupa? Depende do navegador e suas configurações padrão. A melhor maneira de aproximar isso é configurar um monitor de tela no tamanho apropriado, ampliar o navegador, e ter uma captura de tela. Você pode então ter uma idéia de quanto espaço está realmente disponível para exibir informações. No entanto, devido às diferenças nos hábitos de navegação do usuário, é difícil chegar a números definitivos sobre o melhor tamanho para trabalhar. As pessoas não podem ter seus navegadores completamente expandidos, e eles podem ter muitas ou poucas barras de abrir a qualquer momento. Esta técnica permite uma aproximação do tamanho de trabalhar menos para um projeto de largura fixa. (Tradução nossa).

Tamanho da Tela	IE 6	Firefox	Opera	Mozilla	Netscape
800 × 600	779 × 400	781 × 434	777 × 427	779 × 420	781 × 389
1024 × 768	1003 × 568	1005 × 602	1001 × 595	1003 × 588	1005 × 557

Quadro 49 – Valor real aproximado. FONTE: adaptada de MCCLURG (2006).

O Quadro 37 contém valores padrão para um site recomendados por MCCLURG (2006) Os números abaixo mostram a propriedade "aproximados real", disponível para os dois tamanhos de tela dominante e as versões atuais dos *browsers* mais populares. Estes números assumem um padrão de instalação (pelo menos no menu Arquivo, botões padrão, barra de endereços, barra de favoritos, barra de *status* e barra de rolagem). Os valores usados para a elaboração de *layouts* devem considerar os valores *default*, usados pela maioria dos usuários.

O *layout* líquido é uma maneira de controlar os problemas de diferentes configurações de usuário, deve–se também planejar com precisão as áreas que deverão ou não esticar ou encolher para melhor controlar o resultado em cada situação.

5.4.1.5. Localização dos elementos na tela

Segundo AVELAREDUARTE (2009), A maioria dos usuários quando acessa um site desconhecido procura alguns elementos em lugares consagrados pelo uso […] se num site os visitantes encontram facilmente o que procuram, podem se concentrar mais facilmente nas tarefas e nas informações que os levaram até ali.

Também de acordo com AVELAREDUARTE (2009) os usuários estão acostumados a encontrar diferentes tipos de informações disponíveis na página agrupadas, sendo as mais importantes mostradas em primeiro lugar. No Quadro 38 é possível visualizar o outros elementos que são encontrados nos sites em várias posições.

Elemento de Interface	Posicionamento
Logotipo	no alto, à esquerda da página
Menu de navegação	no alto, com links dispostos horizontalmente, ou no lado esquerdo da página, com links dispostos verticalmente
Links para "Voltar à Principal	na área superior esquerda da página
Buscador interno	no alto, na área central ou na parte direita da página
Informações para contato	no alto da página, à direita
Banners e anúncios	no alto ou ao longo do lado direito
Menus drop down e atalhos para páginas	áreas mais profundas no alto da área de conteúdo ou ao longo do lado direito
Campos login e senha	no alto, à esquerda da página
Breadcrumbs ou "migalhas de pão"	no alto da área de conteúdo ou embaixo do logotipo, de forma que fiquem facilmente associados à localização da página em relação à Principal
Carrinho de compras (sites de comércio)	no alto da página
Informações sobre as compras	normalmente no alto
Agrupamentos de links internos	na área esquerda das páginas
Artigos em promoção (sites de comércio)	no alto da área de conteúdo

Área de conteúdo	na área central da página
Links internos para informação adicional	na área à direita da página
Links para outros sites	na parte inferior da página
Informações institucionais	no alto da página em sites institucionais ou na barra localizada na parte de baixo, como nos sites de comércio, em que as informações institucionais não são prioritárias
Links para "Voltar ao alto"	na base de páginas muito altas.
Informações sobre o site e mapa do site na base da página	eventualmente também no alto da página

Quadro 50 – Posicionamento de elementos de interface. FONTE: adaptada de AVELAREDUARTE (2009).

O posicionamento dos elementos de interface também foi estudado por pesquisadores como Nielsen, Adiksson, Krung, Lynch e Bernard. O resultado do estudo encontra–se na Figura 49.

ELEMENTO DE INTERFACE	POSICIONAMENTO	PESQUISADOR
Marca da empresa	Canto superior esquerdo	Nielsen, Adiksson e Bernard
Buscador	Parte superior	Nielsen, Adiksson e Bernard
Navegação global	Parte superior na horizontal	Nielsen, Adiksson e Krug
Navegação local	Coluna esquerda	Nielsen, Adiksson e Bernard
Breadcrumbs	Abaixo da marca da empresa	Adiksson, Chaparro e King
Conteúdo global e contextual	Área central	Bernard
Navegação de rodapé	Parte inferior	Nielsen, Krug e Lynch

Fonte: www.experienciaperfeita.org

Figura 56 – Recomendação para posicionamento de elementos

Ao analisar os dados dos elementos: marca da empresa, buscador, navegação, *beadcrubs* e conteúdo nota–se que a Figura 50 reforça o resultado do quadro 34 destes mesmos elementos.

5.4.1.6. Navegação do sistema

De onde vim ou onde estive? Refere–se ao sentido de localização do usuário em relação às páginas já visitadas. Em listas ou conjuntos de *links*, esta informação é especialmente necessária, pois muitas vezes é impossível lembrar todos os *links* selecionados e todas as páginas visitadas.

Onde estou? Refere–se ao sentido de localização do usuário em relação a um site e à *web* em geral, de forma que se oriente adequadamente para encontrar as informações que procura, ou realizar as tarefas a que se propõe ao selecioná–lo.

Um ótimo recurso para ajudar na localização do usuário na página são os *breadcrumbs*, que mostra a estruturada hierárquica da página onde o usuário se encontra. Outro excelente recurso (muito utilizado) é mudar a cor dos *links* em que as páginas foram visitadas

Para onde vou ou onde posso ir e como chegar? Refere–se ao sentido de localização do usuário em relação à estrutura de informações, que leva–o a encontrar o que está procurando, seja uma notícia, um produto para compra, um texto acadêmico.

Um sistema bem definido à primeira vista, com *links* visíveis e identificáveis, ajuda o usuário a se deslocar sem erros ou expectativas não correspondidas, reforçando o seu sentido de localização dentro do sistema.

5.4.1.7. Layout

No ano de 2003, Dave Shea lançou um *site* chamado CSS *Zen Garden*, foi um grande impacto para os profissionais *web* porque demonstrou verdadeiramente que o design inteiro pode mudar apenas alterando o estilo da página e o conteúdo pode permanecer idêntico. A Figura 51 ilustra o tema padrão Tranquille do site CSS Zen Garden. As Figuras 52, 53, 54 e 55 são quatro dos 200 estilos disponibilizados no site.

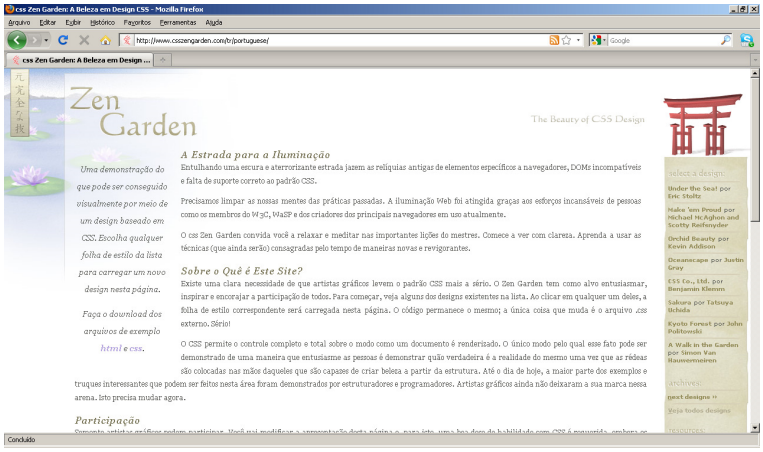

Figura 57– Site Zen Garden – Tema Tranquille

Figura 58– Site Zen Garden – Tema CSS Co., Ltda

Figura 59 – Site Zen Garden – Tema Under the Sea

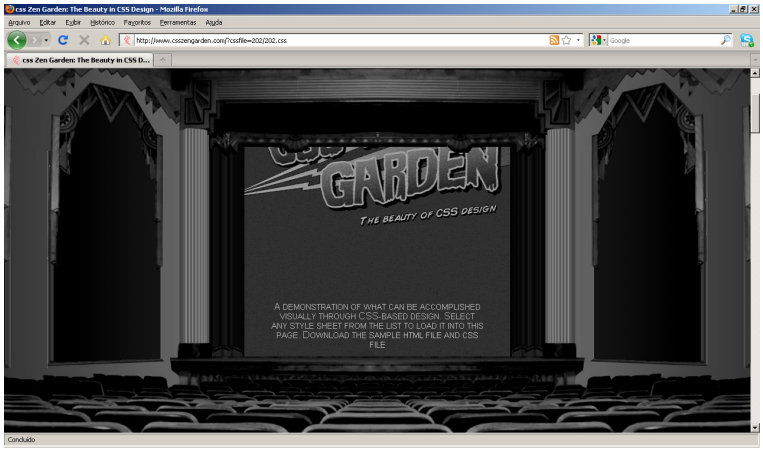

Figura 60 – Site Zen Garden – Tema Retro Theater.

Figura 61 – Site Zen Garden – Tema Wiggles the Wonderworm.

5.4.2 Tecnologias

5.4.2.1. CSS

Segundo a *W3SCHOOLS* (2009), CSS significa *Cascading Style Sheets*. Os estilos definem como exibir os elementos *HTML*. Estilos foram adicionados ao *HTML* 4.0 para resolver um problema. As folhas de estilo externas podem salvar um monte de trabalho. As folhas de estilo externas são armazenadas em arquivos CSS (Tradução nossa).

5.4.2.1.1. CSS resolve um grande problema

HTML nunca foi destinado a conter *tags* para a formatação de um documento. É destinada a definir o conteúdo de um documento, como:

- <h1> Este é um título </ h1>
- <p> Este é um parágrafo. </ p>

Quando *<font>* como *tags*, e atributos de cor foram adicionados à especificação HTML 3.2, iniciou–se um pesadelo para os desenvolvedores web. Desenvolvimento de web sites grandes, onde as fontes e informações de cores foram adicionadas a cada página, tornou–se um processo longo e caro.

Para resolver este problema, a *World Wide Web Consortium* (W3C) criou CSS. Em HTML 4,0, toda a formatação pode ser removido de um documento HTML, e armazenado em um arquivo CSS separado. Folhas de estilo externas permitem alterar a aparência eo layout de todas as páginas em um site, apenas editando um único arquivo e todos os navegadores hoje suportam CSS.

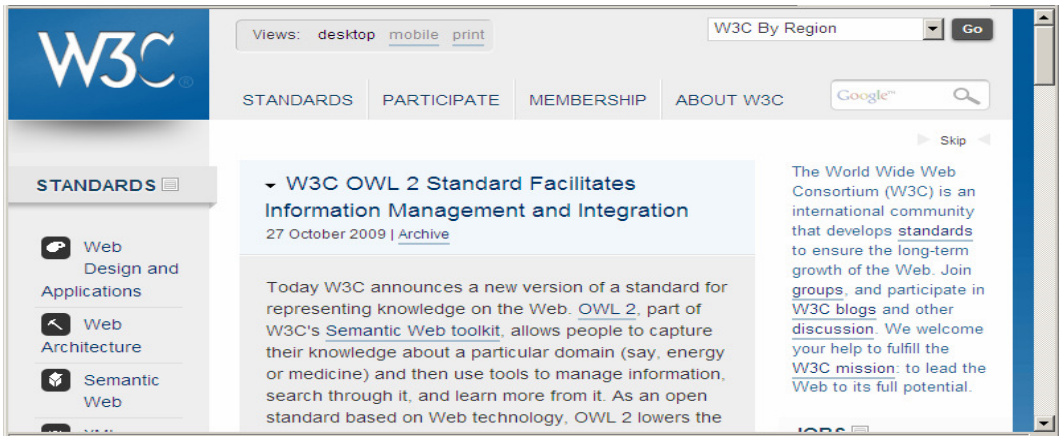

Figura 62 – Site W3C visualizado na resolução 800x600

Nielsen (2006) recomenda também três critérios a considerar na adaptação da interface à resolução de tela:

- Visibilidade inicial: Informações mais importantes da página localizadas no alto, para que o usuário possa tomar conhecimento do assunto geral sem precisar fazer rolagem vertical.

- Estética: Aparência da página e composição dos elementos dispostos de maneira harmônica em diferentes resoluções de tela.
- Legibilidade: Textos fáceis de ler em páginas com colunas de diversas larguras, especialmente colunas nas mais estreitas.

O layout líquido por não ser uma solução perfeita, apresenta algumas desvantagens:

- Dificuldade na leitura em janelas pequenas, em que a largura da mancha que contém os textos fica muito estreita, especialmente em telas de baixa resolução. Este problema pode ser atenuado com a configuração de larguras mínimas de colunas de textos.
- Se o layout ocupa toda a largura da janela do *browser* numa tela de alta resolução, as linhas de textos podem ficar muito longas e difíceis de ler, o que pode ser atenuado com a configuração da largura para apenas 85% da página.

5.4.2.2. Espaço do browser na tela

5.4.2.2.1. Sintaxe CSS

Uma regra CSS tem duas partes principais (ver Figura 57):

- um selector, e uma ou mais declarações.

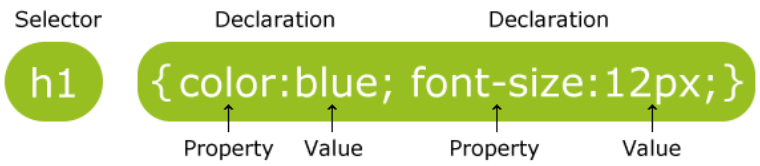

Figura 63 – Sintaxe CSS

O seletor é normalmente o elemento HTML que deseja estilo (ex: h1).

Cada declaração consiste de uma propriedade (ex: color) e um valor (ex: *blue*). A propriedade é o atributo *style* você deseja alterar. Cada propriedade tem um

valor. Declarações CSS sempre terminam com um ponto e vírgula, e os grupos de declaração são cercados por chaves. Exemplo no Quadro 39:

```
p {color:red; text-align:center;}
```

Quadro 51 – Código CSS

Para fazer o CSS mais legível, você pode colocar uma declaração em cada linha, como este exemplo do Quadro 40:

```
p
{
color:red;
text-align:center;
}
```

Quadro 52 – Código CSS mais legível

5.4.2.2.2. Comentários CSS

Comentários são usados para explicar o seu código, e pode ajudá–lo quando você editar o código–fonte em uma data posterior. Comentários são ignorados pelo navegador.

Um comentário começa com CSS "/*", e termina com "*/", como este, do Quadro 41:

```
/ * Este é um comentário * /
p{text-align:center;color: black;font-family:Arial;}
```

Quadro 53 – Código de comentário CSS

5.4.2.2.3. Os seletores de id e class

Além de definir um estilo para um elemento *HTML*, *CSS* permite que você especificar os seus próprios seletores chamado "id" e "classe".

a) O seletor id

O seletor id é usado para especificar um estilo para um elemento único e exclusivo. O seletor usa o atributo id do elemento HTML e é definido com um "#". A regra de estilo é aplicada para o elemento com id = "para1" (Quadro 42).

```
#para1
{
text-align:center;
color:red ;
}
```

Quadro 54 – Código de seletor id em CSS

Observação: Não começar um nome de identificação com um número. Não irá funcionar no *Mozilla* / *Firefox*.

b) A classe Selector

O seletor de classe é usada para especificar um estilo para um grupo de elementos. Ao contrário do seletor de id, seletor de classe é mais freqüentemente usado em vários elementos. Isto permite–lhe definir um estilo especial para todos os elementos de HTML com a mesma classe. O seletor classe usa o atributo class de HTML, e é definido com um "." No exemplo do Quadro 43, todos os elementos HTML com class = "centro" serão alinhados ao centro.

```
.center {text-align:center;}
```

Quadro 55 – Código CSS de seletor classe

Também é possível especificar que apenas os elementos HTML específicos devem ser afetadas por uma classe. No exemplo do Quadro 44, todos os elementos com p class = "centro" será alinhado ao centro. Exemplo:

```
p.center {text-align:center;}
```

Quadro 56 – Código CSS de seletor classe para tag p

Observação: Não começar um nome de classe com um número! Isso só é suportado no Internet Explorer.

5.4.2.2.4. Como inserir CSS

Há três maneiras de inserir uma folha de estilo:

- Externo folha de estilos
- Interno folha de estilo
- Estilo inline

a) CSS Externa

Uma folha de estilo externa é ideal quando o estilo é aplicado a muitas páginas. Com uma folha de estilo externa, você pode mudar a aparência de um site inteiro mudando um arquivo. Cada página tem link para a folha de estilo usando a *tag* <*link*>. A *tag* <*link*> vai dentro da seção *head* (Quadro 45):

```
<head>
<link rel="stylesheet" type="text/css"
href="mystyle.css" />
</head>
```

Quadro 57 – Link Html para CSS externa

Uma folha de estilo externa pode ser escrito em qualquer editor de texto. O arquivo não deve conter quaisquer tags HTML. A folha de estilo deve ser salvo com uma extensão .css. Um exemplo de um arquivo de folha de estilo é mostrado no Quadro 58:

```
hr {color:sienna;}
p {margin-left:20px;}
body {background-image:url("images/back40.gif");}
```

Quadro 58 – Exemplo de folha de estilo

Observação: Não deixar espaços entre o valor da propriedade e as unidades como em "*margin–left:20 px*" (correto: "*margin–left:20px*") irá funcionar no *Internet Explorer*, mas não no *Firefox* ou *Opera*.

b) CSS Interna

Uma folha de estilo interna deve ser usado quando um documento único, tem um estilo único. Você define estilos internos na seção principal de uma página HTML, usando a *tag* <*style*>, ver Quadro 59:

```
<head>
<style type="text/css">
hr {color:sienna}
p {margin-left:20px}
body {background-image:url("images/back40.gif")}
</style>
</head>
```

Quadro 59 – Exemplo de código CSS interno

c) CSS Inline

Um estilo *inline* perde–se muitas das vantagens das folhas de estilo, misturando conteúdo com apresentação. Use este método com moderação! Para usar estilos *inline* você usar o atributo de estilo na marca relevante. O atributo *style* pode conter qualquer propriedade *CSS*. O exemplo do Quadro 48 mostra como alterar a cor e a margem esquerda de um parágrafo:

```
<p style="color:sienna;margin-left:20px">Este é um parágrafo.</p>
```

Quadro 60 – Exemplo de código CSS inline

d) CSS Múltiplos

Se algumas propriedades foram definidas para o mesmo seletor em folhas de estilos diferentes, os valores serão herdadas da folha de estilo mais específico.

Por exemplo, uma folha de estilo externa tem estas propriedades para o seletor h3 (Quadro 49):

```
h3{color:red; text-align:left; font-size:8pt;}
```

Quadro 61 – Exemplo de código CSS externo

E uma folha de estilo interna tem estas propriedades para o seletor h3 (Quadro 62):

```
h3{text-align:right; font-size:20pt;}
```

Quadro 62 – Exemplo de código CSS interno

Se a página com a folha de estilo interna também *links* para a folha de estilos externa as propriedades para h3 será:

- *color:red;*
- *text–align:right;*
- *font–size:20pt*

A cor é herdado da folha de estilo e alinhamento do texto e *font–size* passam a ter a folha de estilo interna.

É possível também criar vários arquivos *CSS* (ex: menu, topo, conteúdo e rodapé) e usá–los no sistema, bastando referenciá–los nos seguintes locais da página:

- dentro de um elemento HTML

- dentro da seção head de uma página HTML
- em um arquivo CSS externo

De um modo geral, pode–se dizer que todos os estilos em cascata "em uma nova folha de estilos" virtual "pelas regras a seguir, onde o número quatro tem a maior prioridade:

1. Browser padrão
2. Externo folha de estilos
3. Interno folha de estilo (na seção de cabeça)
4. Estilo inline (dentro de um elemento HTML)

Assim, um estilo inline (dentro de um elemento HTML) tem a maior prioridade, o que significa que ele irá substituir um estilo definido dentro da tag <head>, ou em uma folha de estilo externa, ou em um navegador (um valor padrão).

Observação: Se o link para a folha de estilo externa é colocada após a folha de estilo interna em HTML <head>, a folha de estilos externa irá substituir a folha de estilo interna!

Um excelente exemplo de aplicação de CSS é o site CSS Zen Garden que demonstra verdadeiramente que o design inteiro pode mudar apenas alterando o estilo da página (CSS) e o conteúdo permanece idêntico.

5.4.2.3. DisplayTag

A biblioteca de *tag* de exibição é uma suíte open *source* de *custom tags* que fornecem elevados padrões de apresentação *web* que irá trabalhar em um modelo MVC. A biblioteca fornece uma quantidade significativa de funcionalidade enquanto ainda está sendo fácil de usar.

A *DisplayTag* trata da exibição da coluna, classificação, paginação, corte, agrupamento e exportação, interligando e decorando com um mesmo estilo XHTML personalizável os dados fornecidos de uma lista.

5.4.2.3.1. Recursos disponíveis

- **Paginação** – Imagine uma aplicação *web* em que é necessário fazer uma consulta ao banco de dados e retornar centenas de registros e apresentá–los página web. Como mostrar estas informações sem visualizar uma tabela enorme que demora muito para ser carregada no navegador? Uma solução clássica é a paginação de resultados: quebrar uma coleção de registros em várias páginas de tamanho limitado. A *DisplayTag* suporta diversas estratégias de paginação e oferece barras customizáveis para a navegação dos resultados.
- **Exportação de dados** – Através do recurso de exportação, que é ativado alterando–se apenas um atributo, o usuário pode transformar a tabela exibida para vários formatos. A *DisplayTag* inclui filtros de exportação predefinidos para CSV (valores separados por vírgulas), Excel, XML e PDF simples.
- **Internacionalização** (i18n) – Normalmente implementa–se a internacionalização através de vários arquivos texto (chamados *resources*), um para cada idioma desejado. Estes arquivos são compostos de pares "chave=mensagem", sendo as chaves fixas e as mensagens customizadas para o idioma. A *DisplayTag* suporta a internacionalização dos cabeçalhos das tabelas e das barras de paginação, entre outros elementos.
- **Flexibilidade visual com *Decorator*** – O *design pattern Decorator*, implementado pela *DisplayTag*, aumenta a flexibilidade do design, através de objetos que customizam ou adicionam dinamicamente comportamentos a outro objeto. Geralmente utilizam–se este *pattern* quando se precisa visualizar uma informação de várias formas diferentes. Por exemplo, apresentar formatos distintos para datas, números, valores monetários etc. sem precisar alterar a classe de negócio.

5.4.2.3.2. As cinco tags fundamentais

A *DisplayTag* é composta pelas *tags caption*, *column*, *footer*, *setProperty* e *table*, todas com prefixo ***display***. No Quadro 51 é apresentado função de cada uma dessas *tags*.

Tag	Função
caption	Cria um cabeçalho na tabela.
column	Cria uma coluna na tabela.
footer	Cria um rodapé na tabela.
setProperty	Permite definir o valor de propriedades relacionadas à paginação, exportação etc.
table	Cria uma tabela. Todas as outras tags da DisplayTag são utilizadas dentro desta tag.

Quadro 63 – Cinco tags fundamentais da tags do DisplayTag

As tabelas da Figura 64 foram geradas a partir de listas usando a *tag* <*display:table*>:

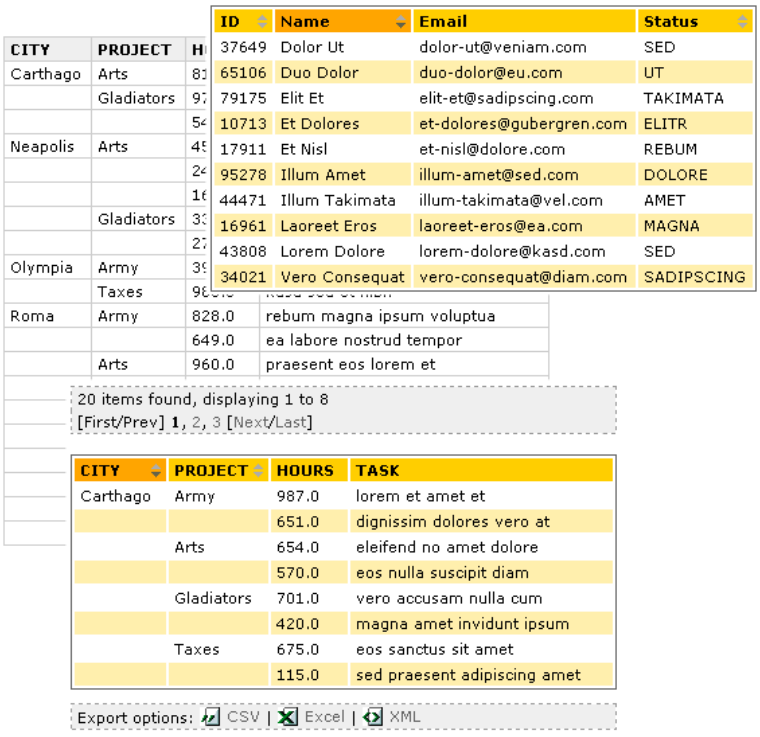

Figura 64 – Tabelas geradas pelo DisplayTag
(Fonte: http://displaytag.sourceforge.net/1.2/)

5.4.3 Ferramentas

5.4.3.1. Corel Draw

O *CorelDRAW* é um programa de desenho vetorial bidimensional para design gráfico desenvolvido pela *Corel Corporation*, Canadá. É um aplicativo de ilustração vetorial e layout de página que possibilita a criação e a manipulação de vários produtos, como por exemplo: desenhos artísticos, publicitários, logotipos, capas de revistas, livros, CDs, imagens de objetos para aplicação nas páginas de Internet (botões, ícones, animações gráficas, etc) confecção de cartazes, etc.

O *CorelDRAW* surgiu em 1988, apenas em inglês. Em 1995, surgiu a primeira versão em 32 bits (*CorelDRAW* 6). Dois anos depois surgiu a primeira versão para computadores Macintosh. No ano seguinte, foi lançada a primeira versão para *Linux*. Em 2003, surgiu a versão 12 para *Windows* XP. A última versão criada em 2008 se denomina *Corel* X4 ("X" em algarismos romanos =10 + 4= versão 14).

Em ambiente software proprietário os concorrentes diretos são os programas Adobe *Illustrator* e *Macromedia Freehand*. Em software livre, para ambiente *Linux*, o principal concorrente é o *Inkscape*.

5.4.3.2. Adobe Photoshop

Adobe *Photoshop* é um software caracterizado como editor de imagens bidimensionais do tipo *raster* (possuindo ainda algumas capacidades de edição típicas dos editores vetoriais) desenvolvido pela *Adobe Systems*. É considerado o líder no mercado dos editores de imagem profissionais, assim como o programa de fato para edição profissional de imagens digitais e trabalhos de pré–impressão.

Sua mais recente versão é apelidada como Adobe *Photoshop CS4* (sigla cujo significado é *Creative Suite* 4, correspondente à décima primeira edição desde seu lançamento), disponível para os sistemas operativos *Microsoft Windows* e *Mac OS* X. Em *software* livre, para ambiente *Linux*, pode ser rodado através da camada de compatibilidade *Wine* e o principal concorrente é o *Gimp*.

5.5 Componente: Comportamento

A camada de comportamento contém todos os efeitos e todas as decisões funcionais da interface. Manter a programação separada da estrutura facilita a

manutenção e evita erros.

5.5.1 ECMAScript ou JavaScript

O Nome oficial do *JavaScript* é *ECMAScript* (desenvolvido e mantido pela organização ECMA). A línguagem foi criada por Brendan Eich da *Netscape* e tem aparecido em todos os *browsers Netscape* e da Microsoft desde 1996.

JavaScript foi concebido para adicionar interatividade a páginas HTML. Segundo a (W3SCHOOLS), *JavaScript* significa:

> JavaScript é uma linguagem de script. A linguagem de script é uma linguagem de programação leve. JavaScript é normalmente incorporada diretamente em páginas HTML. JavaScript é uma linguagem interpretada (significa que executará scripts sem compilação preliminar). Todos podem usar o JavaScript sem comprar uma licença. (Tradução nossa)

Na lista abaixo seguem algumas possibilidades do *JavaScript*, que ultrapassam os limites da linguagem de marcação *HTML* e da linguagem de estilos *CSS* que incluem:

- Efetuar operações matemáticas;
- Gerar páginas de acordo com características do *browser* e do computador do cliente e em tempo de carregamento;
- Tratar determinados eventos, interagindo com o usuário através desse tratamento;
- Manipular janelas do browser, inclusive trocando informações entre janelas;
- Modificar propriedades da página dinamicamente, pois essa é considerada um conjunto de objetos;
- Efetuar verificação de formulários.

5.5.2 Apache Tiles

De acordo com a *Apache Sofware Foundation* (2009), Apache *Tiles* é um *framework* de *templating* construído para simplificar o desenvolvimento de interfaces de usuário da aplicação *web*.

Tiles framework era anteriormente chamado *framework* de componentes, *Tiles* é um *framework* open *source* para fazer a camada de apresentação de trabalho muito mais fácil pela eliminação de grande quantidade de retrabalho e códigos repetidos. São muito úteis quando uma aparência comum entre todas as páginas são necessários, faz a manutenção do código muito fácil e uma separação de distribuição de conteúdo.

Tiles baseia–se na <include> oferecida pela especificação *JavaServer Pages*, assim, deixando–o mais viável criar páginas reutilizáveis. Ela ajuda a fornecer um *framework* completo e robusto para a montagem de páginas de apresentação de componentes de peças. Cada parte ("*Tile*") pode ser reutilizado quantas vezes for necessário em toda a sua aplicação. Isto reduz a quantidade de marcação que deve ser mantida e torna mais fácil mudar a aparência de um *website*. Pode–se dizer que *Tiles* são como componentes visuais.

Um *layout* é uma página *JSP* especial que permite que os pedaços de páginas sejam inseridos. A estrutura usa um arquivo de configuração XML para organizar as peças. E o *framework* não só permite a reutilização de peças, mas também os esquemas que organizá–los.

5.6 *Cas*e Porto Bello

Conforme relatado anteriormente o objetivo geral deste trabalho é implementar uma ferramenta de hotelaria utilizando uma arquitetura em três camadas e os padrões de projeto (design patterns), facilitando o controle das operações básicas de um hotel (hospedagem e reservas), utilizando um design sofisticado, com as opções bem distribuídas e práticas para uso diário, através de uma interface simples, possibilitando a automatização de todo o serviço de gerenciamento, atendendo as necessidades do cliente e permitindo que os usuários tenham acesso às informações e sejam mais produtivos no trabalho.

Ao analisar este objetivo e separar as funções específicas onde o projeto de

inteface atuaria constatou–se dois desafios:

- Do lado do sistema: Na arquitetura *MVC* (*Model View Controler*) o *View* deve ser bem estruturado (seguindo os padrões W3C) separando o código do conteúdo com o código do *layout* para facilitar o desenvolvimento e manutenção do sistema.
- O lado do usuário: Desenvolver uma interface com um layout sofisticado, simples e intuitivo que possibilite ao usuário a facilidade de utilização do sistema e ganho de produção no trabalho executado.

A documentação do sistema (relatório de entrevistas com o cliente, documentos de requisitos, diagramas UML) foi de grande importância para o sucesso do entendimento do problema.

Depois de várias análises foi possível concluiu–se que era necessário criar um protótipo estático. As ferramentas utilizadas no desenvolvimento do layout foram o *CorelDraw* (*Corel*) e *Photoshop* (*Adobe*). Somente após finalizar o protótipo estático em sua terceira versão o protótipo dinâmico foi iniciado e concluído utilizando o editores Dreamweaver (Adobe) e Eclipse.

A primeira tela de acesso ao Sistema de Gerenciamento de Hotelaria Porto Bello é a de *login* que executa a autenticação do usuário (ver Figura 59).

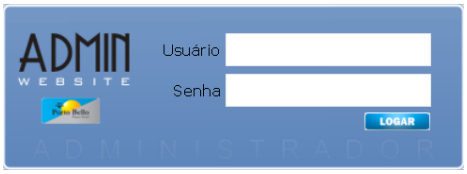

Figura 65 – Tela de *login*

Após a validação da autenticação o sistema redireciona o usuário para a tela de abertura (Home) do site. Nota–se que na Figura 60 o usuário Administrador foi autenticado com sucesso.

Figura 66 – Tela de *Home*

O sistema está estruturado e dividido em quatro áreas: topo, navegação, conteúdo e rodapé.

O Topo contém (localizado a esquerda na parte superior) a imagem da identidade visual da empresa. Seguindo a direita encontra–se o título do módulo e o nome do usuário autenticado. Abaixo há uma imagem (montagem da sala de recepção e um quarto) do hotel. Para navegação foi escolhido um menu tipo barra (por ocupar pouco espaço na tela e estar em posição de destaque) que direciona os usuários para as áreas de interesse: apartamento, funcionário, hóspede, reserva/hospedagem, mapa de acomodações, usuário e fechar.

Na área do conteúdo todas as páginas contém um título que destaca das informações acessadas através menu e submenu do sistema e para exibição das listas foi utilizada a biblioteca DisplayTag. Ícones foram inseridos nas páginas para também ajudar com a intuição e ganho de produtividade do usuário na execução das funções do sistema. O rodapé contém as informações de direitos autorais da empresa e a empresa desenvolvedora do sistema.

O topo, navegação e rodapé são conteúdos estáticos e somente o conteúdo é dinâmico. Estão codificados separadamente e são montados dinamicamente através do *framework Tiles*.

Os Quadros 52, 53, 54 mostram o código do topo, menu e rodapé respectivamente. O menu foi elaborado sem o uso de tabelas para diagramação do conteúdo conforme especificação do W3C *Standard (utilização de lista para organização dos itens do menu)*.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<c:url var="img" value="/img/"/>

<table WIDTH="770" HEIGHT="42" BORDER="0"
  CELLPADDING="0" CELLSPACING="0">
  <tr HEIGHT="100%">
    <td WIDTH="9"></td>
    <td WIDTH="175"><img SRC="${img}topo1.gif"></td>
    <td WIDTH="2"></td>
    <td WIDTH="390" CLASS="destaque_blue">
      <span CLASS="style2">
        M&oacute;dulo Admin</span>istrativo</td>
    <td WIDTH="37"><img SRC="${img}user.gif"></td>
    <td WIDTH="157" CLASS="normal">Administrador</td></tr>
</table>
<table WIDTH="770" HEIGHT="94" BORDER="0"
  CELLPADDING="0" CELLSPACING="0">
  <tr HEIGHT="100%">
    <td WIDTH="100%"><img SRC="${img}topo2.gif"></td></tr></table>
```

Quadro 64 – Código do arquivo header.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<c:url var="jsp" value="/jsp/"/>
<c:url var="img" value="/img/"/>

<ul>
  <li><a HREF="${jsp}apartamentoForm.do">
    <span>Apartamento</span></a></li>
  <li><a HREF="${jsp}funcionarioForm.do">
    <span>Funcion&aacute;rio</span></a></li>
  <li><a HREF="${jsp}hospedeForm.do">
    <span>H&oacute;spede</span></a></li>
  <li><a HREF="${jsp}reservaHospedagemForm.do">
    <span>Res/Hospedagem</span></a></li>
  <li><a HREF="${jsp}mapaAcomocacoesForm.do">
    <span>Mapa de Acomodações</span></a></li>
  <li><a HREF="${jsp}usuarioForm.do">
    <span>Usuário</span></a></li>
  <li><a CLASS="right" HREF="#">
    <span>Fechar </span></a></li></ul>
```

Quadro 65 – Código do arquivo menu.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>

<span CLASS="rodape"><br/>
  &copy; 2007 &bull;  Porto Bello Palace Hotel &bull; Powered by
  <a HREF="#">SIG5 Solutions</a></span>
```

Quadro 66 – Código do arquivo footer.jsp

```
01 <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
02     pageEncoding="ISO-8859-1"%>
03
04 <%@ taglib uri="http://struts.apache.org/tags-tiles-el" prefix="tiles"%>
05 <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
06
07 <c:url var="css" value="/css/"/>
08 <c:url var="img" value="/img/"/>
09 <c:url var="jsp" value="/jsp/"/>
10
11 <html>
12   <head>
13     <title>:: PORTO BELLO PALACE HOTEL ::</title>
14     <style type="text/css" media="screen">
15       @import "${css}header.css";</style>
16     <style type="text/css" media="screen">
17       @import "${css}toolbar.css";</style>
18     <style type="text/css" media="screen">
19       @import "${css}body.css";</style>
20     <style type="text/css" media="screen">
21       @import "${css}footer.css";</style>
22   </head>
23   <body STYLE="margin-top: 10px; margin-bottom: 0px;
24     margin-left: 0px; margin-right: 0px;">
25     <div ID="TableBody">
26       <div ID="header_img">
27         <tiles:insert attribute="header"/>
28       </div>
29       <div ID="vista_toolbar">
30         <tiles:insert attribute="menu"/>
31       </div>
32       <div  ID="body_page">
33         <table BORDER="0" CELLPADDING="0" CELLSPACING="0" >
34           <tr >
35             <td WIDTH="12" BACKGROUND="${img}body_left.gif"></td>
36             <td WIDTH="745" VALIGN="TOP" >
37               <span CLASS="destaque_blue">
38             <br></span>
39             <tiles:insert attribute="body"/>
40             </td>
41             <td WIDTH="13" BACKGROUND="${img}bory_rigth.gif"></td>
42           </tr>
43           </table>
44       </div>
45       <div ID="body_page_down">
46         <table WIDTH="100%" HEIGHT="100%" BORDER="0" CELLPADDING="0"
47             CELLSPACING="0">
48         <tr>
49           <td WIDTH="12" BACKGROUND="${img}body_left_down.gif"></td>
50           <td WIDTH="745" BACKGROUND="${img}body_center_down.png"></td>
51           <td WIDTH="13" BACKGROUND="${img}body_rigth_down.gif"></td>
52         </tr>
53         </table>
54       </div>
55       <div ID="footer">
56         <tiles:insert attribute="footer"/>
57       </div>
58     </div>
59   </body></html>
```

Quadro 67 – Código do arquivo layout.jsp

O Quadro 55 mostra o código do *template* layout.jsp, esta página contém *tags*

Tiles de inserção para montagem das partes do sistema. Também contém *links* para inclusão de css externo (header.css, toolbar.css, body.css e footer.css).

5.7 Navegação do site Porto Bello

A Figura 61 representa a navegação do sistema que conforme dito anteriormente é acessada através da barra de navegação após feita a autenticação.

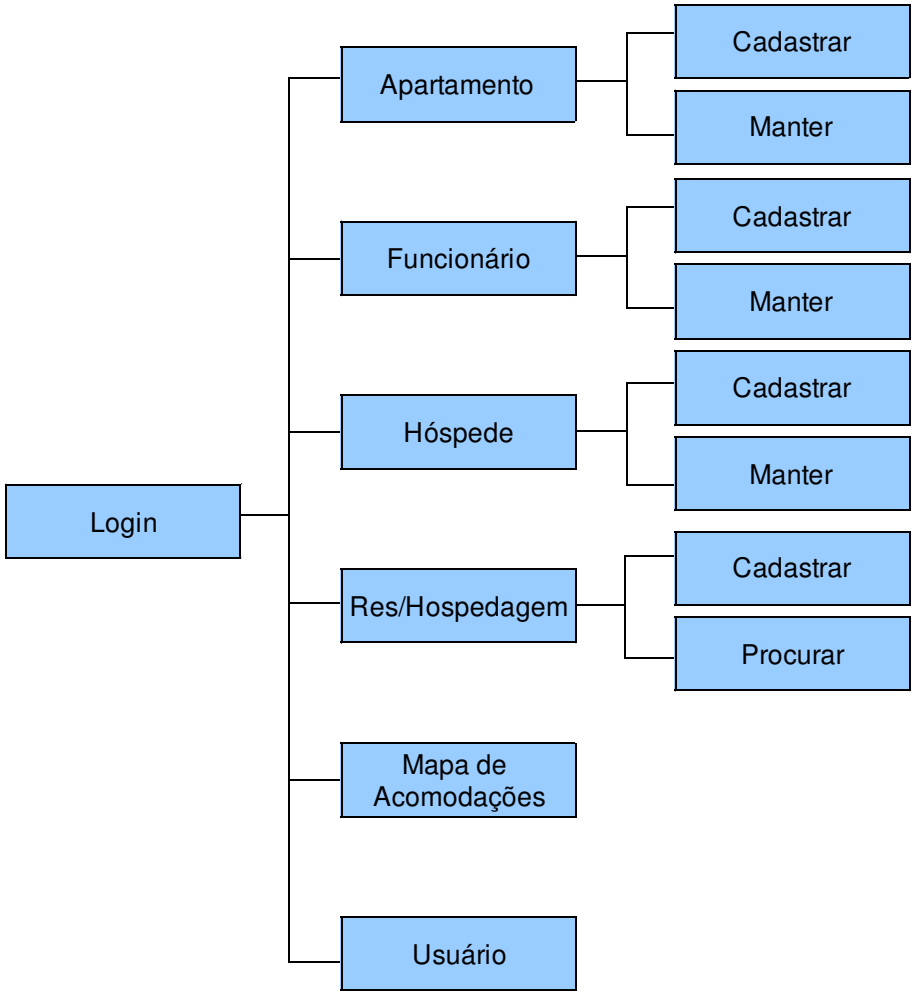

Figura 67 – Diagrama de Navegação do sistema

A Figura 68 mostra a tela de cadastro de cadastro de hóspede.

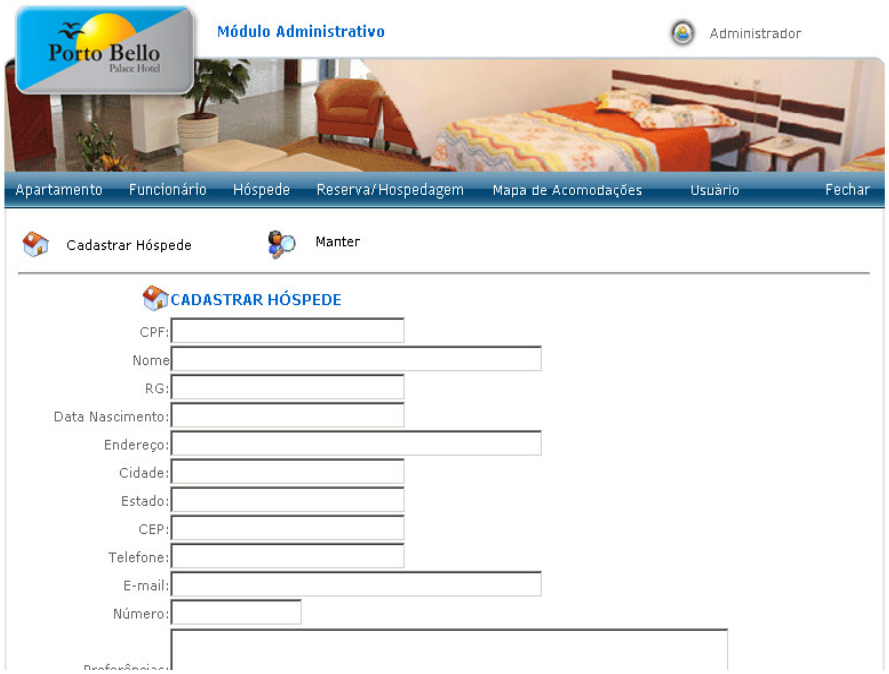

Figura 68 – Tela de Cadastro de Hóspede

É possível visualizar na Figura 63 a tela de exportação dos dados dos apartamentos em planilha excel.

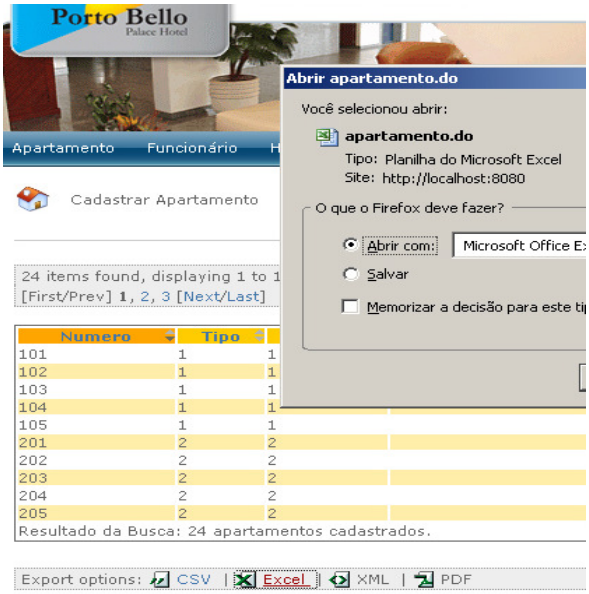

Figura 69 – Tela de Exportação de dados para planilha Excel

Com a recomendação *W3C* para arquitetura *web* (estrutura, apresentação e comportamento separados e complementares) foi possível planejar um projeto de interface bem estruturado e executá–lo em conjunto com as outras áreas do sistema. Com esse padrão implementado, o trabalho de alteração do *layout* do site fica mais fácil, pois os desenvolvedores somente precisarão atualizar os arquivos CSS e o *template* que o código do conteúdo não será alterado.

6 PERSISTÊNCIA DE DADOS

6.1 Introdução

Bancos de Dados (ou bases de dados) são conjuntos de registros estruturados que normalmente agrupam registros utilizáveis para um mesmo fim. Esses dados são organizados de forma a se reorganizar para produzir informação.

Um Banco de Dados é normalmente gerenciado por um conjunto de *softwares* conhecidos como Sistema Gerenciador de Banco de Dados (SGBD). O objetivo principal de um SGBD é o gerenciamento de base de dados retirando da aplicação cliente a responsabilidade de controle ao acesso e a manipulação dos dados, mas permitindo que os mesmo possam incluir, alterar, excluir ou consultar os registros na base de dados contidos.

Os Bancos de dados podem ser classificados quanto ao seu modelo de organização estrutural. O modelo mais adotado nos dias de hoje é o Modelo Relacional, que consiste em uma organização em forma de tabelas compostas por linhas e colunas construindo um Banco de Dados.

Os Bancos de Dados são utilizados em diversos tipos de aplicações, abrangendo todo tipo de *softwares* que necessitam de um armazenamento de dados. Os Bancos de Dados são a forma mais utilizada de armazenamento e produção de informação. Uma BD utiliza tecnologias padronizadas de armazenamento.

Um Banco de Dados pode ou não ser armazenado de forma legível para um computador. Um Banco de Dados pode ter um tamanho bem reduzido, sendo armazenado em arquivos únicos de pequeno porte e possuindo apenas algumas tabelas. Mas também pode ter milhares de tabelas com milhões de registros que ocupem dimensões físicas extremamente grandes.

Os primeiros Bancos de dados considerados modernos, como os que dispomos nos dias de hoje, foram concebidos por Charles Bachman na década de 1960.

6.2 Sistema de Gerenciamento Porto Bello

O empenho em analisar e acompanhar as preferências de armazenamento assume importante posição na escolha da tecnologia a ser usada, no tipo de organização estrutural e na criação das estruturas que atendem as necessidades da aplicação.

No projeto do Banco de Dados da nossa aplicação tem como objetivo armazenar os dados referentes ao gerenciamento de reservas de quartos e a hospedagem. Também foram criadas estruturas para armazenar informações referentes ao funcionário, ao hóspede e ao apartamento, que são informações necessárias para realizar tanto a reserva quanto a hospedagem.

Foram criadas tabelas que identificam com clareza a necessidade de armazenamento de dados do nosso cliente, sendo assim seus dados persistem em uma estrutura externa a aplicação, ficando assim protegidos contra falhas na operação da aplicação ou mesmo falhas da aplicação.

6.3 SGBD – Sistema de Gerenciamento de Bancos de Dados

O SGBD escolhido para ser utilizado no desenvolvimento e na manutenção do nosso Banco de Dados é o MySQL que é um sistema que utiliza a linguagem SQL (Linguagem de Consulta Estruturada) como interface. O MySql é o SGDB mais utilizado atualmente.

O MySql foi criado pela empresa MySql AB e os responsáveis pelo seu surgimento foram David Axmark, Allan Larsson e Michael Widenius, no entanto a equipe de desenvolvimento e manutenção é de cerca de quatrocentas profissionais e mais mil colaboradores. O *software* é gratuito e pode ser facilmente encontrado para download na Internet.

Uma das principais características do SGBD é a possibilidade de colaboração de várias pessoas ao redor do mundo que contribuem testando o *software*, integrando–o a outros produtos, escrevendo a respeito dele e alterando o *software*,

criando assim novas distribuições que são compartilhadas entre os usuários.

No início de 2008 a empresa Sun Microsystems comprou a empresa MySql AB pelo valor de 1 Bilhão de dólares, sendo este o maior valor já pago no setor de licenças livres. Em Abril de 2009 a empresa Oracle comprou a Sun Microsystems e todos os seus produtos inclusive o MySql.

Dentre alguns usuários do MySql estão: NASA, Friendster, Banco Bradesco, Dataprev, HP, Nokia, Sony, Lufthansa, U.S Army, US. Federal Reserve Bank, Associated Press, Alcatel, Slashdot, Cisco Systems e CanaVialis S.A.

As principais características do SGBD MySql são: controle de transições e a utilização de *Stored Procedures*, *Triggers* e Funções. Essa quatro características contribuem para a popularidade do *software*. Alem disso outro ponto que ajuda na disseminação do MySql é o fato de ser um software de código livre.

6.4 Arquitetura do Banco de Dados

A idéia geral de um Banco de Dados é simplesmente armazenar os dados de um determinado sistema a fim de recuperá–lo posteriomente, altera–lo quando necessário e principalmente gerar informação através do cruzamento e interpretação dos dados nele contidos. No entanto nem sempre o usuário do sistema tem idéia de como esse armazenamento é feito fisicamente no disco.

Para que fosse representada tal abstração do armazenamento dos dados foram definidos três níveis conceituais de organização: o nível físico, o nível conceitual e o nível de visualização. No nível físico são definidas as configurações do Banco de Dados, tais como localização dos arquivos em disco e suas configurações. O nível conceitual é onde são definidas as características dos dados, tais como seu tipo, suas restrições e seus relacionamentos. E o nível de visualização é onde o usuário do sistema terá acesso aos dados contidos no banco abstraindo assim todo conceito e configuração definida nos níveis anteriores.

6.5 Modelador de Dados CA Erwin

O Modelador CA Erwin foi à ferramenta escolhida para fazer a modelagem do diagrama de entidade e relacionamentos (DER) do nosso Banco de Dados. O CA Erwin, também conhecido somente por Erwin pertence ao conjunto de programas de apoio ao desenvolvimento de *software* normalmente chamados de Ferramentas Case.

A ferramenta é de extrema facilidade de uso e permite ao desenvolvedor especificar os dados e tabelas envolvidas e seus relacionamentos. Através da ferramenta é possível gerar scripts de criação de bancos bem com o processo inverso, ou seja, a engenharia reversa. Além disso, a ferramenta permite também a criação dos mecanismos de sincronização de dados necessários. Todos esses pontos a tornam um dos *softwares* mais utilizados na área de modelagem de dados.

6.6 Diagrama de Entidades e Relacionamentos (DER)

Diagrama de entidade e relacionamento é um modelo diagramático o que descreve conceitualmente o Modelo de Entidades e Relacionamentos. Sua principal função é representar e um alto nível de abstração as tabelas, os atributos e as relações existentes no Banco de Dados.

Abaixo, a figura 61 representando o DER do Sistema de Gerenciamento Porto Bello.

Figura 70 – DER

6.7 Tabelas

O Banco de Dados desenvolvido possui 8 tabelas, sendo que estas serão descritas a seguir.

- **Tabela pessoa**: é responsável em manter os dados referentes a todas as pessoas que tem o cadastro efetuado pelo sistema ou que tenham acesso ao sistema. Sua chave primária é o atributo id_pessoa_cpf, que apesar de ter o prefixo “id” o campo é uma chave natural. Todas as pessoas cadastradas no sistema possuem um usuário para acesso, por isso foi criada um *foreign key* da tabela usuário.
- **Tabela funcionário**: é uma especialização da tabela pessoas. A tabela funcionário guarda os dados referentes aos funcionários da empresa, tanto os que utilizam o sistema quanto os funcionários que não o utilizam. Sua chave primária é o atributo id_pessoa_funcionario que apesar de tero prefixo “id” o campo é uma chave natural. A tabela funcinário possui uma *foreign key* com cardinalidade um para um chamada id_pessoa_cpf, no entanto devido modelador do Diagrama de Entidades e Relacionamentos e ao tipo de relacionamento não foi possível representar graficamente, mas na construção do Banco de Dados foi inserido.
- **Tabela usuários**: é a tabela na qual serão gravados os dados referentes aos funcionários que possuem acesso ao sistema. Os dados dessa tabela são utilizados no *login* e conseqüentemente no controle de acesso aos casos de uso do sistema. Sua chave primária é o atributo id_usuario, que apesar de identificar o usuário não é usado para a realização do login. A tabela possui um *foreign key* chamada id_grupo referente a tabela grupo.
- **Tabela grupo**: para a implementação do controle de acesso do sistema foi concebida a idéia de criar uma tabela que serviria de parâmetro para a separação entre os diferentes níveis de acesso. Essa tabela possui apenas um identificador e uma descrição, através desse identificador se define o acesso que o usuário terá no sistema. Sua chave primária é o atributo id_grupo.
- **Tabela hospedes**: é a tabela que persiste os dados referentes ao hóspedes

do hotel. É a tabelas mais acessadas do sistema, pois além de possuir telas de inserção, edição, exclusão e de visualização, ainda está presente nos principais casos de uso do sistema, que são o Cadastro de Reserva, *Chek In* e *Check Out*. Sua chave primária é o atributo id_pessoa_hospede. A tabela hospede possui uma *foreign key* referenciando a tabela pessoa. Devido ao tipo de relacionamento entre as tabelas pessoa e hospede não foi possível representar a *foreign key* graficamente.

- **Tabela apartamento**: é responsável por guardar os dados referentes aos apartamentos existentes no hotel. Seu atributo chave é id_apartamento_num, que apesar de ter o prefixo “id” o campo é uma chave natural. Sua chave primária é o atributo id_apartamento_num. O atributo id_tipo_apartemento é uma foreign key referenciando a tabela tipo_apartamento.
- **Tabela tipo_apartamento**: é a tabela onde ficaram definidos os tipos de apartamentos que existem no hotel. Sua chave primária é o atributo id_tipo_apartamento.
- **Tabela reserva_hospedagem**: é o coração da aplicação. As principais funções do sistema têm por base essa tabela. A tabela reserva_hospedagem registra todas as informações referentes às reservas e hospedagens realizadas no hotel. A diferenciação entre os dois tipos de registro é feita através da uma análise aos campos persistidos, caso apenas os campos de data dt_entrada e dt_saida estejam populados o registro se refere a uma reserva, caso além desses dois campos o campo dt_check_in estiver preenchido quer dizer que o registro se refere a um hospedagem iniciada, e caso além desses três campos o campo dt_check_out também estiver preenchido o registro se refere ao uma hospedagem concluída. Sua chave primária é composta e é formada pelos atributos id_res_hospedagem e id_pessoa_cpf.

6.8 Hibernate

Hibernate é um *framework* escrito na linguagem Java para realizar o

mapeamento objeto–relacional, ou seja, faz a ligação entre uma classe de um programa Java a uma tabela do Banco de Dados, e faz a correlação entre uma variável de uma classe Java ao uma coluna ou atributo de uma tabela no Banco de Dados.

A principal característica do *Hibernate* é reduzir a complexidade na relação entre um programa Java e um Banco de Dados.

6.8.1 Arquivo hibernate.cfg.xml

Para realização do mapeamento entre a classe Java e a tabela na Base de Dados, é escrito um arquivo de configuração chamado hibernate.cfg.xml que será mostrado no Quadro 56.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE hibernate-configuration (View Source for full doctype...)>
<hibernate-configuration>
<session-factory>
<property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
<property name="hibernate.connection.password">lucas</property>
<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/portobello</property>
<property name="hibernate.connection.username">root</property>
<property name="hibernate.dialect">org.hibernate.dialect.DB2Dialect</property>
<mapping resource="bello/porto/dao/Funcionario.hbm.xml" />
<mapping resource="bello/porto/dao/Grupo.hbm.xml" />
<mapping resource="bello/porto/dao/Hospede.hbm.xml" />
<mapping resource="bello/porto/dao/Usuario.hbm.xml" />
<mapping resource="bello/porto/dao/ReservaHospedagem.hbm.xml" />
<mapping resource="bello/porto/dao/Apartamento.hbm.xml" />
<mapping resource="bello/porto/dao/Pessoa.hbm.xml" />
<mapping resource="bello/porto/dao/TipoApartamento.hbm.xml" />
</session-factory>
</hibernate-configuration>
```

Quadro 68 – Arquivo de Configuração hibernate.cfg.xml

A configuração realizada no arquivo hibernate.cfg.xml é apenas o mapeamento entre classe e tabela, para que se mapeie as variáveis das classe com as colunas do Banco de Dados é criado um arquivo *.hbm.xml. São criados vários arquivos de configuração entre variável e coluna, sendo que cada arquivo é rsponsável por configurar as variáveis de uma classe a uma tabela, sendo necessárias então um arquivo por mapeamentos desejados. Por exemplo para mapear a tabela funcionários criasse o arquivo funcionários.hbm.xml. Segue abaixo

a representação desse arquivo.

```
<?xml version="1.0" ?>
<!DOCTYPE hibernate-mapping (View Source for full doctype...)>
<!--  Generated 04/12/2007 23:42:19 by Hibernate Tools 3.2.0.beta8
-->
<hibernate-mapping default-cascade="none" default-access="property"
                   default-lazy="true" auto-import="true">

<class name="bello.porto.beans.Apartamento" table="apartamento"
       catalog="portobello" mutable="true" polymorphism="implicit"
       dynamic-update="false" dynamic-insert="false"
       select-before-update="false" optimistic-lock="version">

<id name="idApartamentoNum" type="int">
  <column name="id_apartamento_num" />
  <generator class="assigned" />
</id>

<property name="tipoApartamento" type="int" unique="false"
          optimistic-lock="true" lazy="false" generated="never">
      <column name="id_tipo_apartamento" not-null="true" />
</property>

<property name="numAndar" type="int" unique="false"
          optimistic-lock="true" lazy="false" generated="never">
      <column name="num_andar" not-null="true" />
</property>

<property name="desApartamento" type="string" unique="false"
          optimistic-lock="true" lazy="false" generated="never">
      <column name="des_apartamento" not-null="true" />
</property>

</class>

</hibernate-mapping>
```

Quadro 69 – hbm.xml

Nesse capítulo foi demonstrado como foi planejada a parte de persistência de dados do Sistema de Gerenciamento Porto Bello, desde a escolha das tecnologias e frameworks até a configuração entre a aplicação e a Base de Dados.

Após todo o mapeamento realizado pelos arquivos de configuração basta apenas informar à classe que se deseja fazer acesso e através de uma classe DAO (Data Access Object). Nessa classe estão escritos os métodos responsáveis pela manipulação dos dados.

7 CONCLUSÕES

O crescimento da utilização de sistemas *web* está em ritmo acelerado o que torna imprescindível a adoção de tais sistemas em ambientes onde a velocidade e a adaptabilidade são fatores fundamentais.

Analisando as necessidades do cliente, as suas exigências e também o ambiente onde o sistema será implantado, então foram escolhidas as ferramentas, as metodologias, os frameworks e a linguagem que atende melhor o cliente.

Através da análise foi possível planejar um sistema ágil e confiável e que possui a possibilidade de qualquer computador da rede do hotel, necessitando apenas ter acesso à Internet.

Foi adquirido conhecimento em *frameworks Struts* e *Hibernate*, do padrão MVC, que foram utilizados nesse sistema. A integração dos *frameworks* juntamente com os padrões de projeto resultou na arquitetura em si, estruturando um fluxo bem definido das requisições cliente–servidor.

O resultado que realmente marca esse projeto, é sem duvida o aprendizado, a experiência e a visão do mercado de trabalho, é perceptível o amadurecimento técnico e gerencial agregado, e a junção dos esforços durante essa maravilhosa empreitada foi transformado em uma aplicação, e documentação que satisfez as especificações e requisitos solicitados, superando as expectativas iniciais.

Os resultados obtidos são fruto da dedicação e comprometido de toda equipe envolvida no planejamento, sendo que cada integrante foi responsável por uma parte do projeto, escolhendo a metodologia, as ferramentas e trabalhando na representação da idéia em forma de formulários, documentos e diagramas tornando assim a concepção da idéia por atores externos ao planejamento, fácil e desprovida de entendimento ambíguo.

As experiências adquiridas pelos integrantes da equipe foram de grande valia para a utilização no mundo coorporativo, pontos como o trabalho em equipe, o comprometimento com prazos e principalmente a experiência na parte de analise e projeto de sistemas voltados para web utilizando a plataforma J2EE. Portanto o

principal resultado obtido durante esse projeto foi o crescimento profissional de toda equipe que hoje está apta a desenvolver atividades semelhantes fora do ambiente acadêmico.

8 REFERÊNCIAS

ALUR D.; CRUPI J.; MALKS D. **Core J2EE Patterns**:as melhores práticas e estratégias de design. Elsevier Editora Ltda: 2004.

ALVIM, P. **Tirando o Máximo do J2EE com jCompany**. São Paulo, 2006.150p.

ANGOTI, E. **Desenvolvimento de Sistemas I**. Uberlândia: Faculdade de Ciências Aplicadas de Minas. Apostila. Disponível em: <http://si.uniminas.br/~angoti/arquivos/tiles.pdf>. Acesso em: 13 out. 2009.

APACHE SOFTWARE FOUNDATION. **Tiles 2**. Forest Hill, MD, USA. 2009. Disponível em: <http://tiles.apache.org/> Acesso em: 13 out. 2009.

AVELAREDUARTE. **Localização dos elementos – interface web**. 2009. Disponível em: <http://www.avellareduarte.com.br/projeto/interface/interface1/interface1_composicao.htm>. Acesso em: 22 nov. 2009.

_____. **Componentes da interface web**. 2009. Disponível em: <http://www.avellareduarte.com.br/projeto/interface/interface8/interface8.htm>. Acesso em: 02 dez. 2009

BAUER C.; KING G. **Hibernate em ação**. Rio de Janeiro: Editora Ciência Moderna Ltda. 2005.

BACALÁ JÚNIOR, SILVIO. **Arquitetura de software baseada numa abordagem UML/Redes de Petri com prevenção de bloqueio mortal em sistemas de tempo real**. Mestrado. Uberlândia, 2003. 173p.

BODOFF S; ARMSTRONG E; BALL J; **Tutorial do J2EE Enterprise Edition 1.4**. Rio de Janeiro: Editora Ciência Moderna Ltda. 2002.

BORTOLASSI, Giuliano. **Uma breve introdução ao JEE**: conhecendo o JEE. Disponível em <http://jeenoob.blogspot.com/2007/01/conhecendo–o–jee–part1.html>. Acesso em: 01 jun. 2009.

DEITEL, H.M; DEITEL, P.J. **Java**: como programar. Tradução de Edson Furmankiewicz – São Paulo: Pearson Prentice Hall, 2005. 1110p

DIAS, C. **Usabilidade na web**: criando portais mais acessíveis. Rio de Janeiro: Alta Books, 2002. 312 p.

FOUNDATION, The Apache Software. **Welcome to Struts 1**. Disponível em <http://struts.apache.org/1.3.10/index.html>. Acesso em: 01 jun. 2009.

FRAGMENTAL Tecnologia. **MVC e Camadas**. Disponível em: <http://www.fragmental.com.br/wiki/index.php/MVC_e_Camadas>. Acesso em: 01 mai. 2009.

FURLAN, J.D. **Modelagem de Objetos a través da UML – Unified Modeling Language**. São Paulo: Makron Books, 1998. 329p.

Gaudel, M-C., Marre, B., Schilienger, F. Bernot, G. ***Précis de génie logiciel*.** Masson: 1996. 437p.

HOOK, David. ***Beginning Cryptography with Java***. Indianapolis: Wrox, 2005. 448p.

HUSTED T**. Struts em ação**. Rio de Janeiro: Editora Ciência Moderna Ltda. 2004.

IBRAU. **Ícones, grandes aliados do projetista**. 2006. Disponível em: <http://www.ibrau.com.br/artigoicones.htm>. Acesso em: 05 dez. 2009.

JAVA Community Process. ***Community Development of Java Technology Specifications***. Disponível: <http://jcp.org/aboutJava/communityprocess/pr/jsr244/>. Acesso em: 01 jun. 2009.

JUNIOR,E.; SEGUNDO,A. **Históorico dos Banco de Dados**. Eduardo Junior Alonso Segundo. Disponível em: <http://disciplinas.dcc.ufba.br/svn/MATA60/tarefa1/historico/historico.pdf?revision=2>. Acesso em 05 ago. 2009

KING, A. ***Clickstream Study Reveals Dynamic Web***. 2006. Washtenaw County, Michigan USA. Disponível em: <http://www.websiteoptimization.com/speed/tweak/clickstream/>. Acesso em: 03 dez. 2009

LEITE, J. C. **Projeto de Interfaces de usuário**. 2001. 10f. Tese (Mestrado) – Universidade Federal do rio Grande do Norte, Rio Grande do Norte. Disponível em: < http://www.dimap.ufrn.br/~jair/piu/JAI_Apostila.pdf>. Acesso em: 08 jun. 2008.

LINEBACK, N. ***Graphical User Interface Timeline***. 2006. Disponível em: <http://toastytech.com/guis/guitimeline.html>. Acesso em: 08 jun. 2008.

LINWOOD, J. ***Determine the best elements for Web site navigation***. 2003. Disponível em: <http://articles.techrepublic.com.com/5100–10878_11–1058652.html?tag=nl.e055>. Acesso em: 02 dez. 2009

LLOID,I. ***Accessible HTML/XHTML Forms***. Disponível em: <http://www.webstandards.org/learn/tutorials/accessible–forms/beginner/>. Acesso em: 04 dez. 2009.

MACORATTI, J.C. **Padrões de Projeto**: o modelo MVC – Model View Controller. Artigo disponível em <http://www.macoratti.net/vbn.htm> Acesso em: 01 jul. 2009.

MACROMEDIA DREAMWEAVER HELP. ***Dreamweaver Basics***: Intrudoction. 1997

MARKETSHARE. ***Screen Resolutions***. 2009. Disponível em: <http://marketshare.hitslink.com/report.aspx?qprid=17&qptimeframe=M&qpsp=129&qpct=3&qpmr=10>. Acesso em: 01 nov. 2009.

MCCLURG, J. ***Designing for the Web***. 2006. Disponível em: < http://www.digital–web.com/articles/designing_for_the_web/>. Acesso em: 02 nov. 2009.

NIELSEN, J. **Projetando websites**. Tradução de Ana Gibson – Rio de Janeiro: Enselvier, 2000 – 5ª reimpressão. 416p. Título origina: Designing web usability.

_____. **Breadcrumb Navigation Increasingly Useful**. Fremont CA USA. 2007. Disponível em: <http://www.useit.com/alertbox/breadcrumbs.html>. Acesso em: 03 dez. 2009.

_____. **Right–Justified Navigation Menus Impede Scannability**. Fremont CA USA. 2008. Disponível em: <http://www.useit.com/alertbox/navigation–menu–alignment.html>. Acesso em: 02 nov. 2009.

_____. **Screen Resolution and Page Layout**. Fremont CA USA. 2006. Disponível em: <http://www.useit.com/alertbox/screen_resolution.html>. Acesso em: 02 nov. 2009.

_____. **Scrolling and Scrollbars**. Fremont CA USA. 2005. Disponível em: <http://www.useit.com/alertbox/20050711.html>. Acesso em: 03 dez. 2009.

SANTOS, E. **Web Standard**. 62 p. Slide. São Paulo: Campus Party. Disponível em: <http://www.agni.art.br/palestras/web–standards–no–campus–party>. Acesso em: 24 ago. 2009.

SCHNEIER, Bruce**. Applied Cryptography**. New York: John Wiley and Sons, 1996. 758p.

SCIARRETTA, T. **Clientes de baixa renda já são maioria nas compras on–line**. São Paulo: Folha online. 2007. Disponível em: <http://www1.folha.uol.com.br/folha/dinheiro/ult91u353177.shtml>. Acesso em: 03 jun. 2008.

SHACHOR, G; CHACE, A; RYDIN,M; **Java Server Pages**: biblioteca de tags. Tradução de Rejane Freitas – Rio de Janeiro: Editora Ciência Moderna Ltda, 2002. 594p.

SCHNEIDER, G. WINTERS, J. P., ***Applying Use Cases a Practical Guide***. Ed. Addison Wesley. USA 1998.

SILVA, M.S. **Construindo formulários HTML/XHTML acessíveis**. 2006. Disponível em: <http://maujor.com/tutorial/formac–a.php>. Acesso em: 04 dez. 2009.

SOFTECH NETWORK. **Treinamento Java**. Disponível em: <http://java.danieldestro.com.br/>. Acesso em: 01 jul. 2009.

SPOLSKY, J. **Projeto de Interface com Usuário para Programadores**. 2000. Disponível em: <http://brazil.joelonsoftware.com/uibook/chapters/1.html> Acesso em: 17 ago. 2009.

SOMMERVILLE, Ian. **Engenharia de Software.** Tradução de Maurício de Andrade. 6. ed. São Paulo: Person Education do Brasil, 2003. 591 p. Título Original: Software Engineering.

W3SCHOOLS. **HTML Introduction**. Disponível em: <http://www.w3schools.com/html/html_intro.asp> Acesso em: 27 ago. 2009.

_____. **CSS Introduction**. Disponível em: <http://www.w3schools.com/css/css_intro.asp> Acesso em: 30 set. 2009.

_____. **Introduction to XML**. Disponível em: <http://www.w3schools.com/xml/xml_whatis.asp> Acesso em: 27 ago. 2009.

_____. **JavaScript Introduction**. Disponível em: <http://www.w3schools.com/js/js_intro.asp> Acesso em: 05 dez. 2009.

_____. **XHTML Introduction**. Disponível em: <http://www.w3schools.com/xhtml/xhtml_intro.asp> Acesso em: 27 ago. 2009.

WEB STANDARDS PROJECT. **WaSP**: Fighting for Standards. Disponível em: <http://www.webstandards.org/about/mission/>. Acesso em: 24 ago. 2009.

WIKI,ECLIPCE. **Where did Eclipse come from?**. 2004. Disponível em: <http://wiki.eclipse.org/FAQ_Where_did_Eclipse_come_from%3F> Acesso em: 13 set. 2009.

ZELDMAN, J. **Projetando Web Sites compatíveis**: como construir web sites compatíveis com browsers e dispositivos variados. Tradução de Altair Dias Caldas de Moraes – Rio de Janeiro: Enselvier, 2003. 412p.