UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

CAMPUS PONTA GROSSA

DEPARTAMENTO DE PÓS-GRADUAÇÃO

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE PRODUÇÃO

PPGEP

ROGÉRIO RANTHUM

MODELAGEM E IMPLEMENTAÇÃO DE UM SISTEMA DE INFORMAÇÃO PARA OTIMIZAÇÃO DE EXAMES DE DIAGNÓSTICOS POR IMAGENS

PONTA GROSSA

DEZEMBRO - 2005

ROGÉRIO RANTHUM

MODELAGEM E IMPLEMENTAÇÃO DE UM SISTEMA DE INFORMAÇÃO PARA OTIMIZAÇÃO DE EXAMES DE DIAGNÓSTICOS POR IMAGENS

Dissertação apresentada como requisito parcial à obtenção do título de Mestre em Engenharia de Produção, do Programa de Pós-Graduação em Engenharia de Produção, Área de Concentração: Gestão Industrial, do Departamento de Pesquisa e Pós-Graduação, do Campus Ponta Grossa da UTFPR.

Orientador: Prof. Antonio Carlos de Francisco, Dr.

PONTA GROSSA

DEZEMBRO - 2005

R213 Ranthum, Rogério
Modelagem e implementação de um sistema de informação para otimização de exames de diagnósticos por imagens. / Rogério Ranthum. -- Ponta Grossa : UTFPR, Campus Ponta Grossa, 2006.
115 f. : il. ; 30 cm

Orientador: Prof. Dr. Antônio Carlos de Francisco

Dissertação (Mestrado em Engenharia da Produção) - Universidade Tecnológica Federal do Paraná, Campus Ponta Grossa. Curso de Pós-Graduação em Engenharia de Produção. Ponta Grossa, 2006.

1. Sistemas de Informação. 2. Software livre. 3. Software - Modelagem 4. Telemedicina. 5. Diagnósticos por imagens - Medicina. 6. DICOM (Digital Imaging and Comunications in Medicine). I. Francisco, Antônio Carlos de. II. Universidade Tecnológica Federal do Paraná, Campus Ponta Grossa. III.Título.

CDD 006.6

TERMO DE APROVAÇÃO

ROGÉRIO RANTHUM

MODELAGEM E IMPLEMENTAÇÃO DE UM SISTEMA DE INFORMAÇÃO PARA OTIMIZAÇÃO DE EXAMES DE DIAGNÓSTICOS POR IMAGENS

Dissertação de Mestrado aprovada como requisito parcial à obtenção do grau de Mestre em Engenharia de Produção, do Programa de Pós-Graduação em Engenharia de Produção, Área de Concentração: Gestão Industrial, do Departamento de Pesquisa e Pós-Graduação, do Campus Ponta Grossa da UTFPR, pela seguinte banca examinadora:

Orientador: Prof. Antonio Carlos de Francisco, Dr. - UTFPR

Profa. Maria Salete Marcon Gomes Vaz, Dra. - UEPG

Prof. Luciano Scandelari, Dr. - UTFPR

Prof. João Luiz Kovaleski, Dr. - UTFPR

Ponta Grossa, 13 de dezembro de 2005.

Ao meu filho João Manoel, fonte
inesgotável de alegria, luz e vida.

AGRADECIMENTOS

Ao meu orientador Prof. Antonio Carlos de Francisco, Dr., pelo apoio, orientação e por se dispor a enfrentar este desafio;

A Clínica Sabedotti, em especial ao Dr. Antonio Pedro Sabedotti, que viabilizou a elaboração deste estudo e o desenvolvimento do software;

Aos professores João Luiz Kovaleski, Dr., e Luiz Alberto Pilatti, Dr., responsáveis pelo meu ingresso no mestrado;

A minha família, pelo apoio e compreensão nos momentos mais difíceis;

Aos estagiários, alunos da UTFPR, que participaram e estão participando do projeto de desenvolvimento do sistema;

A todos os amigos que me apoiaram nesta difícil tarefa;

Aos professores Flávio Madalosso Vieira e Lourival Aparecido Góis, pelas suas orientações que foram fundamentais na elaboração deste trabalho;

E em especial, a minha irmã Marly que, sem sua ajuda jamais teria conseguido vencer esta etapa.

RESUMO

Os sistemas de informações têm desempenhado papel fundamental no contexto geral das organizações, seja na agilização de processos produtivos, na organização e arquivamento de documentos e até mesmo na tomada de decisão. A medicina também está inserida neste cenário. Analisando tais fatores este trabalho tem o como objetivo geral apresentar um modelo de para a implementação de um Sistema de Informação para otimização de exames de diagnósticos por imagens. Tal modelo foi desenvolvido utilizando os mais recentes conceitos de produção de software disponíveis no mercado mundial, como: desenvolvimento em 3 camadas, sistemas distribuídos, entre outros. Na busca de um produto voltado para a internet e interoperável, o sistema foi estruturado em camadas, aplicando-se o conceito de Modelo-Visão-Controle, que fornece a flexibilidade necessária ao desenvolvimento, e torna a manutenção do sistema mais organizada e estruturada, pois os módulos são independentes. O sistema foi desenvolvido utilizando ferramentas de software livre. O estudo foi desenvolvido em uma clínica de radiodiagnóstico por imagens da região de Ponta Grossa – PR. O grande desafio era a integração dos equipamentos de diagnóstico por imagens com a rede de computadores, isto foi realizado através do protocolo DICOM (*Digital Imaging And Comunications In Medicine)*, o mesmo tem a finalidade de padronizar a transmissão e recepção de imagens médicas. O resultado obtido foi a criação de um produto portável, flexível e de fácil adaptação, foi conseguido também união das informações clínicas dos pacientes como: nome, idade, endereço etc, com suas imagens médicas,. Outro fator importante é a redução dos custos operacionais das clínicas, pois não há a necessidade da impressão do filme (*filmless*).

Palavras-chave: Sistemas de Informação, Telemedicina, DICOM, Software Livre e Modelagem, Modelo-Visão-Controle.

ABSTRACT

Information systems have played an important role in organizations, at productive processes, at documents organization and even at decision-making; medical area is also in this scenery. This work aims to present a model to implement an Information Systems in order to optimize exams of diagnosis by images. The development of this model used the most modern software production like: three layers development, distributed systems, and others. Trying to develop a product for Internet and inter-workable, the system was structured in layers, applying the Model-Vision-Control concept that allows the necessary flexibility for development, and makes the maintenance more organized and structured. Free software was used to develop the system. The study was done in an images diagnostic radiology clinic in Ponta Grossa region, Paraná state. The challenge was to integrate images diagnosis equipments with computer network; this was done through DICOM protocol (Digital Imaging and Communications in Medicine), which standardizes transmission and reception of medical images. The result was a portable, flexible and adaptable product that joins all clients' information and reduces operational costs, because it is not necessary to print film.

Keywords: Information System, Telemedicine, DICOM, Open Source, Model, Model-View-Controller

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

3-TIER	Três Camadas
ACR-NEMA	*American College Of Radiology-National Eletrical Manufactures*
API	*Application Programming Interface*
ASCII	*Americam Standard CodeI*
CASE	*Computer Aided Systems Engineering* - Engenharia de Sistemas Auxiliado por Computador)
CDP	Centro de Processamento de Dados
DAO	*Data Acess Objec*t
DICOM	*Digital Imaging And Comunications In Medicine*
DTO	*Data Transfer Object*
EJB	*Enterprise Java Beans*
E-VIEW BOX	*E-View Box Project*
GIF	*Graphics Interchange Format* - Formato para Intercâmbio de Gráficos
HTML	*Hepertext Makup Language*
HTTP	*Hipertext Transfer Protocol*
J2EE	*Java 2 Enterprise Edition*
JDT	*Java Dicom Toolkit*
JPEG	*Joint Photographic Experts Group*
JSP	*Java Server Pages*
JVM	*Java Virtual Machine*
MVC	*Model-View-Control*
OID	*Object Indentification Data*
OJB	*Object-Relational Bridge*
OOA	*Object Oriented Analizing*
OOAD	*Object Oriented Analizing & Design*
OOD	*Object Oriented Design*
OOP	*Object Oriented Programming*
PACS	*Picture Achirve System*
PEP	Prontuário Eletrônico Do Paciente
PNG	*Portable Network Graphics*
ROM	*Relation Object Model*
SCP	*Service Class Provider*
SCU	*Service Class User*
SGML	*Standard General Markup Language.*
SI	Sistemas de Informação
SQL	*Structured Query Language*
SSL	*Secure Socket Layer*
TCP/IP	*Tranfer Control Protocol/Internet Protocol*
TI	Tecnologia da Informação
UML	*Unified Modeling Language*
URL	*Uniform Resource Locator. Localizado*
UTFPR	Universidade Tecnológica Federal do Paraná
VO	*Value Object*
W3C	*World Wide Web Consortium*
XML	*Extensible Makup Language*

XSL *Extendible Stylesheet Language*

SUMÁRIO

Capítulo 1

1. Introdução

O progresso das tecnologias de informação e de comunicação tem gerado grandes melhorias desde o fim do século XIX, como a microinformática que popularizou o uso do computador, pela redução de custos, pelas transmissões de dados através de linhas telefônicas, pela difusão da internet e também pela miniaturização de componentes eletrônicos que culminou com a criação de dispositivos móveis (*palmtop, notebook* e telefones celulares). Os limites da ciência ampliaram-se surgindo novas áreas do conhecimento, que estão afetando o dia-a-dia das pessoas.

Segundo Garvin (2000), essas transformações têm sido sentidas nas diversas profissões que, além do domínio específico, defrontam novos desafios no exercício das atividades profissionais.

Com o mercado globalizado e o advento da Internet, a concorrência entre as empresas acirra-se, onde a qualidade é entendida não apenas como uma oportunidade de melhorias na produção, mas torna-se uma revolução no pensamento administrativo (KIM, 1996).

Normalmente, toda empresa está inserida em um ambiente do qual não pode ser dissociada. Este ambiente vem ganhando complexidade nos últimos anos principalmente devido a questões de dimensão organizacional, ou tecnológica, ou demográfica ou de competitividade (CHILD e SMITH, 1987).

Nestes novos ambientes, os sistemas de informação e as tecnologias de comunicação, Internet, Intranet, têm se apresentado necessárias, mas não suficientes, para interligar comunidades de práticas afins, permitindo a interligação de pessoas a pessoas, pessoas a máquinas e máquinas a máquinas, (TERRA, 2001). Nonaka e Takeuchi (1997) mencionam que novas formas de registro e transmissão de conhecimento têm surgido e sido aplicado às empresas.

A informação serve à tomada de decisão, logo a necessidade de decidir com maior precisão é justificada pela necessidade que se tem em agir, dentro das

organizações e no campo da pesquisa. Com informações consistentes, tem-se melhor condição de decisão e também pode-se ‘vender’ mais eficazmente essas decisões perante o nosso ‘público-alvo’ .

A informação vem acrescentar ganho de capital, o qual varia conforme a estratégia escolhida e adotada pela empresa. A importância da informação pode ser aumentada a ponto de transformá-la, por vezes, no próprio centro da atividade da empresa, tornando-se, portanto: muitas vezes um produto desmaterializado. (MANÃS, 1999)

Quando o assunto versa sobre a informática na medicina, pode-se observar que ela tem ser tornado ponto fundamental no apoio ao gerenciamento do grande número de informações. As imagens geradas em equipamento de diagnósticos tem sido de grande utilidade na confecção de laudos precisos na busca do tratamento mais eficiente para os diversos casos, mas em contrapartida o volume de filme e documentos gerados tem dificultado o gerenciamento e armazenamento destes documentos.

Desta forma sistemas que atendam apenas as áreas administrativas das clínicas de imagens tornam-se obsoletos e sem muita utilidade. Isto leva as clínicas de imagens buscar o desenvolvimento de novas ferramentas de software que possam contribuir para a redução de gastos com armazenamento de dados. Assim, a questão que motivou a realização deste trabalho foi: **Como a modelagem e a implementação de um Sistema de Informação pode otimizar exames de diagnósticos por imagens?**

1.1. Objetivos

1.1.1. Tema

Gestão da Informação e do Conhecimento nas Organizações

1.1.2 Objetivo Geral

Desenvolver e implementar um software para otimização de exames de diagnósticos por imagens.

1.1.3 Objetivos Específicos

- Identificar principais tecnologias para implementação e modelagem de um Sistema de Informação para diagnóstico por imagens;
- Modelar e implementar um Sistema de Informação para diagnóstico por imagens;

1.1.4 Delimitação do Estudo

O estudo de caso em questão limita-se aos seguintes itens:
- Empresa: Clínica médica de radiodiagnósticos por imagens
- Tamanho: empresa de pequeno porte
- Atuação: radiodiagnóstico por imagens
- Foco do trabalho: Módulo operacional
- Concentração: otimização do processo produtivo.

A figura 1 ilustra o escopo do trabalho:

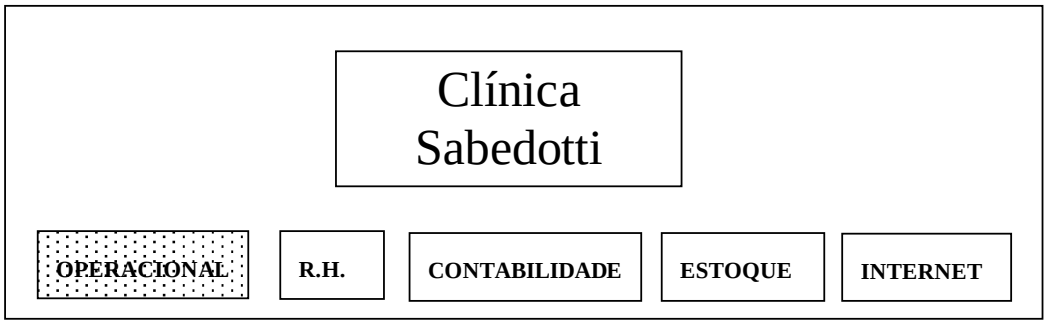

Figura 1 Escopo do Trabalho
Fonte: Organograma Funcional da Clínica Sabedotti.

No projeto de desenvolvimento e implantação de um novo sistema de gerenciamento informatizado para a Clínica Sabedotti, analisou às funcionalidades de cinco módulos principais, como ilustrado na Figura 1, porém por motivos de tempo e complexidade este trabalho está focado apenas no módulo Operaiconal.

1.2 Justificativa

Atualmente existem padrões de armazenamento de dados médicos que permitem a estruturação e manipulação adequada dos prontuários eletrônicos de pacientes que possibilitam a interoperabilidade de informações. (ALEXANDRINI, 2005), porém a maioria dos Sistemas de Informações implementados nas clínicas de radiodiagnóstico por imagens são obsoletos e sem perspectivas de evolução.

Para (Cartitá, 2004) “A implantação de um serviço de radiologia “*filmless*” tem resultado em grande número de avanços operacionais, incluindo melhoria no gerenciamento das imagens e leitura mais rápida, possibilitando acessos quase que em tempo real, eliminação de certo número de passos no processo de disponibilização das imagens, eliminação de exames perdidos e melhoria na produtividade do trabalho em grupo”.

O custo com a impressão de filmes desnecessária é elevada, pois a maior parte dos exames não apresenta nenhum tipo de anomalia. Desta forma os pacientes não retornam a clínica para a retirada de seus exames.

Para que haja uma segunda opinião sobre determinado exame é necessário que o filme seja impresso e enviado através do correio ou outro meio qualquer, dificultando e atrasando o tempo de emissão do laudo para o paciente.

1.3 Estrutura do Trabalho

O estudo está organizado e seqüenciado em 6 capítulos. No Capítulo 1, são apresentados os objetivos e justificativas do trabalho. No Capítulo 2, é feita uma revisão bibliográfica sobre Gestão da Informação, Sistemas de Informação e Sistemas de Gestão da Informação na área médica, os principais padrões utilizados na área médica e apresentados os conceitos de telemedicina.

No Capítulo 3 são descritas as tecnologias empregadas na modelagem e implementação de um Sistema de Informação para a área médica e são apresentados também conceitos sobre software livre, Linguagem Java e a Arquitetura J2EE.

No Capítulo 4 são descritos os procedimentos metodológicos aplicados ns pesquisa e como foi modelado o Sistema de Informação.

O Capítulo 5 está estruturado com a descrição de como foi feita a implementação do software e quais as principais dificuldades encontradas.

E por fim, no sexto capítulo são apresentados os resultados obtidos com a modelagem e implementação do novo Sistema de Tecnologia da Informação para diagnóstico por imagens, as conclusões do trabalho e são apresentadas recomendações e sugestões de temas e questões para futuros trabalhos.

Capítulo 2

2. GESTÃO DA INFORMAÇÃO

O Termo gestão da informação assume, atualmente, grande importância no mundo contemporâneo, o qual tem sido utilizado para indicar o modo globalizante de administrar as diversas informações, sejam essas nas áreas econômica, política ou cultural.

De acordo com Rodrigues (2003, p.1), o conceito de gestão da informação surgiu nos anos 70 – denominado por muitos também de Tecnologia da Informação – portanto, ‘é o termo genérico para todas as formas de se processar, arquivar, recuperar, classificar, organizar, usar dados, transformando-os, por tudo dito antes, em informação”.

Para Graeml (2000, p18,) “o conjunto de tecnologias resultantes da utilização simultânea e integrada de informática e telecomunicações tem-se chamado tecnologia da informação”.

Laudon e Laudon (1999), afirmam que a tecnologia de comunicações é usada para conectar partes diferentes do hardware e para transferir dados de um ponto a outro, via redes. Uma rede liga dois ou mais computadores entre si para transmitr voz, dados imagens, sons e vídeo ou para compartilhar recursos tais como uma impressora. A tecnologia de comunicações consiste em meios físicos e software que suportam a comunicação via meios eletrônicos.

A gestão para este novo tempo exige o máximo possível de informações para a tomada de decisões. O conceito de informação deriva do latim e significa um processo de comunicação, ou seja, alguma coisa relacionada à comunicação, sobretudo “informação é um processo que visa ao conhecimento, ou, mais simplesmente, informação é tudo que reduz a incerteza.”

Para que a gestão de informação seja eficaz, “é necessário que se estabeleça um conjunto de políticas coerentes que possibilitem o fornecimento de informação relevante, com qualidade suficiente, precisa, transmitida para o local certo e no tempo correto” (REIS, 2004, p. 90)

À medida que torna mais eficiente o conhecimento e a articulação entre os segmentos que constituem a instituição, também apóia os gestores na tomada de decisão. A tecnologia digital é o instrumento que permite gerir a informação, utilizando-se como mais uma alternativa, objetivando a agilização das informações e tornando a sua transmissão mais eficiente e, assim, facilitar a participação, a interação e a tomada de decisão.

A velocidade de transmissão da informação em redes de computadores, indiferente se pelo uso da fibra ótica, por satélite ou por rádios digitais, possibilita acessar e interagir de diversas maneiras de acordo com as condições de cada usuário. Observa-se que cada vez mais as tecnologias proporcionam uma riqueza de variedades de acesso à informação.

2.1 TECNOLOGIA DA INFORMAÇÃO

Os avanços da computação e de outras formas de tecnologias estão presentes na vida das organizações. É difícil encontrar qualquer forma de organização ou de processo organizacional que não tenha sido alterado pelas novas tecnologias. (MARTINELI, 2001, p.22)

Para Chiavenato (2000, apud Martinelli), a tecnologia da informação invade a vida das organizações, provocando profundas transformações. Ela permite a compressão do espaço, uma vez que trouxe o conceito de escritório virtual ou não territorial. Prédios e escritórios sofreram uma brutal redução em tamanho. Os arquivos eletrônicos acabaram com o papel e com a necessidade de móveis, liberando espaço para outras finalidades; a fábrica otimizada foi decorrência da mesma idéia aplicada aos materiais em processamento e à inclusão dos fornecedores como parceiros no processo produtivo. Os centros de processamento de dados (CPDs) foram enxugados (*downsizing*) e descentralizados através de redes integradas de microcomputadores nas organizações. Surgiram as empresas virtuais conectadas eletronicamente, dispensando prédios e reduzindo despesas fixas que se tornaram desnecessárias.

A tecnologia da informação desempenha papel estratégico: “Ajudar o desenvolvimento do conhecimento coletivo, e do aprendizado contínuo, tornando mais fácil para as pessoas na organização compartilharem problemas, perspectivas, idéias e soluções”. (TEIXEIRA FILHO, 2003, p.3)

Segundo Gonçalves (1993), na sociedade industrializada, o progresso técnico apresenta pelo menos três metas básicas: a redução do esforço de trabalho, o aumento da produtividade e a melhoria da qualidade do produto.

Uma vez que a tecnologia é capaz de reduzir o tempo gasto para a realização de uma tarefa, uma análise muito pouco explorada é a discussão sobre a quem pertence o tempo do trabalhador liberado pela utilização de novas tecnologias. O autor salienta que é freqüente encontrar análises que tentam identificar se a tecnologia nova resulta em impactos positivos, ao permitir aos trabalhadores que anteriormente realizavam uma tarefa mecanizada, passar a desempenhar funções com um maior grau de esforço intelectual, a partir da reestruturação na organização do trabalho, uma vez que suas rotinas sejam automatizadas.

O ambiente da era da informação, tanto para as organizações do setor de produção quanto para as do setor de serviços, exige novas capacidades para assegurar o sucesso competitivo. A capacidade de mobilização e exploração dos ativos intangíveis ou invisíveis tornou-se muito mais decisiva do que investir e gerenciar ativos físicos tangíveis.

Kaplan e Norton (1997) ressaltam que as empresas da era da informação estão baseadas em um novo conjunto de premissas operacionais como:

> a) processos interfuncionais - as empresas buscam vantagens competitivas através da especialização de habilidades funcionais: nas áreas do conhecimento, administração, marketing e tecnologia;
> b) ligação com clientes e fornecedores - manter clientes e fornecedores a uma distância segura; a TI permite que as empresas de hoje integrem os processos de suprimentos, produção e entrega;
> c) segmentação de clientes - as empresas da era da informação devem aprender a oferecer produtos e serviços customizados aos seus diversos segmentos de clientes, sem serem penalizadas nos custos por operações de alta variedade e baixo volume;
> d) escalas globais extrapolando as fronteiras nacionais - as empresas da era da informação concorrem com as melhores empresas do mundo e devem combinar as eficiências e a agressividade competitiva do mercado global com a sensibilidade às expectativas dos clientes locais;
> e) inovação - os ciclos de vida dos produtos continuam diminuindo; a vantagem competitiva numa geração da vida de um produto não garante a liderança na próxima plataforma tecnológica;
> f) trabalhadores de conhecimento (*knowledge workers*) - as empresas criam fortes distinções entre dois grupos de funcionários: a elite intelectuais gerentes e engenheiros que utilizavam suas habilidades analíticas para projetar produtos e processos; e a força

> do trabalho direto que era o principal fator de produção nas empresas da era industrial.

Percebe-se que para que as empresas tenham competitividade e qualidade em seus serviços é necessária a utilização da tecnologia da informação como instrumento e que a mesma esteja presente através de departamentos bem estruturados e atualizados técnica e tecnologicamente.

2.2 SISTEMAS DE GESTÃO DA INFORMAÇÃO

A tecnologia da informação está redefinindo os fundamentos dos negócios. Atendimento ao cliente, operações, estratégias de produto e de marketing e distribuição dependem muito, ou às vezes até totalmente, dos SI. A tecnologia da informação e seus custos passaram a fazer parte integrante do dia-a-dia das empresas. (O'BRIEN , 2002, p.135)

Hirschheim (1995, p.11, apud Zanetti Junior, 2004) define Sistema de Informação das seguintes formas:

Um sistema de informação (SI) pode ser definido em termos de duas perspectivas: uma relacionada à sua função e outra à sua estrutura.

Da perspectiva estrutural, um SI "consiste em uma coleção de pessoas, processos, dados, modelos, tecnologia e linguagem parcialmente formalizada, formando uma estrutura coesa que serve a algum propósito ou função"

Da perspectiva funcional, um SI é "uma mídia tecnologicamente implementada para o propósito de gravar, armazenar e disseminar expressões lingüísticas assim como apoio ao desenvolvimento de inferências".

Ao executar estas funções básicas, os sistemas de informação "facilitam a criação e a troca de significados que servem a propósitos socialmente definidos tais como: controle, entendimento e argumentação (por exemplo, formulação e justificativa de reivindicações.

Pode-se notar que nas duas perspectivas de SI as pessoas estão incluídas nas fronteiras, o que significa que os "serviços proporcionados por um sistema de informação em parte dependem das capacidades e contribuições das pessoas" .

Em outras palavras, as pessoas têm um papel fundamental para permitir que os SI atinjam seus propósitos. (ZANETI JUNIOR, 2003)

Um sistema de informação (SI) pode ser definido como um conjunto de componentes inter-relacionados trabalhando juntos para coletar, recuperar, processar, armazenar e distribuir informação com a finalidade de facilitar o planejamento, o controle, a coordenação, a análise e o processo decisório em empresas e outras organizações. (LAUDON, 1999)

Os sistemas de informação contêm, entre outras, informações sobre pessoas, lugares e coisas de interesse do ambiente ao redor da organização e dentro da própria organização. Os sistemas de informação essencialmente transformam a informação em uma forma utilizável para a coordenação de gerentes a tomar decisões, analisar e visualizar assuntos complexos e resolver outros tipos de problemas.

2.3 SISTEMAS DE INFORMAÇÃO NAS ORGANIZAÇÕES

Sistemas de informação eficazes têm um papel importante nos negócios e na sociedade atual, podendo ter um grande impacto na estratégia corporativa e no sucesso organizacional das empresas. (LAUDON, 1995)

Convive-se permanentemente com um grande volume de dados disponibilizados através das tecnologias de informação. Verifica-se que tais dados necessitam cada vez mais de um tratamento prático e de bom senso que os transformem em INFORMAÇÃO pertinente. (DAVIS, 1989)

Os SI e a tecnologia da informação têm grande importância nas organizações atuais. Eles podem alterar os processos empresariais de várias formas. Algumas delas são, descritas a seguir por Alter (1996):

- Aumentando a capacidade das pessoas, através do fornecimento de informações, ferramentas e treinamento;

- Apoiando o trabalho de gerenciamento;

- Eliminando desperdícios, eliminando papéis desnecessários, reutilizando o trabalho (por exemplo, modelos de cartas), eliminando etapas de trabalho desnecessárias e atrasos, eliminando variações desnecessárias em procedimentos e sistemas e/ou eliminando atividades contraprodutivas;

- Estruturando o trabalho de forma a promover as melhores práticas: melhorando a manipulação de dados e o trabalho geral de escritório, apoiando o fluxo de trabalho e permitindo que o trabalho ocorra ininterruptamente;

- Automatizando as interfaces com os clientes, automatizando o trabalho de projetos e/ou automatizando a manufatura;

- Integrando através de funções e de organizações: ligando fornecedores e clientes através de troca eletrônica de dados, apoiando o processo de planejamento organizacional, colaborando no projeto de produtos e através de manufatura integrada por computador.

Para Davenport (1994), os Sistemas de Informação podem ser vistos como:

- Captando informação dos processos com o objetivo de compreensão;
- Melhorando a análise da informação e tomada de decisão;
- Substituindo ou reduzindo a mão de obra humana em um processo;
- Melhorando a coordenação entre tarefas;
- Modificando a seqüência de processo ou possibilitando o paralelismo;
- Permitindo a monitoração rigorosa da situação e objetos do processo;
- Permitindo a coordenação de processos à distância;
- Permitindo a eliminação de intermediários em um processo.

2.3.1 SISTEMAS DE INFORMAÇÕES NA ÁREA MÉDICA

A atividade hospitalar acompanha o início da civilização, desde os mercados da Babilônia, no Egito e Grécia antigos, na Índia (226 a.C.) e Ceilão (437 a.C.) (MAUDONNET , 1988).

Com a evolução da informática nos hospitais, nasceu o Prontuário Eletrônico do Paciente, visando a melhorar a eficiência e a organização do armazenamento de dados de saúde, com a promessa de não só substituir o prontuário em papel, mas também elevar a qualidade da assistência à saúde através de novos recursos e aplicações (MCDONALD ,1990).

Para algumas pessoas, um hospital é uma estrutura física preparada para atender pessoas doentes ou necessitadas de cuidados de saúde, através de médicos, enfermeiros e outros profissionais. Para outras, é uma organização de prestação de serviços em saúde, formada por profissionais do ramo, utilizando-se de uma estrutura física própria e de equipamentos e materiais médico-hospitalares.

Entretanto, como em todos os demais setores da economia, o hospitalar está também sendo questionado quanto à resposta de seu modelo produtivo às necessidades da sociedade.

O gerenciamento do estado de saúde de pacientes é uma área muito rica no que diz respeito à quantidade de informação envolvida. Nesta área, os profissionais da saúde, especialmente os médicos, devem lidar com uma grande quantidade de dados relacionados à informação médica dos pacientes. Considerando a complexidade do gerenciamento e sua importância humana e econômica, sistemas que possam assistir este trabalho podem ajudar esses profissionais a realizar seu trabalho de uma forma melhor e mais econômica. A maioria dos sistemas médicos considera o problema do diagnóstico de uma doença, mas o gerenciamento de um paciente abrange muito mais que isso. O diagnóstico representa menos de 5% do tempo do médico (ALTMAN , 1999).

A maioria das visitas ao médico está relacionada com a evolução de um problema previamente diagnosticado (LUCAS, 2004). Assim, os profissionais da saúde precisam de ajuda, especialmente para poder acompanhar os progressos de seus pacientes. Alguns sistemas foram construídos para assistir os profissionais da saúde com o gerenciamento e tratamento de pacientes, mas a maioria deles foi construída para uma doença particular (LUCAS , 2004), ou para tratar de uma tarefa específica dentre as várias tarefas associadas ao gerenciamento e tratamento de pacientes.(RIESCO, 2000)

Neste sentido, destacam-se os esforços no desenvolvimento dos chamados Prontuários Eletrônicos dos Pacientes (PEP). O PEP segundo (Waegemann, 1996), atual presidente do *Medical Record Institute*, é dividido em cinco níveis evolutivos:

> Registro Médico Automatizado: este nível de sistema representa a maioria dos casos na atualidade. A informação é armazenada em computadores pessoais e não está de acordo com os requisitos legais e, portanto, o prontuário em papel é mantido em conjunto. Desta forma, papel e registro eletrônico coexistem.
>
> Registro Médico Computadorizado: neste nível, médicos e toda a equipe coletam a informação no papel e a imagem dos documentos resultantes é armazenada de forma digitalizada no sistema computacional. Em geral, esse tipo de sistema é departamentalizado, com pouca documentação mas, já atinge alguns dos requisitos legais, podendo dispensar o papel em alguns casos.
>
> Registro Médico Eletrônico: consiste em um modelo interdepartamental, reunindo os requisitos legais para confidencialidade, segurança e integridade dos dados.
>
> Registro Eletrônico do Paciente: sistemas neste nível interligam todas as informações do paciente. Inclusive dados fora da instituição (interinstitucional). Para se chegar a este estágio, é necessária uma

maneira de identificar o paciente de forma unívoca e nacional.

Registro Eletrônico de Saúde: neste último nível, além das características evolutivas dos anteriores, a responsabilidade de manter o prontuário é dividida entre profissionais de saúde e paciente.

O registro de informações de saúde e de doença dos pacientes é a tarefa diária de todos aqueles que trabalham na área assistencial. O chamado Prontuário Médico ou do Paciente, ou ainda Registro Médico, é o agrupamento das anotações dessas informações. O prontuário em papel vem sendo usado há milhares de anos, já desde os tempos de Hipócrates, passando por diversas transformações ao longo do tempo, principalmente no último século quando se tornou mais sistematizado.

De acordo com o ministério da saúde (COSTA, 2001), um prontuário eletrônico do paciente pode ser descrito como:

- Um conjunto de documentos padronizados, ordenados e concisos, destinados ao registro dos cuidados médicos e paramédicos prestados ao paciente em um hospital ou clínica médica;
- Um conjunto de informações coletadas pelos médicos e outros profissionais de saúde que cuidaram de um paciente;
- Um registro com suporte à pesquisa clínica, estudos epidemiológicos, avaliações da qualidade do atendimento.

Para que as informações médicas dos pacientes possam ser armazenadas e recuperadas de forma rápida e organizada é necessário a utilização de Prontuários eletrônicos.

2.3.2 – Padrões para Sistemas de Informações na Área Médica

O *Digital Imaging and Communications in Medicine* (DICOM) é um padrão para comunicação e armazenamento de imagens médicas e informações associadas a elas. Atualmente o padrão DICOM é utilizado por diversas modalidades de equipamentos de geração de imagens médicas. Este padrão possui um mecanismo para troca de informações entre diversos tipos de imagens, assim como para a comunicação das mesmas.

O padrão foi desenvolvido por um comitê de trabalho do *American College of Radiology* (ACR) e do *National Electrical Manufactures Association* (NEMA) que

iniciou os trabalhos em 1983. Esse comitê foi formado com o intuito desenvolver um padrão para comunicação digital de informações de imagens, o comitê publicou a primeira versão em 1985, que foi chamada de ACR-NEMA 300-1985 ou ACR-NEMA *Version* 1.0 e a segunda versão em 1988, chamada de ACR-NEMA 300-1988 ou ACR-NEMA *Version* 2.0. A terceira versão do padrão, nomeada de DICOM 3.0 foi apresentada em 1993 que foi substancialmente enfatizado, o conteúdo alterado, discutindo, alguns problemas da primeira e segunda versão e criado um novo processo, principalmente o protocolo de comunicação para rede (SANTOS, 1996).

Os objetivos iniciais eram:

- Promover a comunicação de informações de imagens médicas digitais, sem levar em consideração os fabricantes de aparelhos;
- Facilitar o desenvolvimento e a expansão dos Sistemas PACS que também podem se comunicar com outros sistemas de informação hospitalar;
- Permitir a criação de uma base de dados para a geração de diagnósticos que possam ser examinados por uma grande variedade de aparelhos distribuídos geograficamente (ACR-NEMA, 2005).

O padrão hoje está essencialmente completo, apesar das mudanças que ainda possam acontecer devido à evolução da área, pois ele é um padrão multi-partes, significando que as informações podem ser acrescidas quando há necessidade. Como um padrão estável e perfeitamente desenvolvido, ele está sendo implementado por diversas companhias tecnológicas e produtora de imagens médicas.

O padrão DICOM facilita a capacidade de atividade conjunta de equipamentos de imagens médicas porque especifica:

- Um conjunto de protocolos a serem obedecidos pelos equipamentos exigindo a adaptação deles ao padrão;
- A sintaxe e semântica de comandos e informações associadas, as quais devem ser trocadas usando o protocolo;
- Informações que vem ser fornecidas com uma implementação para que a adaptação para o padrão seja cumprida.

O padrão não especifica:

- A implementação detalhada de algumas características do padrão sobre um equipamento que exige adaptação;
- O conjunto completo de características e funções esperadas implementado integrando um grupo de equipamentos, cada um exigindo adaptação DICOM;
- Um processo de teste e validação para avaliar uma adaptação aplicada ao padrão.

2.3.2.1 Objetivos do padrão DICOM

Entre os principais objetivos do padrão DICOM podem ser considerados (ACR-NEMA, 2005):

- Endereçar a semântica de comandos e dados associados. Para que equipamentos possam atuar uns sobre os outros, deve haver padrões de modo que equipamentos estejam esperando para reagir a comandos e dados associados;
- É explicito em determinar a adaptação necessária de implementações do padrão. Em particular, uma instrução de adaptação deve especificar informações suficientes para determinar as funções para que a necessidade de trabalho conjunta possa ser esperada com outro equipamento que também se ajuste ao padrão;
- Facilitar operações em ambiente de rede, sem a necessidade de um mecanismo de interface de rede;
- É estruturado para acomodar a introdução de novos serviços, facilitando assim suporte para futuras aplicações em imagens médicas;
- Faz uso de padrões internacionais existentes sempre que aplicável, e se adapta à documentação estabelecida para padrões internacionais.

O padrão vem sendo desenvolvido com uma ênfase em imagens médicas para diagnósticos geradas pela radiologia técnicas e a fins; de qualquer forma, ele é aplicável para uma grande linha de imagens relativas à troca de informações em ambientes médicos.

2.3.2.2 Suporte para rede DICOM

O padrão original ACR-NEMA definia uma simples interface paralela de 50 pinos, como meio de troca de mensagens. Isto limitava o padrão para operações ponto-a-ponto, com uma rede conectando pontos externos.

A ultima versão DICOM 3.0 está voltada para comunicação entre equipamentos, seja através de rede ou por ligações ponto a ponto. O propósito é que cada equipamento utilize seus próprios padrões e formatos para armazenar e gerenciar seus dados, mas quando for necessária a comunicação com outros equipamentos de diferentes fabricantes, é fundamental a existência de uma linguagem comum para que os equipamentos diversos fabricantes sejam capazes de se entender.

2.3.2.3 Vantagens do padrão DICOM

O padrão DICOM diferencia-se dos outros formatos de imagens tais como JPEG, PNG, GIF, e outros, por permitir que as informações dos pacientes sejam armazenadas, de forma estruturada, juntamente com a imagem, isto é, elas são armazenadas contendo ponteiros, conhecidos como *Tags* que identificam e limitam as informações. A imagem propriamente dita no Padrão DICOM é baseada no formato JPEG com ou sem compressão, dependendo do equipamento que a gerou, pois cada companhia de tecnologia em imagem pode implementar de uma forma, desde que obedeça à adaptação do padrão.

A grande vantagem dessa estrutura é permitir fazer a leitura do arquivo e extrair as informações necessárias para a comunicação direta, ou seja, gerenciar as imagens e informações dos pacientes de forma coerente, mantendo a integridade; outra vantagem é que ele possibilitou melhorar o desempenho e auxilia no desenvolvimento de Sistemas PACS (KIMURA, 12005).

2.3.3 Telemedicina

A Telemedicina tem como ponto de partida a utilidade do exercício médico, organizado e eficiente, à distância, através da transmissão de imagens estáticas, áudio, vídeo, e várias formas de informações. Utilizando-se para tanto redes de telecomunicações e seus componentes, tais como: cabos, fibras óticas, satélites, radio digital, Internet etc., tendo como objetivos a informação, o diagnóstico e o

tratamento de indivíduos isoladamente ou em grupo, desde que baseado em dados, documentos ou qualquer outro tipo de informação.

Para a Organização Mundial de Saúde, Telemedicina é a oferta de serviços ligados aos cuidados com a saúde, nos casos em que a distância é um fator crítico; tais serviços são providos por profissionais da área da saúde, usando tecnologias de informação e de comunicação para o intercâmbio de informações válidas para diagnósticos, prevenção e tratamento de doenças e a contínua educação de prestadores de serviços em saúde, assim como para fins de pesquisas e avaliações; tudo no interesse de melhorar a saúde das pessoas e de suas comunidades. (UNIFESP, 2005)

Telemedicina é oferta dos serviços de saúde por telecomunicação remota. Inclui consulta interativa e serviços de diagnóstico. (DECS, 2005).

Define-se também telemedicina como sendo o exercício da Medicina através da utilização de metodologias interativas de comunicação audiovisual e de dados, com o objetivo de assistência, educação e pesquisa em Saúde. (UNIFESP, 2005)

Atualmente a Telemedicina é encarada como uma forma de difundir cuidados na área da Saúde para localidades desprovidas dos mesmos, ou ainda, deficitárias de determinados tipos de procedimentos, com o objetivo amplo de permitir igualdade de acesso aos serviços médicos, independentemente da localização geográfica do indivíduo. Além desta importante atividade assistencial, o desenvolvimento da Telemedicina, em função do seu caráter interativo, possibilita a atuação nas áreas de ensino e pesquisa.

As vantagens da implantação de um sistema de Telemedicina são descritas a seguir: (UNIFESP, 2005)

- Acesso instantâneo à informação;
- Aumento da eficiência em todos os tipos de medicina;
- Personalização dos cuidados de saúde;
- Monitorização dos doentes crônicos;
- Aumento na eficácia do diagnóstico médico;
- Redução significativa do tempo de deslocação de doentes e médicos.

Como um dos objetivos deste trabalho é caracterizar as melhorias e otimizações inseridas no processo produtivo em clínicas de radiodiagnósticos por imagens, a telemedicina é responsável pela redução dos custos operacionais nestas clínicas e implementa em todo os sistema a melhoria e segurança nos diagnósticos através da segunda opinião de um médico que pode estar a quilômetros de distância, na agilização da disponibilização da informação e na redução de custos operacionais da clínica com a não impressão de filmes.

2.4. Considerações finais do capítulo:

A gestão da informação vem sendo utilizada em larga escala nas organizações, pois é através dela que as empresas podem baixar seus custos operacionais e agilizar seu atendimento. A área médica através de seus Sistemas de Informações tem procurado ao longo do tempo disponibilizar de uma forma mais rápida laudos clínicos de seus pacientes. Mas de forma ainda tímida e experimental, alguns esforços estão sendo feitos para agilizar o processo de resultados de exames aos pacientes.

No próximo capítulo serão apresentadas as tecnologias empregadas no desenvolvimento do modelo de um Sistema de Informação para clínicas de radiodiagnóstico por imagens, e como sua utilização otimizou os exames de diagnósticos por imagens.

Capítulo 3

3. Tecnologias

Os principais hospitais e clínicas do mundo dispõem de Sistemas de Informações que utilizam a mais variada gama de software. Normalmente, sistemas do ponto de vista técnico, caros e lentos.

O principal objetivo deste capítulo é descrever as principais tecnologias de desenvolvimento empregadas no desenvolvimento deste modelo, e como o uso de software livre pode baratear o desenvolvimento de um sistema robusto e eficiente.

3.1. Software Livre

Para Saleh (2004) “O software livre tem sido, nos últimos anos, objeto de atenção por parte dos profissionais e da imprensa especializada em informática. Na bibliografia sobre o assunto, é possível verificar que existem, e estão disponíveis, softwares livres com sofisticação suficiente para atender a grande parte das demandas empresariais. Verifica-se também que podem existir vantagens concretas em sua utilização, principalmente no que tange a redução de custos. No entanto a participação de mercado desses programas é muito restrita somente a nichos específicos”.

Enquanto os softwares proprietários têm um modelo de desenvolvimento fechado, onde apenas uma empresa ou indivíduo tem o controle sobre as funcionalidades, correções e melhoramentos, o software livre utiliza um modelo aberto, onde qualquer pessoa pode ter acesso ao código fonte e exercer o direito de livremente utilizar, redistribuir ou alterar o programa.

Durante muitos anos, o modelo proprietário, que trata o programa de computador como uma obra fechada e secreta, foi aceito como única forma possível de produção de software de qualidade, uma vez que o esforço do programador seria compensado pela venda de licenças de uso. No entanto, o modelo livre subverte essa ordem, fazendo que os produtos sejam compartilhados, de modo que os custos de seu desenvolvimento sejam divididos entre todos os interessados que os utilizam e desenvolvem.

Como esses programas são livremente distribuídos não é utilizado o conceito da venda de licenças, que é substituído por modelos de negócios que focam principalmente a prestação de serviços.

O sistema aqui descrito foi desenvolvido com base no sistema operacional Linux e visa à compatibilidade com os softwares livres disponíveis para ele e voltados para o desenvolvimento de software. A adoção do Linux e do software livre foi influenciada pelos diversos benefícios (técnicos, sociais e econômicos) normalmente associados ao software livre, em particular no Brasil (Silveira e Cassino, 2003), tais como:

> O software pode ser obtido e mantido atualizado por baixo custo, reduzindo custos em empresas e instituições de pesquisa possibilitando o investimento em outras áreas, como, por exemplo a compra de equipamentos.
>
> O software livre também pode apresentar uma grande economia na importação de software para o Brasil e, ao mesmo tempo ampliar o espaço para o desenvolvimento local do software; no caso de pesquisa em computação, software livres já disponíveis podem muitas vezes evitar a necessidade de desenvolvimento das partes de um sistema periférico ao objeto específico da pesquisa.
>
> O uso de software livre em instituições de pesquisa no Brasil ainda pode elevar o conhecimento brasileiro na área de computação, graças ao contato facilitado com código-fonte de alta qualidade e à troca de informações e experiências entre pesquisadores de diversos países atuando sobre softwares desenvolvidos de forma aberta.
>
> Tem se tornado claro que softwares livres são capazes de oferecer excelentes níveis de qualidade e desempenho, e o seu modelo de desenvolvimento colabora para uma grande velocidade na implementação de novos recursos aliada à manutenção de um baixo número de falhas de programação (*bugs*);
>
> O software livre, hoje, tem se configurado com uma das poucas alternativas viáveis ao monopólio atualmente existente no mercado de sistemas operacionais e aplicativos de escritório para microcomputadores; esse monopólio coloca a economia e o estado

> em situação fragilizada, em especial no caso de países em desenvolvimento como o Brasil;
>
> O fato de praticamente não haver restrições à redistribuição do software pode incentivar uma maior colaboração tanto entre programadores quanto entre usuários, facilitando o desenvolvimento e troca de experiências no uso de sistemas computacionais e, talvez, promovendo um espírito comunitário que pode ter reflexos benéficos na sociedade de maneira geral;
>
> Finalmente, o software livre tem sido apontado como peça fundamental no processo de inclusão digital das populações carentes, ou seja, nos processos de democratização do acesso aos sistemas computacionais e à informação disponibilizada através deles e da apropriação do universo digital enquanto ferramenta de trabalho e meio de comunicação individual e de massa por parte dessas populações.

Além disso, outras vantagens colaboraram para a escolha do Linux como plataforma básica para a estruturação de sistema voltados para a internet, tais como: (LAGO, 2005, p.5)

- dado ao interesse intrínseco no software livre discutida acima, é interessante promover o crescimento do uso do software livre em áreas como o desenvolvimento de sistemas médicos, pois isso pode colaborar para seu crescimento em outras áreas;

- as implementações dos principais programas relacionados (inclusive, por exemplo, o servidor de banco de dados) estão disponíveis sob licenças de software livre, possibilitando a alteração de algum desses programas caso necessário ou interessante;

- de forma similar, todas os padrões envolvidos são abertos e desenvolvidos pela comunidade de usuários; isso pode possibilitar a incorporação e adoção imediata das técnicas desenvolvidas na pesquisa, o que dificilmente

aconteceria no caso de especificações controladas por uma empresa não diretamente envolvida com o movimento de software livre;

- o sistema operacional em si é altamente portátil, além de ser amplamente compatível com os padrões POSIX[1], viabilizando a implantação dessas mesmas técnicas em outras plataformas de hardware, seja através do próprio Linux, seja através da adaptação para outros sistemas operacionais semelhantes ao UNIX.

O desenvolvimento do projeto foi motivado por alguns requisitos tecnológicos:

- seguir os protocolos padrão da Internet;
- apresentar independência de plataforma no cliente, bastando um navegador Web para acessá-lo;
- prover independência de Gerenciador de Banco de dados e do Servidor Web;
- ser baseado em software livre, diminuindo e viabilizando os custos do projeto;
- ser modularizado, permitindo que novas funcionalidades pudessem ser adicionadas no futuro.

Como a solução utiliza protocolos abertos, o acesso aos dados pode também ser disponibilizado para clientes, a partir de fornecedores que implementem esses protocolos. Outra consideração importante é a portabilidade da solução proposta. Qualquer um dos produtos envolvidos pode ser substituído sem alterar a estrutura geral do sistema.

3.2. Linguagem Java

A Linguagem Java for criada pelo grupo liderado por James Gosling na *Sun Microsystems*, e é uma linguagem computacional completa, independente de plataforma e com uma série de facilidades para a integração com a internet.

A Linguagem Java é de alto nível, orientada a objetos, simples, portável, de arquitetura neutra, distribuída, de alto desempenho, interpretada, que dá suporte a paralelismo e concorrência, com coletor de lixo, robusta, dinâmica e segura.

[1] POSIX é um conjunto de especificações para sistemas operacionais; seu objetivo é facilitar a transposição de aplicações entre diferentes sistemas operacionais compatíveis com essas especificações. As especificações são desenvolvidas por um grupo de trabalho do IEEE.

Os programas escritos em Java são executados por uma máquina virtual Java, ou Java VM (*Java Virtual Machine*), que interpreta o código, e é independente de plataforma.

3.2.1. Arquitetura J2EE

J2EE, ou Java 2 *Enterprise Edition*, é uma plataforma para desenvolvimento de aplicações distribuídas. Apresenta facilidades para a utilização dos recursos computacionais e distribuídos tais como acesso a banco de dados, componentes Web, utilização de mensagens assíncronas, execução de processos transacionais, persistentes ou não etc.

A arquitetura J2EE apresenta uma API, especificada pela *Sun MicroSystems*, que proporciona um padrão para a implementação dos diversos serviços que oferece, sendo que isto pode ser feito diferentemente por várias empresas, de formas distintas mas ainda assim oferecendo as mesmas facilidades, por estarem de acordo com as especificações impostas para a sua construção (BOND et al, 2003).

3.2.1.1. Visão da plataforma

A arquitetura J2EE se apresenta em várias camadas, sendo que cada uma é composta por componentes e serviços que são providos por um *container*.

Segundo Temple (2004), pode-se entender como *container* o próprio navegador que fornece recursos e facilidades para o componente, neste caso as páginas HTML. O componente por sua vez, pode oferecer diversos serviços ao usuário, através do suporte do *containers*, tais como facilidades visuais como botões, hiperlinks, figuras e tabelas, e o próprio serviço de navegação.

Em um servidor J2EE, podemos ter diversos *container's* interagindo entre si. A seguir uma breve explicação de cada camada da arquitetura e de seus componentes.

Camada cliente: acesso por meio de interfaces *standalone* (aplicações Java), páginas HTML ou *Applets*. Nesta camada os componentes residem em um *Applet Container*, em um HTML container *(Web browser*) ou em *um Application Client Container*.

Camada *Web*: esta camada é implementada por JSP's e *Servlets*, que fornecem a lógica para a camada cliente (ou de apresentação) do negócio. JSP's e

Servlets residem no *Web* Container. JSP's oferecem a facilidade de utilizar algumas lógicas de apresentação em uma página *Web* sem muitas dificuldades tecnológicas. O *Servlet* apresenta-se como um controlador das ações executadas pelos usuários nas páginas de apresentação, e fornece recursos para obter dados dessas ações e realizar as operações desejadas.

Camada de Negócios: esta camada trata da lógica de negócio da aplicação. É nela que se implementa todas as regras de negócio, alocação de recursos, persistência de dados, validação de dados, gerência de transações e segurança, providos por componentes conhecidos por EJBs.

Camada EIS - *Enterprise Information System*, ou Sistema de informações empresariais: nesta camada é que se encontram os sistemas de banco de dados, sistemas legados, e a integração com outros sistemas que não são do Padrão J2EE.

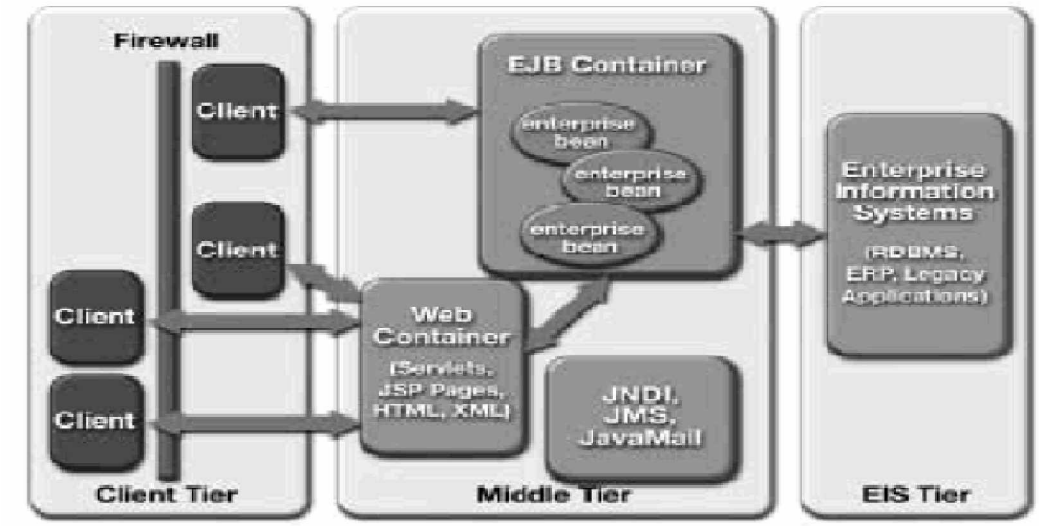

Figura 2 Camadas, componentes, c*ontainers* e sua interação.
Fonte: Braz (2003, p.1)

O desenvolvimento de aplicações em camadas torna o sistema mais complexo na fase de análise e desenvolvimento, mas quando se trata da manutenção os programadores de computador tem mais facilidades, pois cada camada tem sua independência, embora integradas, conforme está ilustrador na Figura 2.

3.3. Camadas da Aplicação

No mundo do J2EE, existem geralmente quatro camadas: um cliente, uma aplicação *Web* construída ou com *servlets* Java ou com *JavaServerPages* (JSP), uma camada de lógica de negócio construída em *Enterprise JavaBeans* (EJB) e uma camada de persistência que acessa um banco de dados relacional (DESIGN PATTERNS, 2003).

A Plataforma J2EE é basicamente um modelo de aplicação distribuído em multicamadas no qual a lógica da aplicação é dividida em componentes de acordo com sua função. Isto permite que vários componentes da aplicação que compõem uma aplicação J2EE sejam instalados em máquinas diferentes.

Os *patterns* a seguir propiciarão a melhoraria no desempenho de seus sistemas em multicamadas e irão torna-los mais fáceis de se manter ao reduzir a complexidade. Como todos eles estão relacionados ao particionamento da aplicação, esses são *patterns* essenciais para quase todas as aplicações J2EE e altamente relevantes para todos os desenvolvedores. O *pattern Model-View-Controller* (MVC) reforça um projeto modular e de fácil manutenção e força a separação de camadas. O session (EJB) organiza a lógica de negócio para o cliente. O *Data Access Object* (DAO) fornece uma interface flexível entre a lógica de negócio e as fontes de dados reais. Finalmente, o *Data Transfer Object* (DTO), também conhecido como um *value object*, facilita a comunicação entre as camadas.

3.4. Aplicações em Três Camadas

Neste modelo de três camadas, a lógica de apresentação está separada em sua própria camada lógica. A separação em camadas torna os sistemas mais flexíveis permitindo que as partes possam ser alteradas de forma independente. As funcionalidades da camada de negócio podem ser divididas em classes e essas classes podem ser agrupadas em pacotes ou componentes reduzindo as dependências entre as classes e pacotes; podem ser reutilizadas por diferentes partes do aplicativo e até por aplicativos diferentes. O modelo de três camadas tornou-se a arquitetura padrão para sistemas corporativos com base na WEB.

De acordo com Lautert e Oliveira (2004), a idéia do modelo MVC é facilitar as atividades de desenvolvimento de software, utilizando orientação a objetos e dividindo os sistemas em três camadas: modelo (ou camada de negócios), visão (ou

apresentação) e controle, conforme Figura 4. Basicamente, esta visão busca diminuir o tempo perdido com atividades repetidas em algumas etapas do desenvolvimento de sistemas semelhantes. Desta forma, obtém-se um ganho significativo em produtividade, há uma redução na quantidade de erros durante o processo de desenvolvimento e melhora a manutenibilidade e suporte dos sistemas, incrementando a qualidade do produto final.

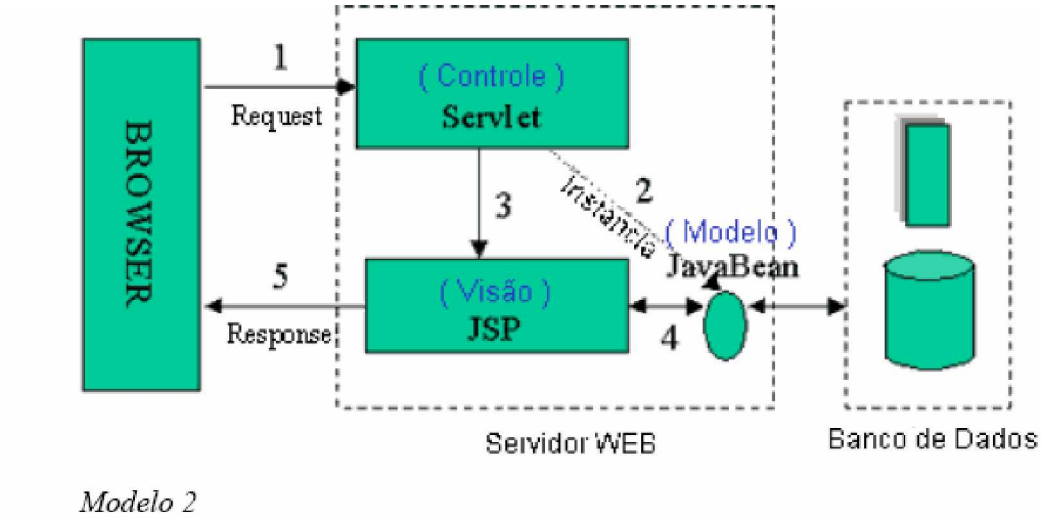

Figura 3 Divisão do modelo MVC (Modelo 2)
Fonte: Lautert e Oliveira (2004, p. 1).

Os autores descrevem o modelo de forma a reduzir o acoplamento entre as partes, e as dividem em:

- Camada de Controle: é a camada responsável pelo fluxo, qualidade e segurança das informações no sistema. É a ligação entre as camadas de negócio e apresentação;
- Camada de Modelo: são as atividades relativas ao próprio negócio do sistema, como classes de persistência que conversam com a base de dados, "*beans*" da própria modelagem do sistema e suas relações;
- Camada de Visão: são os formulários, relatórios - a interface em geral do sistema com os utilizadores. Elas mostram os resultados gerados na camada de modelo. Geralmente utilizam tecnologias como JSP, HTML ou XSL.

Na arquitetura MVC o modelo representa os dados da aplicação e as regras do negócio que governam o acesso e a modificação dos dados. O modelo mantém o

estado persistente do negócio e fornece ao controlador a capacidade de acessar as funcionalidades da aplicação encapsuladas pelo próprio modelo.

A tecnologia JSP foi projetada para ser flexível, mas a maneira como sua aplicação irá se comportar poderá ser realizada de diversas maneiras:

- Combinar livremente HTML e *scriplets* JSP.
- Combinar livremente HTML e *scriplets* JSP e delegar funcionalidades para *Servlets*.
- Utilizar *Servlets*, páginas JSP e *beans* (distribuídos ou não) para implementar uma arquitetura do tipo MVC.

A primeira abordagem, combinação de grandes quantidades de código Java com HTML em páginas JSP, conduz, freqüentemente, a aplicações difíceis de serem utilizadas, mantidas e estendidas e, assim, portanto, não é recomendada pela Sun.

Delegar funcionalidades para *beans* é uma abordagem viável, porque o código Java passa a estar centralizado em estrutura de dados possivelmente reutilizável, a partir do momento que o mesmo é retirado das páginas de scripts JSP. Esta arquitetura é comumente conhecida como “Model 1” da Sun (GEARY, 2002).

A última abordagem, a combinação de *Servlets*, páginas JSP e *beans* (classes de negócios da aplicação) em uma arquitetura MVC, resulta em um software extensível e sustentável, porque encapsula funcionalidades e reduzirá o impacto de futuras mudanças. Esta arquitetura de desenvolvimento, conhecida pela Sun como “Model 2” , é análogo ao padrão arquitetural MVC (GEARY, 2002).

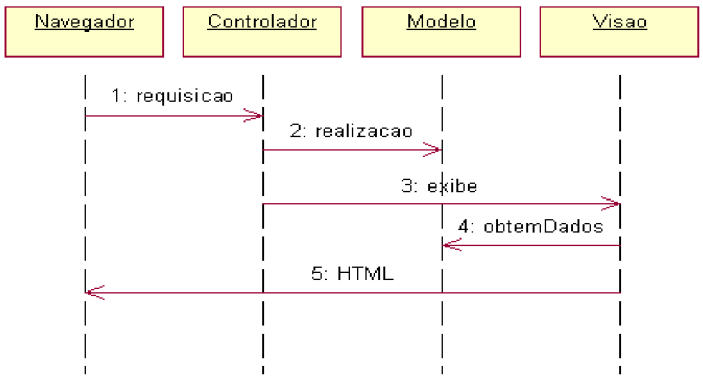

Figura 4 Modelo MVC para Aplicações Web

A Figura 4 mostra o diagrama de seqüência para o Padrão MVC no caso de um Cliente HTML. O Controlador recebe a requisição do Navegador e realiza a ação correspondente no Modelo. Em seguida, ele escolhe a Visão a ser exibida. A Visão obtém os dados necessários do modelo e a página HTML gerada é exibida no navegador.

3.5. Camada de Controle

3.5.1. *Struts*

Em uma definição breve, struts é um *framework* que implementa o modelo MVC. Explicando melhor, *JavaServlets* foram criados para lidar com os *requests* feitos pelos browsers. Páginas JSP foram criadas para criar conteúdo dinâmico para exibição. *Struts* é um *framework* que através de um *servlet* especial lê um arquivo XML de configuração, e assim envia os *requests* primeiramente a uma classe Java que tratará esse *request*, realizando toda a lógica do negócio, e logo após chama uma página JSP que tratará de exibir a página para o navegador. Desta maneira, aplicações WEB tornam-se mais fáceis de serem projetadas, criadas e especialmente, mantidas.

A sua principal característica é prover uma camada de controle flexível baseada em padrões de tecnologia já bem estabelecidos, *Struts*. Entre as tecnologias usadas pelo *framework*, estão: *Servlets*, J*avaBeans* e XML (*eXtensible Markup Language* - Linguagem de Marcação Extensível) (BITTENCOURT, 2004).

Segundo Souza (2004), o *Struts* provê a sua própria camada de controle, bastando que as classes da aplicação estendam a classe *org.apache.struts.action.Action* para que elas executem o processamento desejado. Para a configuração do fluxo da aplicação, deve-se construir um arquivo (*struts-config*) no formato XML definindo quais ações devem ser mapeadas para os objetos responsáveis. Essa facilidade permite que todo o fluxo do sistema permaneça separado do código-fonte.

Para auxiliar na criação das páginas JSP, existe a *taglib* que faz parte da distribuição do *framework*. As *tags* que compõem essa *taglib* possuem funções que permitem desde a criação de formulários até a renderização de erros provenientes da camada de controle.

Struts tem o objetivo de gerenciar uma aplicação. Para tanto, configura-se no descritor da aplicação (o arquivo *WEB-INF/web.xml*) que a classe *ActionServlet* do *struts* irá receber as requisições do browser através do comando /do/. Nesse mesmo arquivo configura-se um ou mais arquivos que servirão de configuração ao *struts*. Na Figura 5, é representado o funcionamento do *struts.*

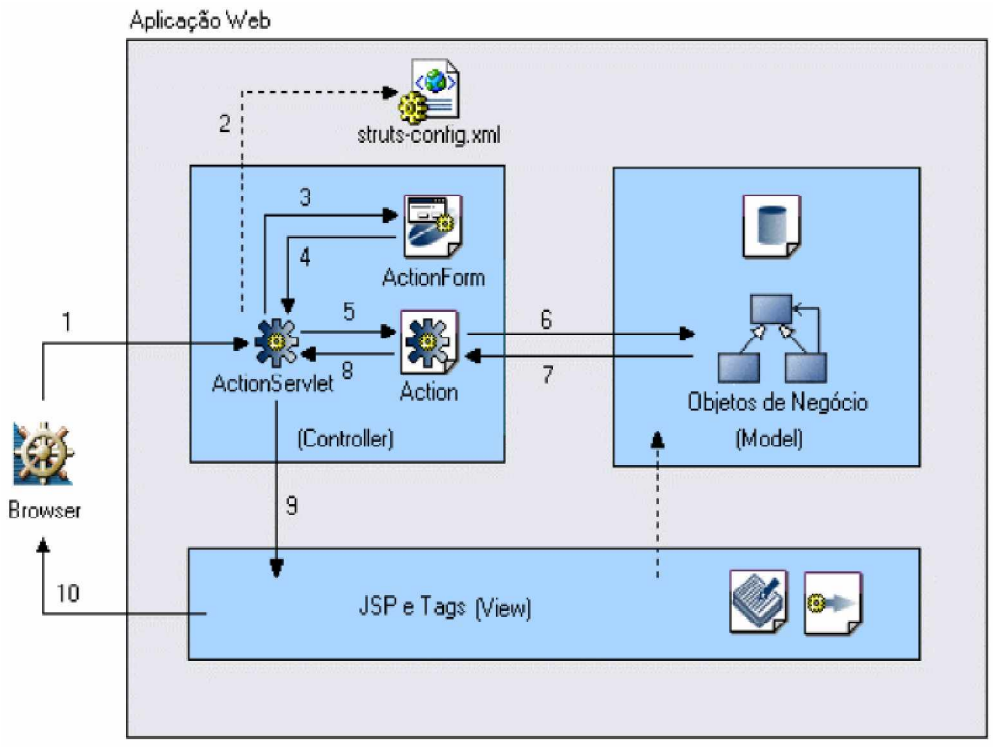

Figura 5 Funcionamento do struts.
Fonte: Costa et al (2004, p.15).

3.6. Camada de Modelo

3.6.1. OJB

Object-Relational Java Bridge (OJB) é um *framework* que implementa persistência objeto/relacional para Java, permite desenvolver classes persistentes em Java. O OJB permite que objetos sejam manipulados sem a necessidade de implementar nenhuma interface em especial ou estender alguma classe específica. A biblioteca dá suporte a cenários cliente-servidor (aplicações distribuídas) ou *standalone*, de forma que é possível utilizar a API OJB para persistência de objetos mesmo em ambientes J2EE (*Entity Beans* utilizando *Bean-Managed Persistence*).

3.6.2. *Pattern* DAO

O DAO (*Data Access Object*) é utilizado para encaplusar a lógica de acesso a dados. Assim, se for necessário a alteração de banco de dados, não é necessário alterar todo sistema, mas somente os DAOs.

Dentro do DAO são realizadas as consultas ou o acesso aos métodos do OJB. A intenção real de existência dos DAOs é que eles não possuam nenhuma lógica de negócio, apesar de algumas vezes ser necessário encaplusar algo dentro deles, especialmente quando outros *patterns* da camada de modelo não estão presentes (LAUTERT e OLIVEIRA, 2004).

Quando utilizado junto com OJB, ambos realizam o trabalho de abstrair a base, pois o OJB já mascara o tipo do banco de dados, ficando para o DAO a parte de controlar as conexões, exceções, retornos para os níveis superiores, entre outros.

O *Framework* OJB, tem a funcionalidade de mapear via objetos XML, os atributos de uma base de dados *PostgresSQL*. Uma classe DAO, irá possuir implementação referente à utilização deste *Framework* para persistir, recuperar e atualizar os dados provenientes de uma requisição de uma regra de negócio.

Depois que a regra de negócio, propriamente dita, for executada, é a vez de uma classe “DAO” (*Data Access Object*) entrar em ação. Ela deverá pegar o objeto que recebe como argumento e saber tratá-lo de acordo com os métodos de negócio que foram invocados na regra de negócio.

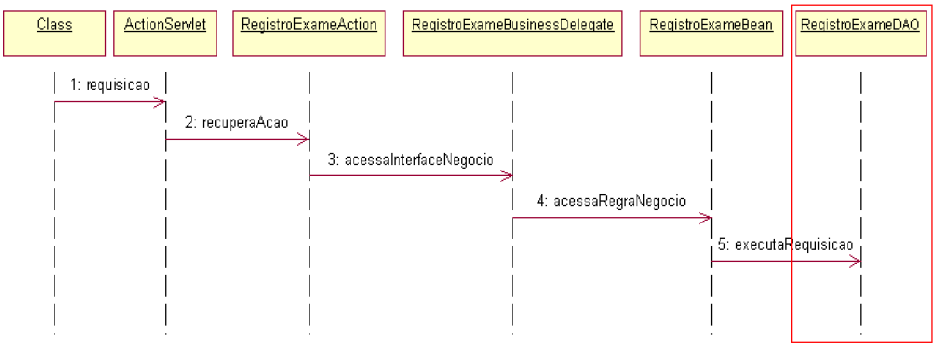

Figura 6 Diagrama de seqüência DAO.

Nesta situação, uma requisição é feita antes dos dados chegarem à regra de negócio da aplicação, ela passa por uma ação específica, um *Action* que irá identificar qual *Business Delegate* está associada com aquela requisição (Figura 6).

3.6.3. *Value Object*

Os *Value Objects* trabalham coletando conjuntos de informações relacionadas em um único objeto. Este objeto pode ser transferido do EJB para o cliente com uma única chamada remota. Em vez de fazer chamadas separadas para receber nome, endereço e formas de pagamento, o cliente pode receber um objeto contendo tudo em uma única chamada. Já que cada chamada na rede pode adicionar uma fração de segundo ao tempo que ela gasta para executar uma atividade, reduzir este *overhead* é geralmente o mais efetivo melhoramento de desempenho possível para uma aplicação J2EE.

Um objeto que pertencente ao *Value Object* tem como principal função mapear os dados digitados pelos clientes em uma aplicação e posteriormente trafegá-los pela rede, confrontando-se com as mais diversas camadas da aplicação. Um *Value Object* é popularmente conhecido como “VO”, ele deve ser constituído apenas com atributos e métodos de acesso.

3.6.4. *Pattern* EJB

Enterprise Java Beans (EJB) é uma arquitetura para componentes no lado servidor que possibilita e simplifica o processo de construir aplicações de objetos distribuídos em Java. Usando EJB, pode-se escrever aplicações escaláveis, robustas e seguras sem escrever sua própria infra-estrutura complexa para objetos distribuídos. EJB é um ambiente de desenvolvimento rápido para aplicações do lado do servidor; pode-se rápida e facilmente construir componentes do lado do servidor em Java através de uma infra-estrutura de objetos distribuídos provida pela indústria. EJB é desenvolvido para prover portabilidade e reusabilidade qualquer que seja o vendedor de serviços corporativos da camada do meio, ou seja, da *middlleware*.

O principal objetivo do EJB é administrar os objetos de negócio (incluindo os DAOs), e fornecer uma camada padrão para acesso a camada de modelo, definindo uma interface de alto nível aos subsistemas da aplicação, facilitando assim seu uso. Quando utilizada juntamente com *Struts*, é chamada de dentro das *Actions*, e assim,

essas duas classes formam a 'cola' entre a camada de modelo e a camada de controle.

Variação do EJB: em vez do EJB chamar método de negócios que chamariam DAOs, ele mesmo trata de chamar os DAOs e executar a lógica de negócio, eliminando assim mais um nível de abstração.

Um *enterprise beans* pode conter um ou mais objetos Java porque um componente pode ser mais do que um simples objeto. Sem levar em consideração a composição dos *enterprise beans*, os clientes do *beans* manipulam com uma simples interface do componente exposta para os clientes. Esta interface, assim como o *enterprise bean* propriamente dito, deve ser escrito conforme a especificação para *Enterprise Java Beans*. A especificação requer que os *beans* declarem alguns métodos requeridos, estes métodos permitem que o container EJB gerencie os beans uniformemente, sem levar em consideração em qual container o bean estará rodando (ENTERPRISE JAVA BEANS, 2003).

O cliente de um *enterprise bean* pode ser qualquer um – talvez um *servlet*, um *applet*, ou até mesmo outro *bean* corporativo. Em último caso, uma requisição do cliente a um *bean* pode resultar que um canal de *bean* seja executado.

A implementação do sistema trabalha com o tratamento de objetos distribuídos, garantindo segurança, persistência, transações e confiabilidade para o sistema, onde os chamados *Enterprise Java Beans* (ENTERPRISE JAVA BEANS, 2003), componentes de negócio são registrados em um servidor J2EE.

3.7. Camada View

A camada *View* é feita através de páginas JSP, que contém a parte HTML das páginas e toda a lógica para exibição dos dados.

Páginas JSP são como *templates* usados para produção automática de *servlets*. Escreve-se uma página HTML com alguns comandos Java, que são traduzidos em tempo de execução a um *servlet* com as seguintes propriedades:

a. O texto HTML na página é convertido em comandos Java de escrita de HTML e

b. os comandos Java na página são apenas copiados para dentro do servlet.

3.8. XML: (eXtensible Markup Language):

O XML é um padrão para publicação, combinação e intercâmbio de documentos multimídia, desenvolvido pelo consórcio W3C (World Wide Web Consortium). Assim como outras linguagens de marcação, XML lida com instruções embutidas no corpo de documentos chamadas *tags*, que permitem a descrição de dados. XML tem como base, linguagens mais antigas como, SGML e HTML, onde atualmente são empregadas na representação de estruturas de dados estruturados e semi-estruturados e seu intercâmbio na WEB.

3.9. XSL

Um documento XSL é um XML que consegue transformar um documento XML em outros formatos de documentos (HTML, Texto, PostScrip e RTF), além de poder utilizar alguns elementos de estilo disponíveis. A linguagem XSL pode ser utilizada para acrescentar aspectos de apresentação aos elementos de um documento XML. Desta forma, é possível criar múltiplas representações da mesma informação a partir de vários documentos XSL diferentes aplicados a um único documento XML.

3.10. *PostgreSQL*

O *PostgreSQL* é um SGBD objeto relacional, de livre distribuição, com código fonte aberto (open source), permitindo que possa ser modificado conforme a necessidade de utilização. Oferece suporte à Linguagem SQL de acordo com os padrões SQL92/SQL99.

Seu desenvolvimento teve início em 1985 na Universidade da Califórnia em Berkeley. Hoje é mantido por um grupo de programadores através da Internet. É comparável em recursos aos melhores bancos de dados comerciais existentes, inclusive sendo superior em muitos aspectos.

O *PostgreSQL*, na sua origem acadêmica, foi projetado para atender a potencialidade exigida em servidores de banco de dados para Internet. Ele funciona com conexões TCP/IP, oferecendo o acesso multiplataforma para aplicações na rede local ou Internet, de forma transparente e simultânea.

Por ser objeto relacional, inclui características dos gerenciadores de banco de dados relacionais e conceitos de orientação a objetos, como:

- Integridade referencial
- Constraints: chave primária e chave estrangeira para tabelas do banco de dados
- OID

  Herança: permite a criação de duas tabelas e uma herda os atributos da outra.

Desta forma, a tabela poupança além de conter seus atributos (taxa_juros e acumulados), conterá também os atributos de conta (nome_usuario, saldo_anterior, saldo_atual)

3.11 Entendendo a tecnologia Applet

Uma página de internet pode conter vários recursos, como texto, imagem, som, vídeo etc. E um destes recursos é o *Java Applet.* Este recurso é referenciado por uma página HTML da mesma maneira que os citados anteriormente: através de *tags*. No caso do *Applet*, a *tag* utilizada é a <APPLET></APPLET>. Esta *tag* será explorada com um pouco mais de profundidade logo adiante.

Segundo definição da *Sun* (SUN DEVELOPER NETWORK, 2005), um *Applet* é um programa, escrito na linguagem *Java*, que é chamado por uma página HTML. Da mesma maneira que outros recursos como texto ou imagem, o programa *Applet* é transferido do servidor *Web* para a máquina do usuário quando a página é acessada. Quando isto acontece, a Máquina Virtual *Java* (JVM – *Java Virtual Machine*) residente no navegador será ativada para executar o código do programa *Applet*. Assim, segundo Ricarte (2000)**,** *Applets* são executados no contexto de um outro programa (no caso, o sistema *Web*), o qual interage com o *Applet* e determina assim sua seqüência de execução.

Uma desvantagem clara é que será necessário aguardar o carregamento do programa *Applet*, para ser executado na máquina do usuário. Assim sendo, fica evidente que o tamanho deste programa *Applet* não deve ser muito grande, para não provocar a "fúria" dos usuários. De outro lado, também se percebe a vantagem de não ter que aguardar o processamento do programa toda vez pelo servidor, para a devolução de uma resposta ao cliente a cada alteração executada sobre a imagem (o que, numa aplicação distribuída como esta, significaria um grande problema).

Além de que, a solução *Applet* também mantém a característica de uma boa manutenibilidade de um sistema *Web*, já que o programa ficará armazenado no servidor. Quando o código-fonte for alterado, seus usuários passarão a utilizar a nova versão, sem se preocupar em fazer a atualização manual do programa.

3.11.1 As *tags* HTML

Uma página HTML pode conter vários *Applets*. Como já foi citado, a *tag* responsável por esta referência é <APPLET>...</APPLET>. No local em que está escrita esta *tag* é criado um espaço na página para a execução. O tamanho deste espaço é definido pelos atributos *height* e *width* (altura e largura). Outros atributos importantes são: *archive*, que fornece o nome do arquivo compactado (.jar, ou .zip) do *Applet*; *alt,* que, assim como nas *tags* de imagens, especifica um texto alternativo se o *Applet* não for carregado; *code*, que especifica o nome da classe (caso não se trate de um .jar, ou .zip, como citado anteriormente); o *codebase* fornece o endereço onde se encontra o *Applet;* e outros atributos que tratam da apresentação do *Applet,* como alinhamento (*align*) e espaçamento (*hspace, vspace*).

Outra funcionalidade importantíssima no uso de *Applets* é a passagem de parâmetros para o programa. Sem isto, a dinâmica de um programa *Applet* estaria seriamente comprometida. Entre as *tags* <APPLET> e </APPLET>, podem ser adicionadas outras, como o <PARAM>. Esta *tag* é responsável pela passagem de parâmetros ao *Applet*. Nela são definidos o *name* (nome) da variável e seu *value* (valor). O uso desta *tag* com funcionalidades de código JSP pode trazer grande dinamicidade ao programa.

3.11.2 A classe Applet

A criação de uma aplicação *Applet* é similar a qualquer outra aplicação Java. O programa Java deve ser criado e o arquivo de *bytecodes* (*class*) deve ser gerado. No entanto, os métodos de controle do mesmo se diferem das aplicações Java tradicionais.

O professor Ricarte (2000) cita algumas das diferenças em seu *site*. A execução de um *Applet*, por exemplo, não é iniciada pelo método *main()*, como nas

aplicações convencionais. Um *Applet* é executado como uma *thread* subordinada ao navegador.

O método *init()* é invocado pelo navegador no momento em que o *Applet* é carregado. Após isso, o método *start()* é invocado. Esse método é chamado também quando a área do *Applet* torna-se visível (uma espécie de *onFocus()*), reiniciando as operações que eventualmente tenham sido paralisadas pelo método *stop()* (que é executado quando o *Applet* perde o foco). Assim, evita que operações que demandem muito processamento continuem a executar desnecessariamente. O método *stop()* também é chamado antes de um *destroy()*, que serve para liberar recursos - além de memória - que o *Applet* tenha eventualmente alocado para sua execução (obviamente, na finalização do programa).

Assim sendo, nada impede que um programa *Applet* instancie alguma janela *JFrame*, por exemplo, o que torna esta tecnologia muito poderosa, sempre observando, no entanto, aquelas restrições citadas anteriormente, como o tamanho do arquivo.

Não se pode esquecer, naturalmente, do método *getParameter()*, que permite a comunicação do *Applet* com o navegador no qual ele está inserido, para recuperar aqueles parâmetros passados pelas *tags* <PARAM>.

3.11.3 Segurança e restrições

Existem restrições sobre que tipo de processamento pode ser feito nos *Applets*, a fim de evitar que este atue de forma indesejada no ambiente de seu usuário (por exemplo, apagando arquivos ou aproveitando sua presença no ambiente do usuário a fim de obter informação local).

Como o próprio site da *Sun* (SUN MICROSYSTEMS SECURITY, 2005) ilustra, quando o *Applet* é executado, *"*o modelo de segurança da plataforma atua no sentido de que o Applet só possa 'brincar' em sua caixa de areia e limita seu acesso a um conjunto restrito de recursos*"*.

O resultado disso é o isolamento desta "caixa de areia", impedindo que o programa acesse conteúdo da máquina do usuário, ou salve qualquer arquivo com código malicioso nela.

No entanto, existem casos que este acesso é necessário, como por exemplo, para abrir uma imagem que o usuário queira visualizar e que se encontra no seu disco. Para isso, existe um arquivo de configuração e assinatura de *Applet*.

A política de segurança padrão é estabelecida em um arquivo do sistema, *Java.policy*. Cada usuário pode estabelecer adições a essa política através da criação de um arquivo particular de estabelecimento de política *Java.policy* (RICARTE, 2000).

3.11. Considerações finais do capítulo

A interconexão das redes de computadores está disponibilizando informações a qualquer parte do mundo. O emprego das tecnologias descritas anteriormente permitiram o desenvolvimento de um modelo de Sistema de Informação para área médica flexível e interoperável.

No próximo capítulo será apresentada a modelagem de um Sistema de Informação para radiodiagnóstico por imagens, desenvolvido utilizando os conceitos de gestão da informação abordados anteriormente e as tecnologias descritas anteriormente.

Capítulo 4

4. Procedimentos metodológicos e modelagem do sistema

4.1 Procedimentos Metodológicos

A pesquisa tem um caráter pragmático, é um “processo formal e sistemático de desenvolvimento do método científico. O objetivo fundamental da pesquisa é descobrir respostas para problemas mediante o emprego de procedimentos científicos” (GIL, 1999)

Desta forma, foi desenvolvida uma pesquisa de natureza aplicada com o objetivo de gerar conhecimentos sobre como a modelagem e implementação de um Sistema de Informação poderão otimizar exames de diagnósticos por imagens em clínicas de imagens.

O problema foi abordado nesta pesquisa de forma qualitativa, pois os fatores que definem a modelagem, implementação e otimização dos processos produtivos de uma clínica de radiodiagnóstico são dinâmicos, isto é, um vínculo indissociável entre a teoria da Inovação Tecnológica e a subjetividade da modelagem e implementação do modelo propriamente dito.

Do ponto de vista de seus objetivos a pesquisa foi exploratória propiciando maior familiaridade como os conceitos de Sistemas de Informação e seus impactos em um sistema complexo e real.

O procedimento técnico adotado foi a pesquisa-ação, pois foi realizada em estreita associação com uma ação ou com resolução de um problema coletivo. Os pesquisadores e participantes representativos da situação ou de problema estão envolvidos de modo cooperativo e participativo. (SILVA, 2001, p. 22)

Para se atingir os objetivos propostos nesta pesquisa, utilizou-se o método científico indutivo, através da aplicação de um conjunto de procedimentos intelectuais e técnicos fundamentados na experiência, análise, avaliação e validação da implementação de um Sistema de Informação e seu impacto no processo produtivo de uma clínica de radiodiagnóstico.

4.2 Modelagem

O grande desafio das equipes de desenvolvimento de aplicações é cada vez mais produzir aplicativos seguros, eficientes, de fácil manutenção, reutilizáveis e em prazos cada vez menores.

O paradigma da orientação a objetos tem tido um grande avanço nestes últimos tempos.

O sucesso para o desenvolvimento de aplicações com tecnologia orientada a objetos está intimamente ligada a arquitetura que vamos usar para construir a aplicação. A tendência indica que está arquitetura estará baseada na organização da aplicação em camadas e na observação dos padrões utilizados pelo mercado.

A organização em camadas é a chave para a independência entre os componentes e esta independência é que vai atingir os objetivos de eficiência, escalabilidade, reutilização e facilidade de manutenção.

Em um primeiro instante, produzir aplicativos multicamadas pode parecer mais complexo. O termo camada pode significar uma separação física ou lógica. No contexto de apresentação deste trabalho, a produção de software vai considerar camada como uma referência à separação lógica de responsabilidades.

4.2.1 Aplicações monolíticas

Nos inicio dos anos 70, um aplicativo era desenvolvido para ser usado em uma única máquina. Geralmente este aplicativo continha todas as funcionalidades em um único módulo gerado por uma grande quantidade de linhas de código e de manutenção difícil.

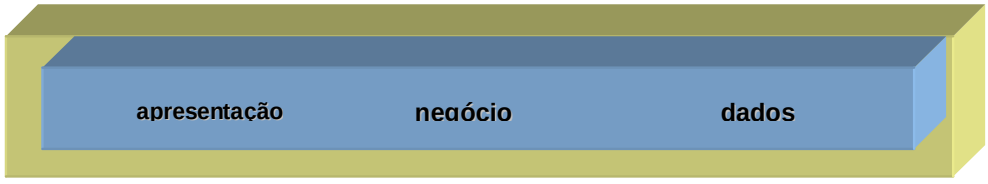

Figura 7 Aplicação monolítica.
Fonte: MVC, 2004.

A entrada do usuário, verificação, lógica de negócio e acesso à banco de dados estava presente em um mesmo lugar. Pode-se definir este tipo de aplicação como aplicação de uma camada ou monolítica, como demonstrada na Figura 7.

4.2.2 Aplicações em duas camadas

A necessidade de compartilhar a lógica de acesso a dados entre vários usuários simultâneos fez surgir às aplicações em duas camadas. Nesta estrutura (Figura 6) a base de dados foi colocada em uma máquina específica, separada das máquinas que executavam as aplicações. Nesta abordagem, há aplicativos instalados em estações clientes contendo toda a lógica da aplicação.

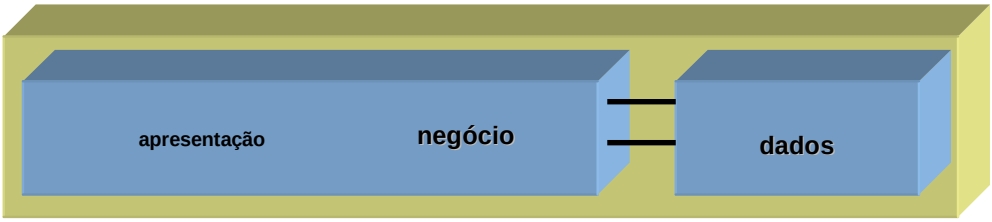

Figura 8 Aplicação em duas camadas.
Fonte: MVC, 2004.

Um grande problema neste modelo é o gerenciamento de versões, pois para cada alteração os aplicativos precisam ser atualizados em todas as máquinas clientes.

4.2.3 Aplicações em três camadas

Com o advento da Internet houve um movimento para separar a lógica de negócio da interface com o usuário. A idéia é que os usuários da WEB possam acessar as mesmas aplicações sem ter que instalar estas aplicações em suas máquinas locais.

Neste modelo (Figura 9), o aplicativo é movido para o Servidor e um navegador WEB é usado. O aplicativo é executado em servidores WEB com os quais o navegador WEB se comunica e gera o código HTML para ser exibido no cliente.

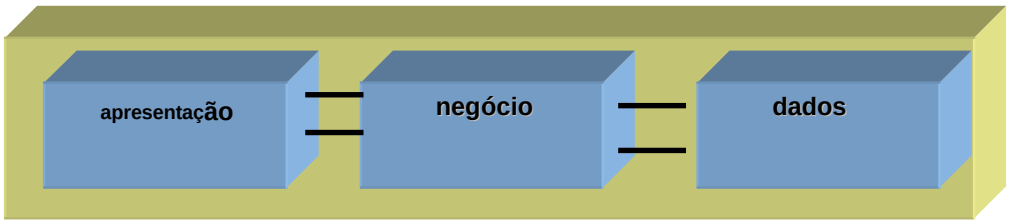

Figura 9 Aplicação em três camadas.
Fonte: MVC, 2004.

Neste modelo de três camadas a lógica de apresentação está separada em sua própria camada lógica. A separação em camadas torna os sistemas mais flexíveis permitindo que as partes possam ser alteradas de forma independente.

As funcionalidades da camada de negócio podem ser divididas em classes e essas podem ser agrupadas em pacotes ou componentes, reduzindo as dependências entre as mesmas e pacotes; podem ser reutilizadas por diferentes partes do aplicativo e até por aplicativos diferentes. O modelo de três camadas tornou-se a arquitetura padrão para sistemas corporativos com base na WEB, e é o modelo escolhido para apresentação deste estudo.

O crescimento do paradigma de orientação a objetos ajudou a promover uma maior modularidade, pois os objetos encapsulam seus dados (propriedades, métodos) e oferecem funcionalidades através de seus métodos. Projetando-se de forma adequada os objetos podem ter reduzido as dependências entre si, ficando assim fracamente acoplados e serão mais fáceis de manter e evoluir.

O modelo de três camadas físicas (3-*tier*) divide um aplicativo, de modo que a lógica de negócio resida no meio das três camadas físicas. Isto é chamado de camada física intermediária ou camada física de negócios. A maior parte do código escrito reside na camada de apresentação e de negócio.

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 recente e foi originalmente desenvolvida para mapear as tarefas tradicionais de entrada, processamento e saída para o modelo de interação com o usuário. Usando o padrão MVC fica fácil mapear esses conceitos no domínio de aplicações WEB multicamadas.

Na arquitetura MVC o modelo representa os dados da aplicação e as regras do negócio que governam o acesso e a modificação dos dados. O modelo mantém o estado persistente do negócio e fornece ao controlador a capacidade de acessar as funcionalidades da aplicação encapsuladas pelo próprio modelo.

Um componente de visualização renderiza o conteúdo de uma parte particular do modelo e encaminha para o controlador as ações do usuário; acessando também os dados do modelo via controlador e definem como esses dados devem ser apresentados.

O controlador define o comportamento da aplicação, é ele que interpreta as ações do usuário e as mapeia para chamadas do modelo. Em um cliente de aplicações WEB essas ações do usuário poderiam ser cliques de botões ou seleções de menus. As ações realizadas pelo modelo incluem ativar processos de negócio ou alterar o estado do modelo. Com base na ação do usuário e no resultado do processamento do modelo, o controlador seleciona uma visualização a ser exibida como parte da resposta a solicitação do usuário. Há normalmente um controlador para cada conjunto de funcionalidades relacionadas.

A arquitetura de três camadas, que está representada na Figura 10, é uma implementação do modelo MVC. O modelo MVC está preocupado em separar a informação de sua apresentação.

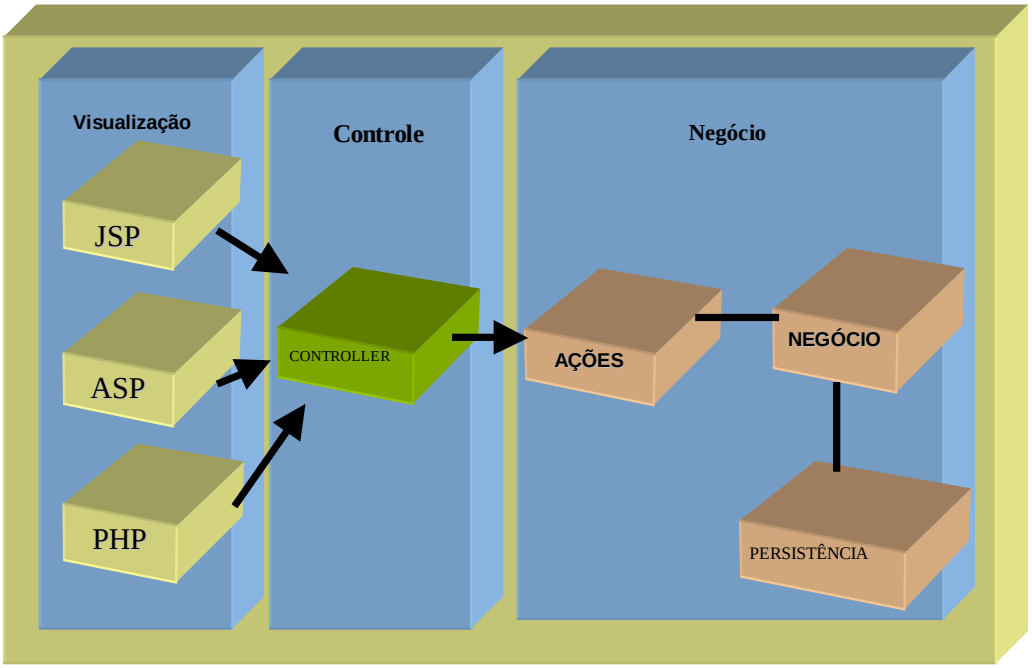

Figura 10 Aplicando MVC em um modelo de três camadas.
Fonte: MVC, 2004.

Camada de apresentação ou visualização - Não está preocupada em como a informação foi obtida ou onde ela foi obtida, apenas exibe a informação.

- Inclui os elementos de exibição no cliente: HTML, XML, ASP, Applets;
- É a camada de interface com o usuário;
- É usada para receber a entrada de dados e apresentar o resultado.

Camada de lógica da Aplicação - É o coração da aplicação. Responsável por tudo que a aplicação vai fazer.

- Modela os dados e o comportamento por atrás do processo de negócios;
- Preocupa-se apenas com o armazenamento, manipulação e geração de dados;
- É um encapsulamento de dados e de comportamento independente da apresentação.

Camada de Controle - determina o fluxo da apresentação servindo como uma camada intermediária entre a camada de apresentação e a lógica de negócios, responsável por controlar e mapear as ações.

4.2.4 Arquitetura com o Padrão Arquitetural MVC (*Model* 1)

A arquitetura "*Model* 1" submete solicitações a páginas JSP, que acessam indiretamente objetos de negócio através de beans, ou ainda através das próprias páginas JSPs. O acesso indireto separa as alterações em páginas JSPs daquelas em objeto profissional, que são tipicamente freqüentes, especialmente para grandes projetos, porque estas alterações são tratadas nos beans de negócio. Contanto que interfaces beans permaneçam constantes, páginas JSP são independentes de implementações.

Desenvolvedores de software implementam os objetos de negócio e os beans. Idealmente, autores de páginas WEB seriam responsáveis por páginas JSP, resultando em uma divisão de trabalho onde objetos de negócio e páginas WEB são desenvolvidas em paralelo por desenvolvedores com diferentes habilidades. Esta divisão de trabalho é difícil de ser obtida com a arquitetura "Model 1" pois páginas JSP são responsáveis pela geração de conteúdo, o que quase sempre requer código Java, além do seu escopo original.

Uma página JSP acessa diretamente, através de uma conexão, dados em um meio de armazenamento. Isso significa que uma página JSP será detentora de objetos relacionados diretamente a acesso a dados, tratando, portanto, da manutenção de uma obtenção de dados. Isso não é uma boa prática, e deixa sua página completamente não reutilizável (Figura 11).

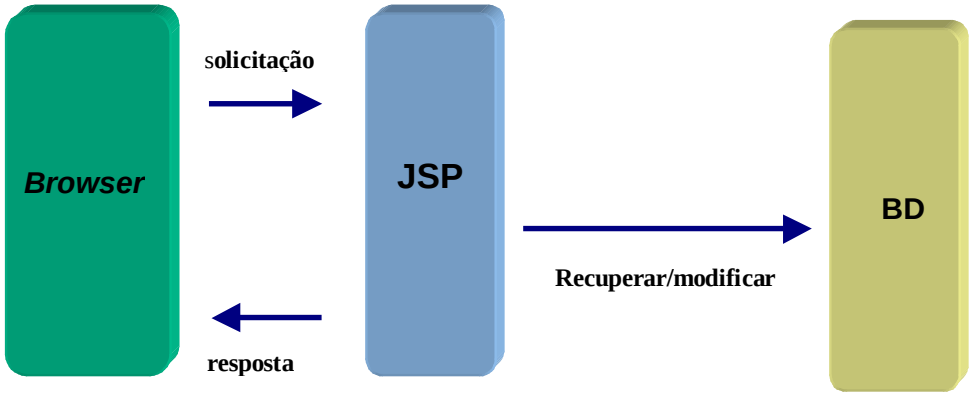

Figura 11 “Model 1” da Sun.
Fonte: MVC, 2004.

Uma página JSP acessa um componente (*JavaBeans*), que possui agora lógica (acesso a dados e outros) específica de componentes encapsulada, facilitando assim a reutilização desta classe para demais necessidades e requisições (Figura 12).

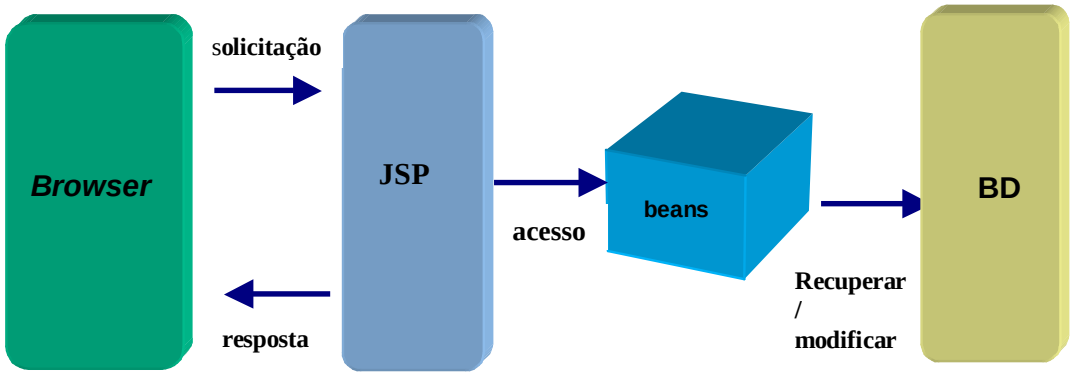

Figura 12 “Model 1” da Sun.
Fonte: MVC, 2004.

Em virtude de uma visão de trabalho entre autores de páginas WEB e desenvolvedores de software ser difícil de se obter com a arquitetura “Model 1”, esta abordagem é apropriada apenas para pequenos projetos com poucos desenvolvedores, todos fluentes em Java, JSP, HTML.

4.2.5 Arquitetura com o Padrão Arquitetural MVC (Model 2)

A Arquitetura "*Model* 2" submete suas solicitações a um Servlet que acessa objetos de negócio para criar conteúdo. Este conteúdo é armazenado em um bean, que é acessado por uma página JSP. A página JSP subseqüentemente apresenta o conteúdo, tipicamente em HTML.

Separar geração de conteúdo de apresentação é benéfico, pois código Java na maior parte, é restrito à geração de conteúdo. Este encapsulamento de código Java permite que desenvolvedores de software se concentrem em Servlets e objetos de negócio e autores de páginas concentram-se em Scriptles e demais páginas.

Model 2 é uma arquitetura MVC, onde objetos de negócio representam o modelo (dados/model), Servlets representam os controladores (controller) que gerenciam solicitações, e as páginas JSP são visões(views) do modelo. A arquitetura MVC, originada do Smaltalk, resistiu ao teste do tempo porque separa lógica profissional de lógica de apresentação. Esta separação permite componentes plugáveis, resultando em software flexível, reutilizável e adaptável.

4.2.5.1 Modelo padrão da arquitetura MVC

Este modelo (Figura 13) representa o fluxo básico de ações entre cada responsabilidade. Perceba que o *Controller* é responsável por delegar a regra de negócio de uma ação específica, e também encontrar a visualização específica daquela requisição.

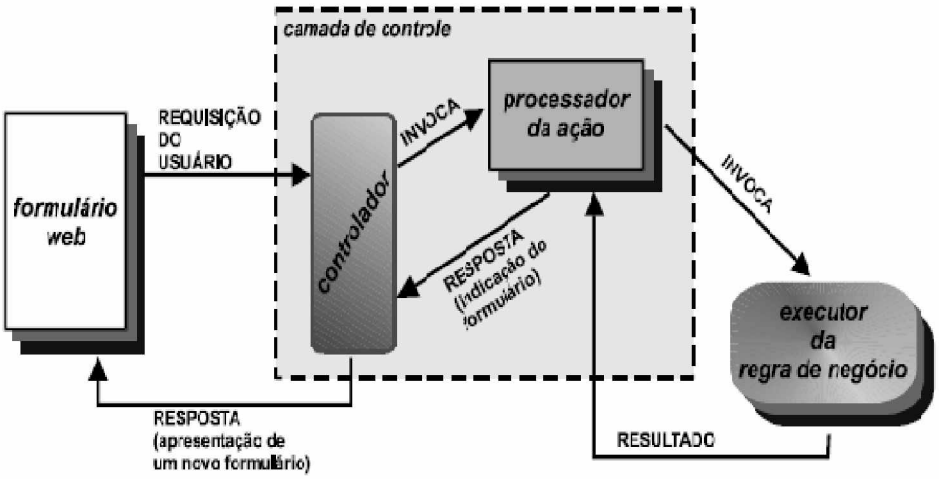

Figura 13 Descrição do fluxo de uma aplicação "Model 2".

Fonte: (PAIS, 2004)

O padrão MVC encapsula três abstrações gerais que estão presentes na maioria das aplicações: modelos, visões e controladores. Ao encapsular o que outras aplicações entrelaçam, aplicações MVC são muito mais flexíveis e reutilizáveis do que suas contrapartes tradicionais.

4.2.6 Vantagens do modelo MVC.

1. Como o modelo MVC gerencia múltiplos visualizadores usando o mesmo modelo é fácil manter, testar e atualizar sistemas múltiplos;
2. É muito simples incluir novos clientes apenas adicionando seus visualizadores e controles;
3. Torna a aplicação escalável;
4. É possível ter desenvolvimento em paralelo para o modelo, visualizador e controle, pois são independentes.

4.2.7 Desvantagens do modelo MVC

1. Requer uma quantidade maior de tempo para analisar e modelar o sistema;
2. Requer pessoal especializado;
3. Não é aconselhável para pequenas aplicações.

4.2.8 Abordagem de desenvolvimento

De acordo com a fase de criação da aplicação, alguns conjuntos de etapas foram desenvolvidas na abordagem de desenvolvimento do projeto. Para este, será utilizado um paradigma chamado "**Modelo Evolutivo**", que é baseado no principio de desenvolvimento incremental e interativo, onde novas funções serão adicionadas a cada ciclo, gerando uma nova versão do *software* permitindo um *feedback* mais imediato do usuário.

Ciclo de Requisitos e Análise do sistema.

Ciclo Projeto/Modelagem.

Ciclo Implementação.

Ciclo Testes e homologação e transferência da aplicação.

Ciclo Manutenção.

4.2.9 Ciclo de requisitos e análise do sistema

Os requisitos referem-se às necessidades dos usuários. É de fundamental importância a compreensão para se ter sucesso no desenvolvimento de uma aplicação.

A análise de requisitos garante uma estrutura de dados adequada para que futuras aplicações tais como estudos clínicos, possam ser implementados e também contar com todas as informações necessárias.

A técnica de análise utilizada, e que atualmente mais se destaca, é a Análise Orientada a Objetos (OOA – *Object Oriented Analizing*) que traz diversos benefícios, tais como: funcionalidades complexas podem ser desenvolvidas com uma codificação menor, melhor e mais objetiva onde um rápido desenvolvimento será alcançado, tendo uma aplicação que poderá ser reutilizada e mantida facilmente.

4.2.10 Ciclo de Projeção e modelagem do sistema

Levando em consideração um estudo realizado, que demonstra que até 60% de todo o código produzido em uma aplicação é destinado a construção de interfaces e problemas de interação com o usuário, adotou-se uma técnica decorrente da decisão tomada na fase anterior à modelagem/projeção do sistema, onde seguindo as mesmas características da Análise Orientada a Objetos (OOA – *Object Oriented Analising*) adotou-se o *Design* Orientado a Objetos (OOD - *Object Oriented Design*), construindo assim uma interessante técnica de desenvolvimento para fechar a segunda fase de desenvolvimento do projeto, a Análise e *Design* Orientados a Objetos (OOAD – *Object Oriented Analising & Design*).

Esta técnica ajuda na obtenção de ferramentas de 4ª geração como a Ferramenta *CASE* que ajudarão na modelagem (desenho) da aplicação, facilitando

assim a construção de protótipos no ciclo de desenvolvimento e contribuindo para o aceleramento de diversas fases do sistema, dentre elas a codificação da aplicação e geração de bases de dados para armazenamento de informações.

Como linguagem de modelagem e unificação dos requisitos do sistema será utilizado a UML (*Unified Modeling Language* – OMG, 2004) para especificar, padronizar e documentar entidades do sistema, utilizando uma linguagem natural e capaz de traduzir do mundo real os problemas reais encontrados para que os mesmos possam ser computacionalmente aplicados.

4.2.11 Ciclo de implementação

A Linguagem de desenvolvimento escolhida foi Java, por ser uma linguagem orientada a objetos, portável, extensível e que abrange um grande número de *framework*s disponível no mercado para incrementar e especializar diversas tarefas do projeto.

O paradigma utilizado para a construção deste projeto será de objetos orientados (OOP – *Object Oriented Programming*), para garantir confiabilidade da regra negocial, escalabilidade da aplicação dentre outras características, requisitos esses, fundamentais para aplicação.

A utilização deste conceito tem como objetivo agilizar o processo produtivo do sistema em desenvolvimento, e garantir também flexibilidade e extensibilidade, para utilização de padrões de projetos (*Design Patterns*) e *framework*s (componentes de *software*), reforçando a idéia de uma aplicação em um ambiente de desenvolvimento totalmente livre e open source, livre de licenças e dependência direta de fornecedores.

A implementação do sistema trabalhou com o tratamento de objetos distribuídos, garantindo segurança e confiabilidade para o sistema, onde os *chamados Enterprise Java Beans*, componentes de negócio serão utilizados para suprir necessidades de desenvolvimento de regras de negócio para aplicação podendo ser reutilizados de acordo com a necessidade do projeto.

4.2.12 Ciclo de Testes

Técnicas de testes podem ser implementadas para assegurar que o *software* está realmente em acordo com suas especificações e livre de erros. Testes de unidade, testes de integração, testes de sistema e testes de instalação são exemplos de técnicas que poderão ser utilizadas.

De acordo com as necessidades do projeto, testes de unidades serão aplicados na fase de implementação do projeto.

O teste de unidade, realizado através do *Framework Junit*, será realizado de dois modos: Em *Alfa-test*: os testes foram realizados pelos próprios desenvolvedores no ambiente de desenvolvimento e em *Beta-Tests*, onde os testes serão realizados em um ambiente de execução pelos próprios usuários do *software*.

4.2.13 Ciclo de homologação e transferência

Através dos componentes como Ant, *scripts* automatizados de compilação e o componente *JBoss*, servidor de aplicação, tarefas serão aplicadas para que a aplicação seja polimórfica e de acordo com os comandos dados, a aplicação será criada de uma forma, ou seja, tanto para testes unitários em ambiente de produção como testes em ambiente de execução, além de reformular várias características de acordo com a necessidade de execução e *deploy* (colocar a versão do sistema em execução no servidor de aplicações) da aplicação.

4.3 Considerações finais do capítulo

A modelagem de sistemas é a etapa fundamental no desenvolvimento e implementação de sistemas, desta forma foram utilizadas as mais recentes técnicas de modelagem.

No próximo capítulo será apresentado como o sistema foi implementado, utilizando a modelagem descrita anteriormente e quais as dificuldades encontradas no desenvolvimento do mesmo.

Capítulo 5

5. Implementação

A implementação do sistema foi baseada nos padrões descritos a seguir, as figuras que compões este capítulo foram extraídas de Matos (2005, p. 65 e 95), que utiliza a técnica da UML de desenvolvimento e implementação de sistemas.

Padrões Arquiteturais

- MVC (*Modelling View Controller*).

Padrões estruturais

Ø Camada de Apresentação (JAVA BLUE PRINTS, 2005):

- *Front Controller.*
- *View Helpers*.
- *Composite View.*
- *Comands.*
- *Dispatcher.*
- *Service to Worker.*
- *Business Delegate.*
- *Value Object.*
- *Service Locator.*

Ø Camada de Negócio:

- *Session Façades.*

Ø Camada de Integração ou Persistência:

- *Data Access Objects.*

5.1. PADRÕES DA CAMADA DE APRESENTAÇÃO:

5.1.1. Front Controller

A requisição de páginas *WEB* possui um acesso centralizado, facilitando assim a manutenção de informações e o fluxo da aplicação (Figura 14).

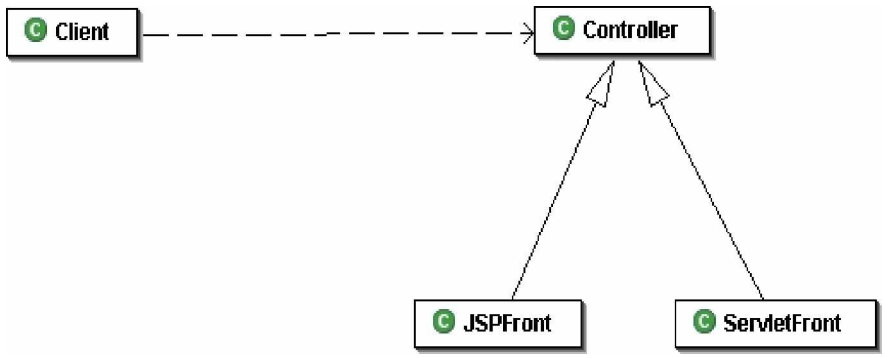

Figura 14 Estrutura padrão *Front Controller*.

Todas as páginas da aplicação possuem um ponto comum de requisição (Figura 15). Via tecnologia XML, o fluxo da aplicação é delegado para as ações específicas.

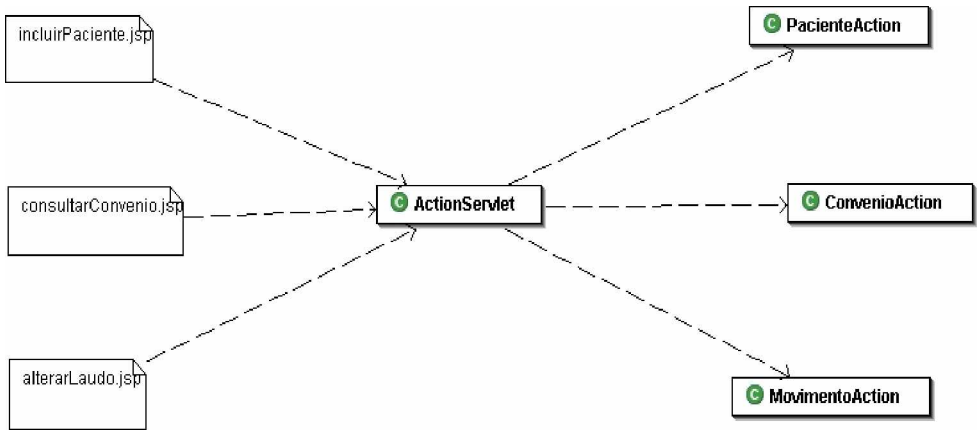

Figura 15 Diagrama de componentes do padrão *Front Controller* na aplicação.

Um *Servlet* (ou até mesmo uma página com *scriplets*) intercepta a requisição do usuário e direciona, ou “agrega valor” a requisição corrente.

Seqüência de ações

Quando uma ação específica é selecionada, a requisição é delegada para a regra de negócio relacionada (Figura 16).

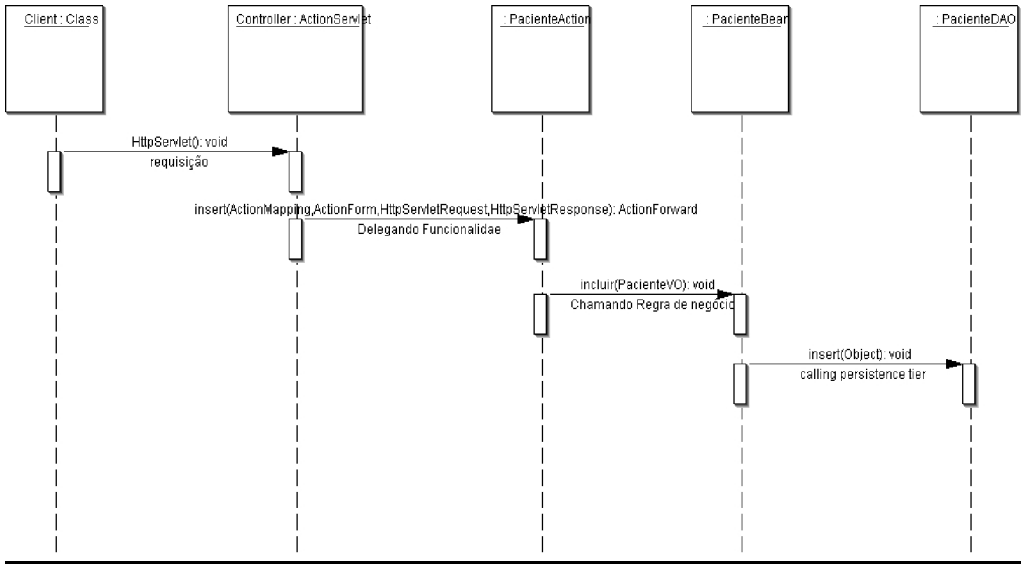

Figura 16 Seqüência de ações do padrão *Front Controller* na aplicação.

Esta técnica permite que a aplicação tenha um único ponto de requisição, reduzindo o tempo de manutenção do software e centralizando o processamento. Isto torna uma aplicação para internet mais ágil.

5.1.2. View Helpers

Uma visualização específica de um cliente poderá possuir várias outras visualizações incluídas, onde estas por sua vez, poderão possuir auxiliadores (Figura 17).

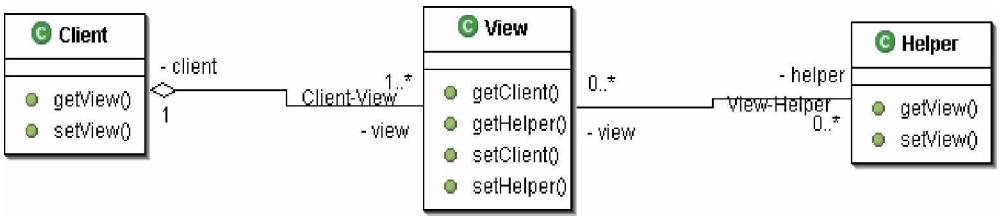

Figura 17 Estrutura do padrão *View Helper*.

Páginas de *script* pedem ajuda a um auxiliador que executará uma regra capaz de formatar os dados de acordo com a tecnologia de apresentação específica, neste caso *WEB* (Figura 18).

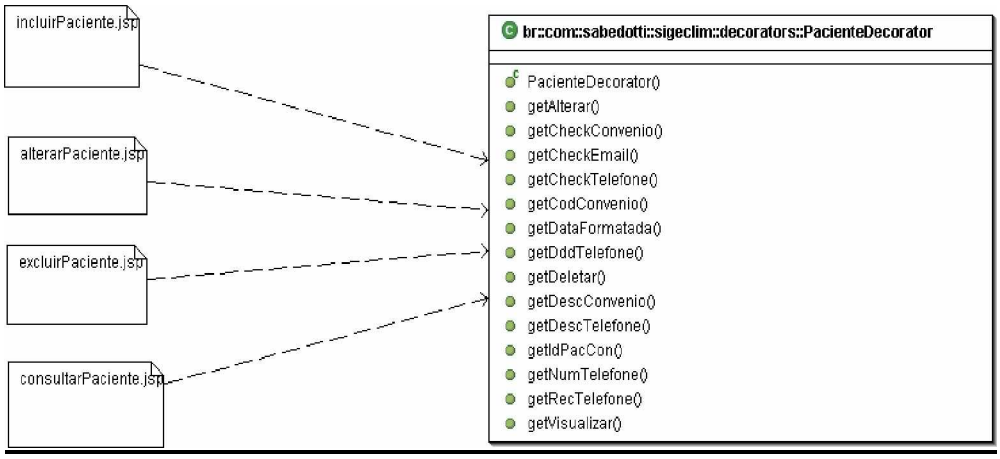

Figura 18 Padrão *View Helper* na aplicação.

Estes auxiliadores possuem diversas funcionalidades e podem ser utilizados de diversas maneiras, mas sempre tentando atingir o mesmo objeto: facilitar a disponibilização de conteúdo para a aplicação cliente. Utiliza-se um *JavaBeans* ou uma *Tag* JSP personalizada para encapsular funcionalidade e separar a funcionalidade de *scriptlets*.

Seqüência das ações

Esta situação demonstra a execução de um *Helper* para recuperar dados e disponibilizá-lo para uma outra visualização, que será responsável por executar uma consulta em uma base de dados (Figura 19).

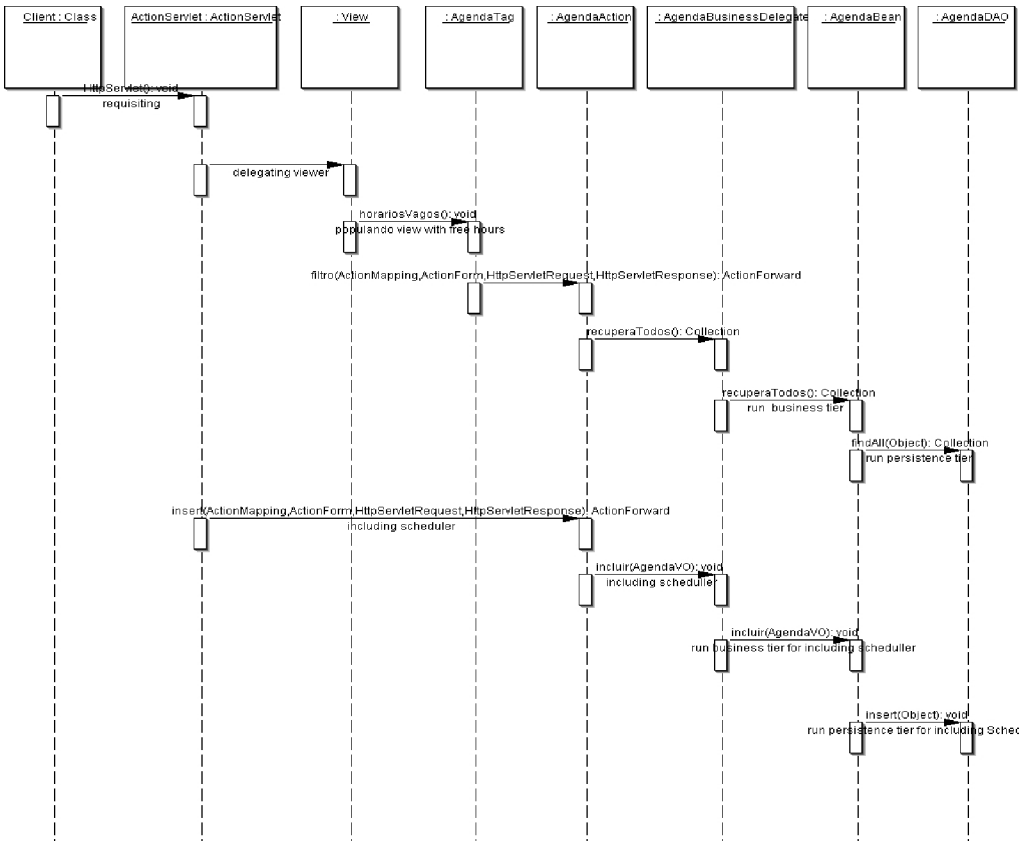

Figura 19 Seqüência de ações do padrão *View Helper* na aplicação.

Diferente do que acontece em uma aplicação tradicional, que há a necessidade de uma consulta a base de dados através de códigos SQL, neste padrão o usuário não tem a preocupação com quais campos da base de dados serão recuperados, pois o parâmetro passado na requisição é um objeto com uma chave primária e o retorno será também um objeto.

5.1.3 Composite View

Uma visualização poderá ser composta por uma ou mais visualizações, fazendo parte de um todo. Compõe-se uma página JSP de vários subcomponentes diferentes, para fornecer uma visão típica de página da *WEB* com vários painéis (Figura 20).

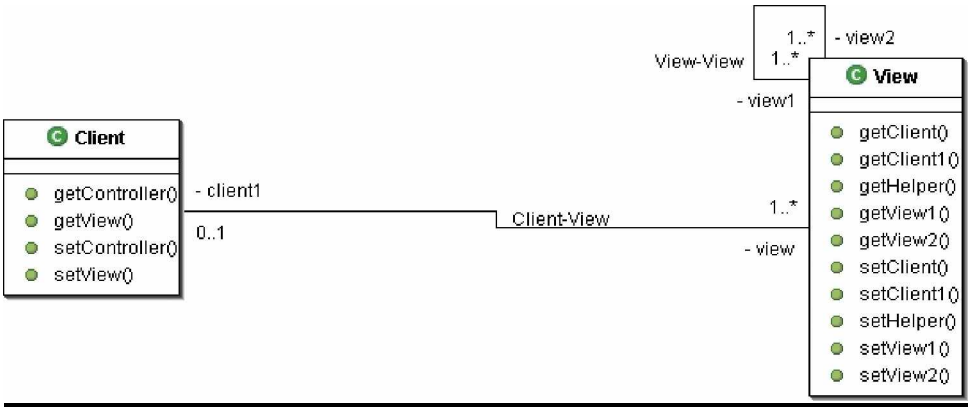

Figura 20 Estrutura do padrão *Composite View*.

Toda página na aplicação é composta por várias visualizações (Figura 21), fica transparentes para os usuários, e até mesmo para determinados desenvolvedores da aplicação em questão.

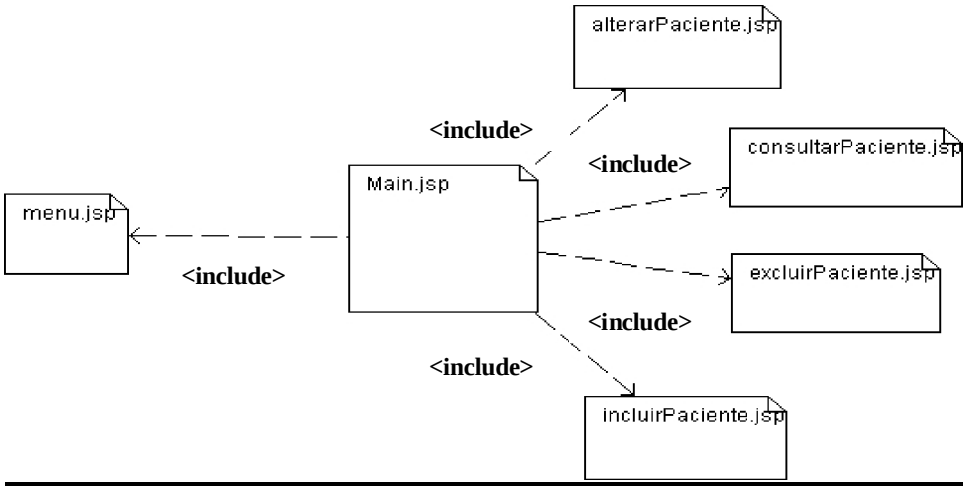

Figura 21 Padrão *Composite View* na aplicação.

Este padrão encapsula as várias visualizações das páginas JSP, desta forma os desenvolvedores da aplicação não necessitam ter a preocupação de definir a

página que será exibida, pois o *Composite View*, que será acionado e através do parâmetro passado invocará a página solicitada.

5.1.4. Service to Worker:

Este padrão define que toda requisição poderá possuir uma ou mais ações associadas a ele (Figura 22).

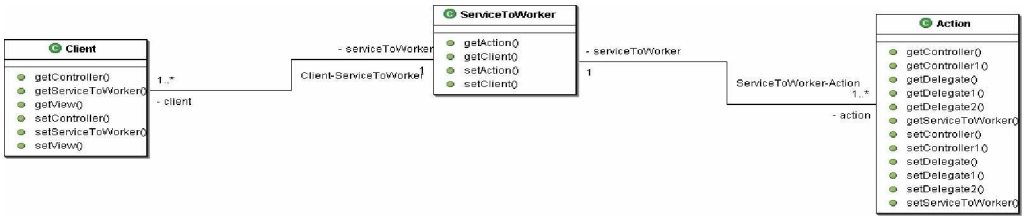

Figura 22 Estrutura do padrão *Service To Worker*.

Este padrão é representado na aplicação por um *ActionServlet*, uma classe que engloba outros padrões relacionados a requisição de ações e execução de regras de negócio da aplicação.

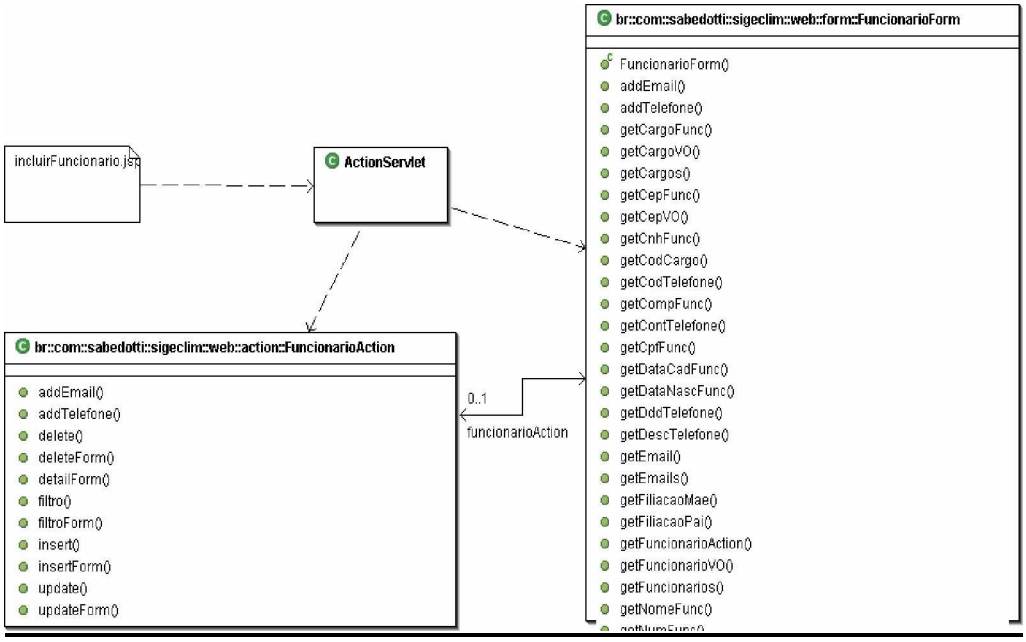

Figura 23 Padrão *Service To Worker* na aplicação.

Quando uma requisição decorrente de uma página JSP (incluirFuncionario.jsp) é acionada, contendo dados para cadastro de um novo funcionário na aplicação, por exemplo, este padrão indica que classes específicas terão que estar aptas a reconhecer a requisição e a tratar este requerimento com uma regra de negócio adequada que será executada de uma forma abstrata, reforçando assim a modularização da aplicação (Figura 23).

Seqüência de ações

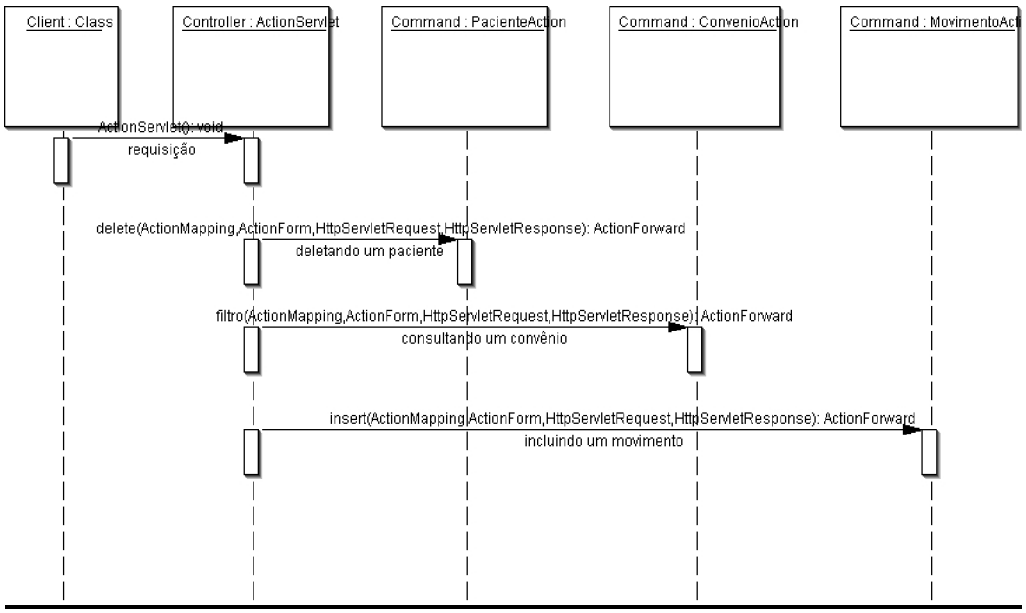

Figura 24 Seqüência de ações do padrão *Service To Worker* na aplicação.

A seqüência de ações do Padrão *Service To Worker* representada na Figura 24, demonstra como ação é implementada.

5.1.5. Dispatcher View

Este padrão indica que, decorrente da execução do padrão *Service to Worker* que delega a ação para uma classe que abstrai a chamada de uma regra de negócio específica na aplicação, que após que esta regra de negócio for executada, a mesma será “despachada” (Figura 25), conseqüentemente para uma nova visualização. Na aplicação, este padrão também é representado pela classe *ActionServlet*.

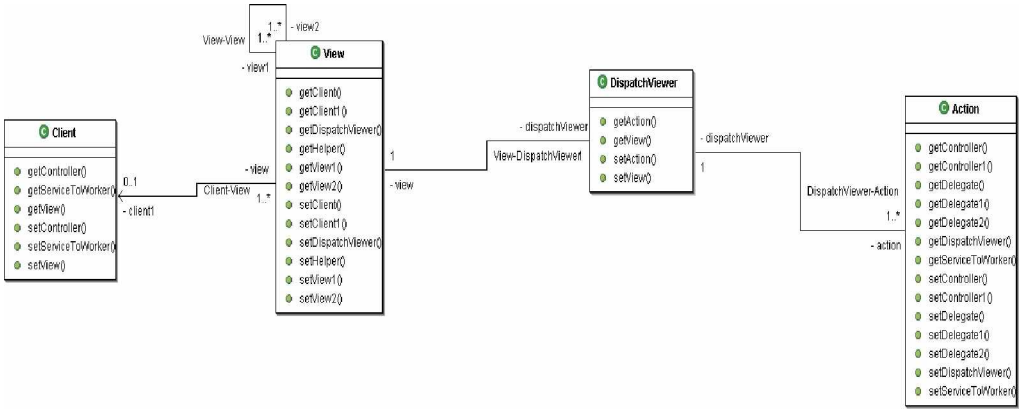

Figura 25 Estrutura do padrão *Dispatcher View*.

Após a execução de uma regra e negócio específica, proveniente de uma requisição, o *controller* da aplicação será responsável por direcionar o fluxo da aplicação (Figura 26).

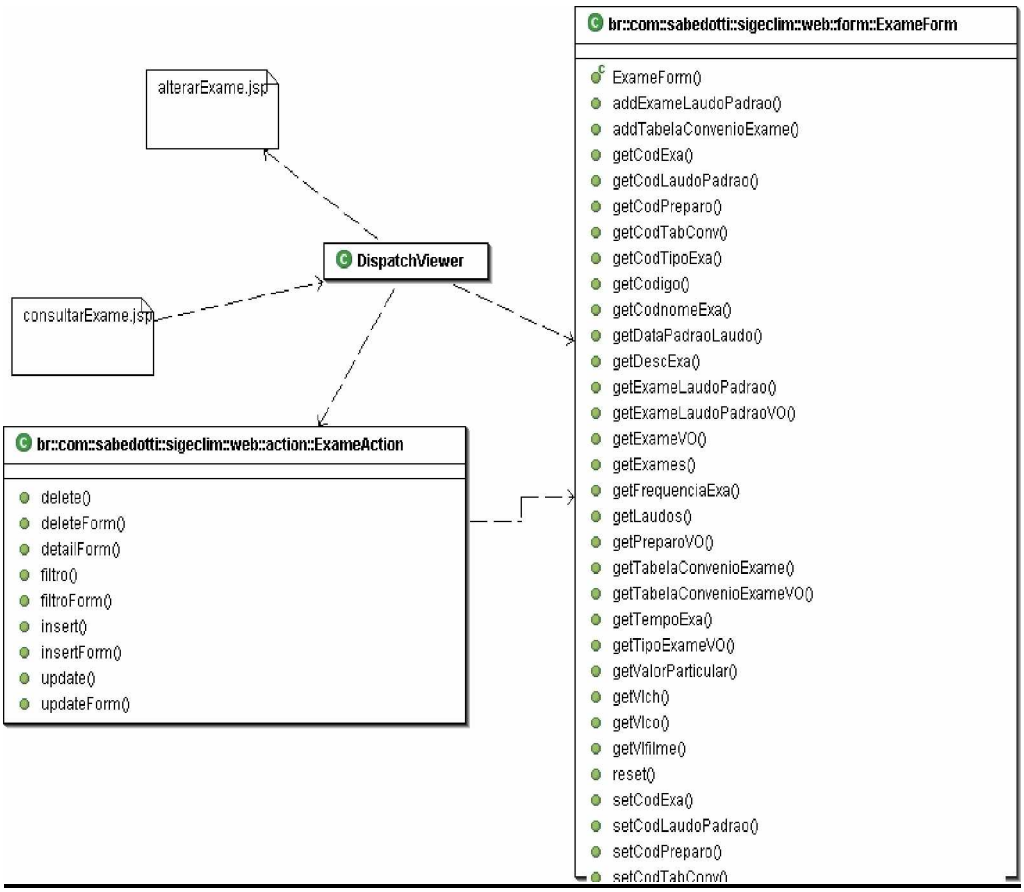

Figura 26 Padrão *Dispatcher View* na aplicação.

Seqüência de ações

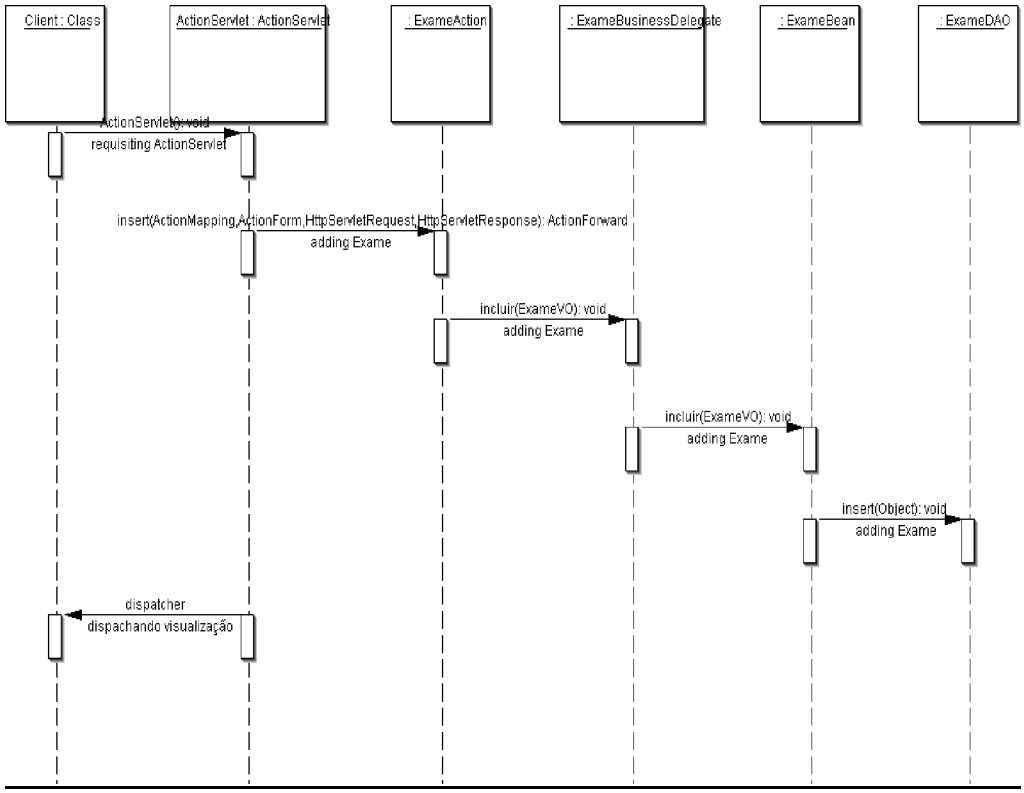

Figura 27 Seqüência de ações do padrão *Dispatcher View* na aplicação.

O diagrama de seqüência apresentado na Figura 27 demonstra o fluxo realizando para a inclusão de um exame.

5.1.6 Business Delegate

Um objeto no lado cliente oculta a interação específica do componente referente à regra de negócio da aplicação (EJBs : Session Façades), e expõem estes métodos de uma forma local para o cliente (Figura 28).

Atrás de uma requisição e um cliente, sempre existirá um cliente de uma regra de negócio específica. Isto poderá ser realizado de diversas formas. Umas das formas mais eficientes, e que ajuda no desacoplamento da aplicação, é a construção de uma camada de código responsável por mapear a “verdadeira” regra de negócio da aplicação. A partir do momento em que uma *Business Delegate* é utilizada, os desenvolvedores de uma aplicação conseguirão obter maior flexibilidade, tanto na

manutenção da camada de apresentação da aplicação, quanto na regra de negócio, propriamente dita.

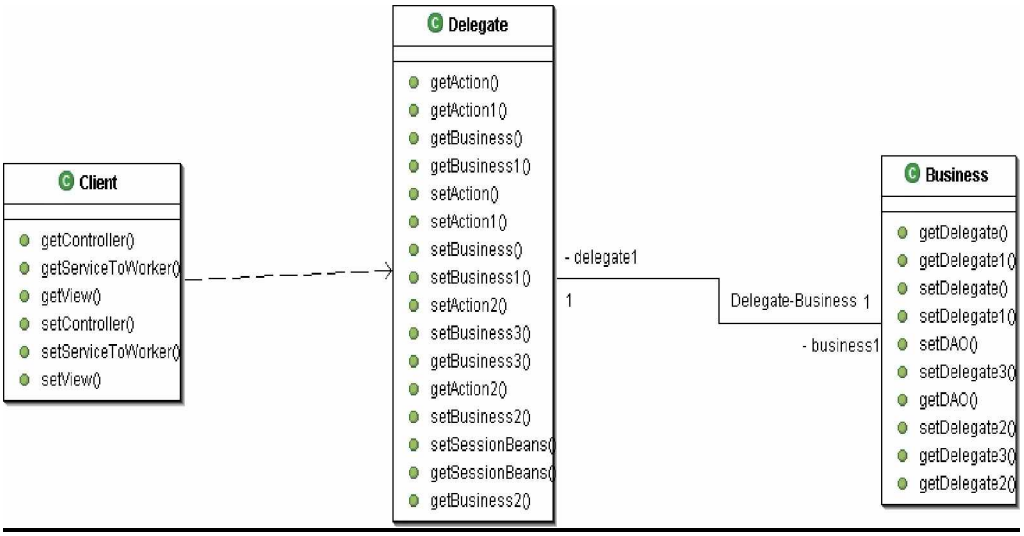

Figura 28 Estrutura do padrão *Business Delegate*.

A camada de código *Business Delegate,* como demonstrado abaixo (Figura 29), não é específica de uma camada de negócio, pelo contrário, ela serve justamente para tirar a responsabilidade e dependência de uma regra negocial em relação a uma camada de apresentação, onde o contrário também é verdadeiro.

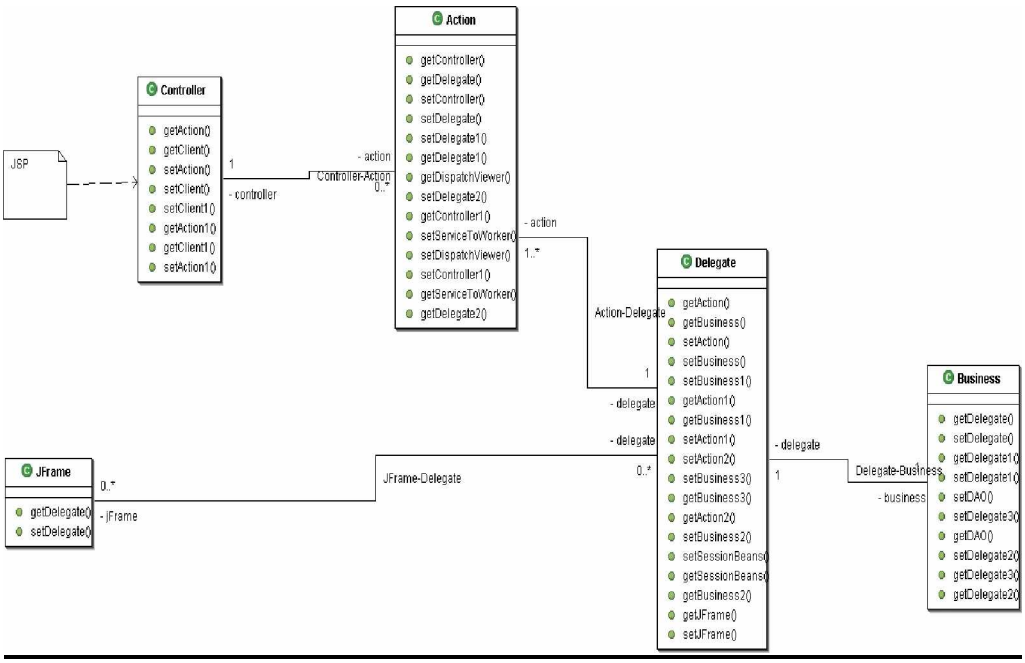

Figura 29 Detalhamento da estrutura do padrão *Business Delegate*.

Nesta situação, existem duas classes chamadas “PacienteBusinessDelegate” e “ConvenioBusinessDelegate”, que mapeiam, respectivamente, a classe “PacienteBean” e ConvenioBean, que possuem por sua vez, a “verdadeira” regra de negócio da aplicação.

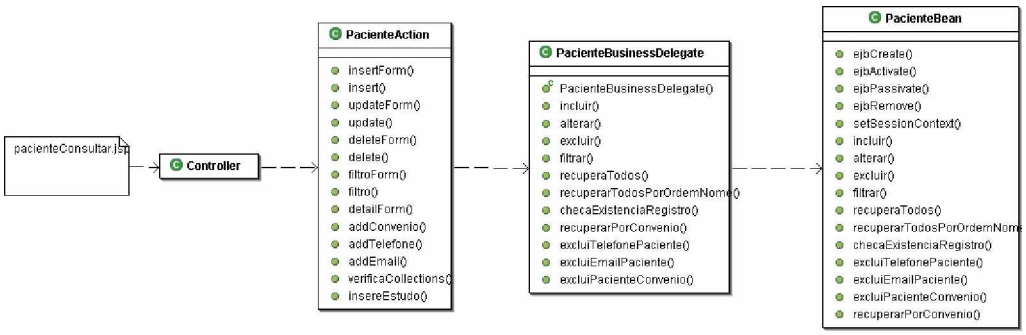

Figura 30 Padrão *Business Delegate* na aplicação.

Estas classes possuem um importante papel de representar a regra de negócio e abstrai-la para as ações do programa decorrentes de requisições de usuários. Elas são responsáveis por criar uma espécie de “ponte” para manter a regra de negócio independente da camada de apresentação, e também a apresentação independente de uma regra de negócio específica (Figura 30).

Seqüência de ações

Toda e qualquer chamada a uma regra de negócio na aplicação, passa antes por uma chamada a uma classe de delegação de negócio, que abstrai essas chamadas (Figura 31).

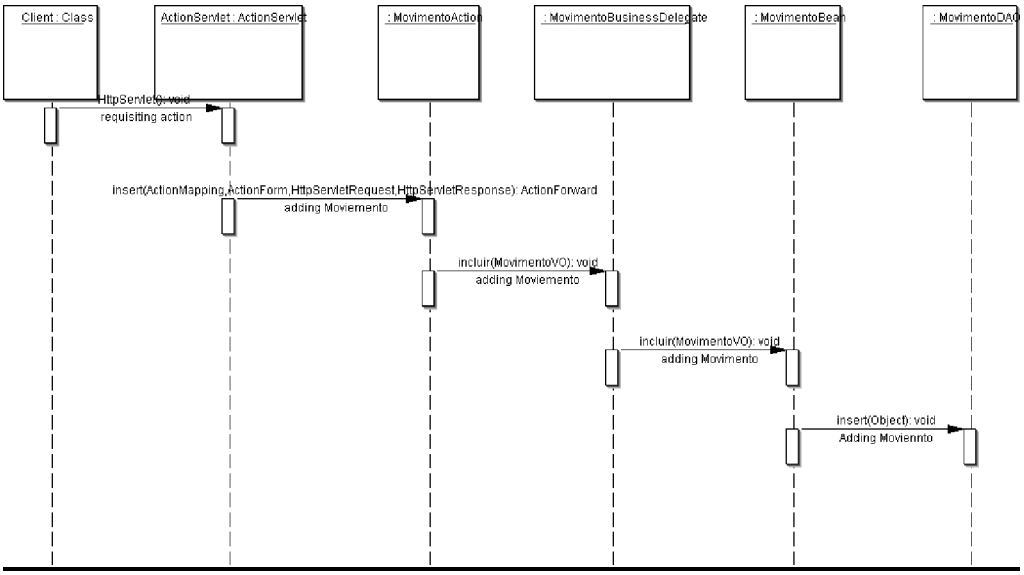

Figura 31 Seqüência de ações do padrão *Business Delegate* na aplicação.

Um dos fatos importantes da aplicação deste padrão é tornar a aplicação flexível em relação à camada de apresentação e a regra de negócio, pois caso seja necessário, o usuário poderá optar por utilizar qualquer tipo de apresentação para o sistema, que poderá ser através de páginas na internet, aplicativos *desktop* ou até mesmo apresentar o sistema em dispositivos móveis.

5.1.7. Service Locator

Um objeto auxiliar no lado do cliente coloca em *cache* as interfaces base de um componente de negócio freqüentemente utilizadas. Como declarado anteriormente, uma *Business Delegate* mapeia as ações de um objeto real de negócio. A classe *Business Delegate* possui em seu construtor uma chamada a esta classe, comportando-se como um *Proxy*. Toda vez que uma classe que delega funcionalidades a uma classe de negócio é inicializada, ela faz uma busca na rede pela sua classe de negócio específica, para poder usufrui seus métodos de negócio (Figura 32).

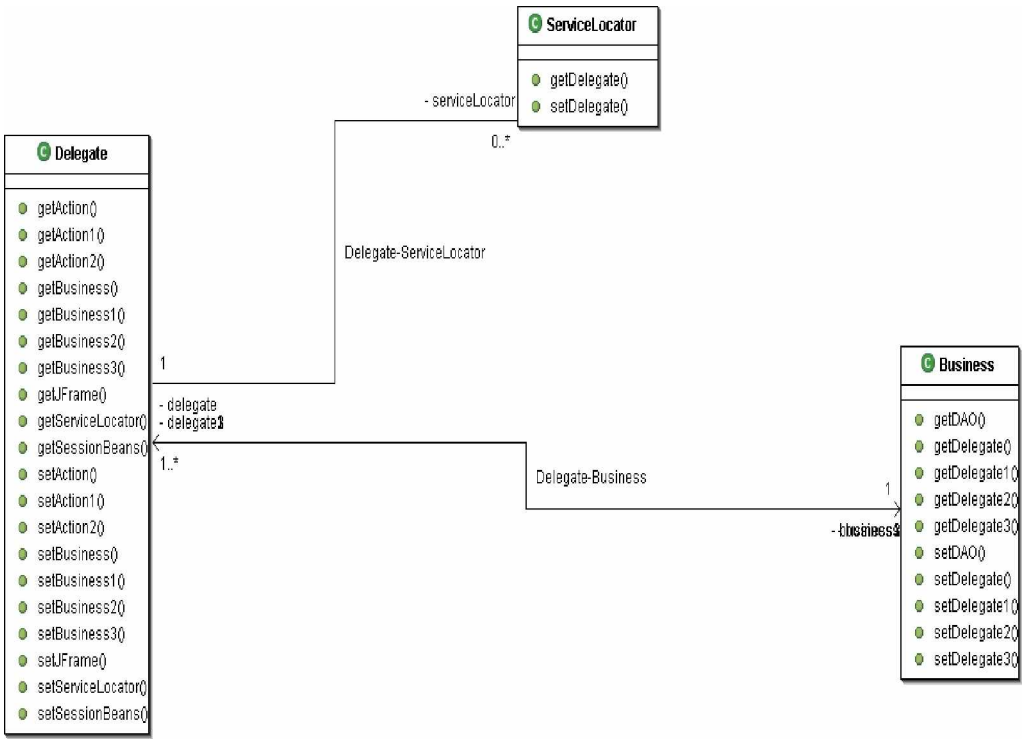

Figura 32 Estrutura do padrão *Service Locator*.

A classe ExameBusinessDelegate mapeia as regras de negócio da classe ExameBean. Para poder utilizar seus métodos, esta classe deve fazer uma busca na rede, pois se trata de uma classe distribuída, e conceitualmente sua localização naquele momento não é conhecida.

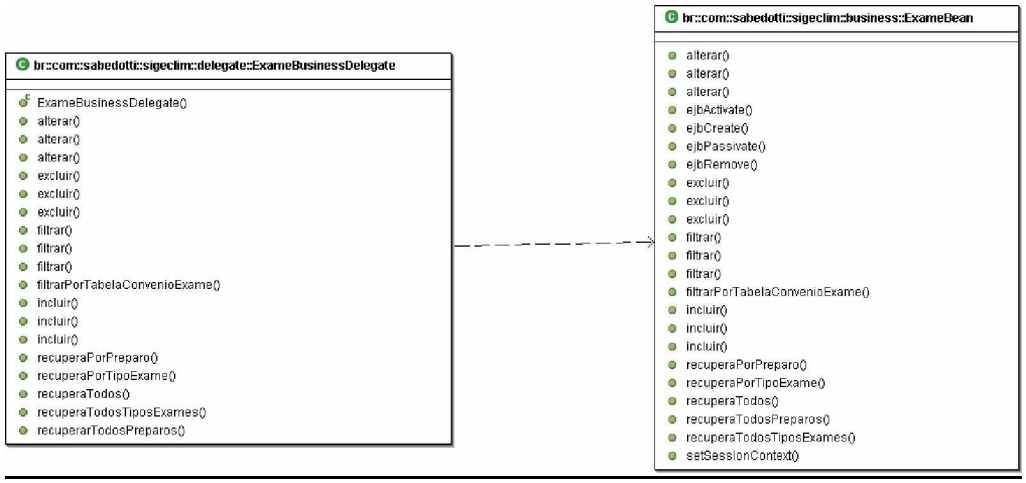

Figura 33 Padrão *Service Locator* na aplicação.

Toda vez que uma instância da classe ExameBusinessDelegate, é criada, automaticamente, uma busca é feita para localizar este objeto de negócio distribuído (Figura 33).

Seqüência de ações

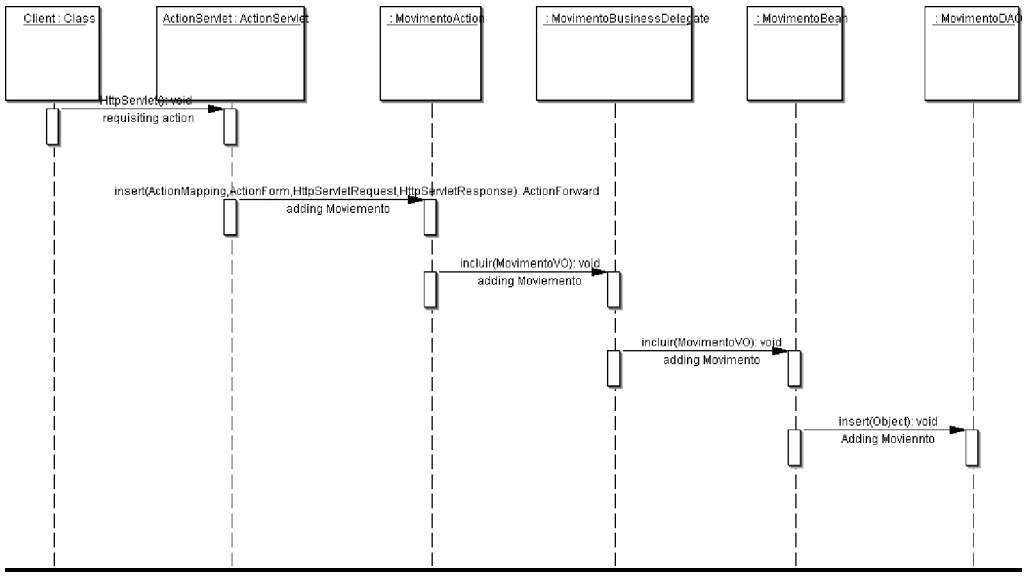

Figura 34 Seqüência de ações do padrão *Service Locator* na aplicação.

Como este objeto de negócio, na aplicação, é um objeto distribuído e controlado por um *Container Manager,* a classe que delega ações para o Session *Bean* deve ser capaz, primeiramente, de encontrá-lo na rede para depois conseqüentemente invocar suas regras, como é apresentado na Figura 34.

5.1.8. Value Object

Fornece uma foto (um *snapshot*) dos dados para ser usado como um lote de dados (objeto de valor) conveniente entre cliente e servidor, (evitando assim o tráfego excessivo de rede).

Este padrão foi criado para facilitar o transporte de informações. A partir do momento em que um objeto de valores existe, é necessária uma classe de negócio para interpretá-lo e transformar seus dados em informações úteis para a aplicação.

Quando este objeto é preenchido com dados, ele deverá percorrer as camadas da aplicação, para a geração de resultados (Figura 35).

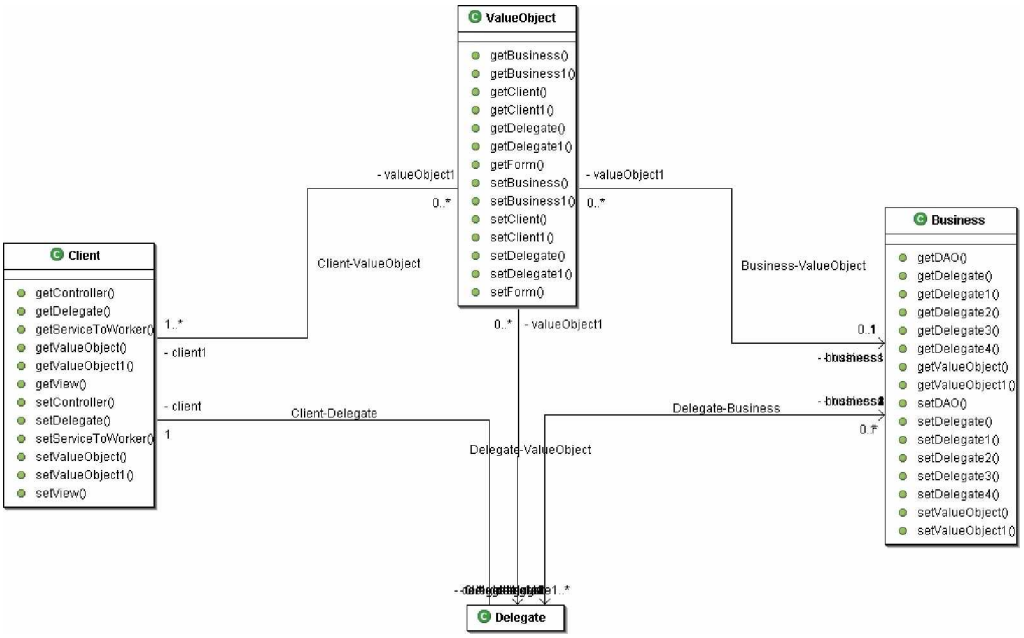

Figura 35 Estrutura do padrão *Value Object.*

Aqui é demonstrada a situação de cadastramento de um paciente. Os dados do paciente são digitados na camada de apresentação *WEB*, e os dados são populados em um objeto de valor e transferidos para demais camadas da aplicação até que finalmente possa ser persistido, depois de passar por diversas etapas, entre elas validação do seu conteúdo. (Figura 36).

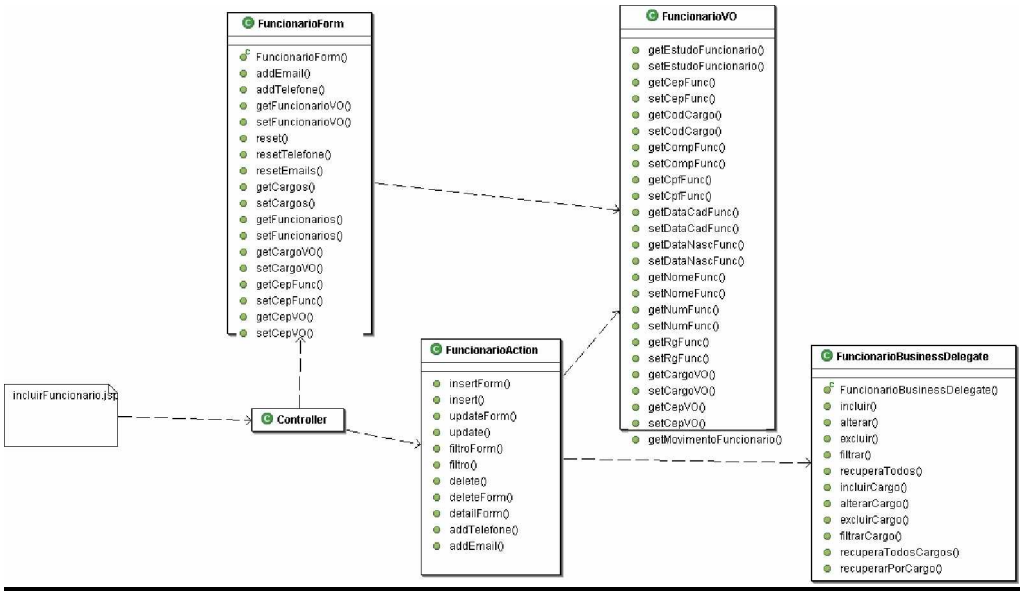

Figura 36 Padrão *Value Object* na aplicação.

Um objeto que pertencente ao padrão *Value Object,* tem como principal função mapear os dados digitados pelos clientes em uma aplicação e posteriormente trafegá-los pela rede, confrontando-se com as mais diversas camadas da aplicação.

Seqüência de ações

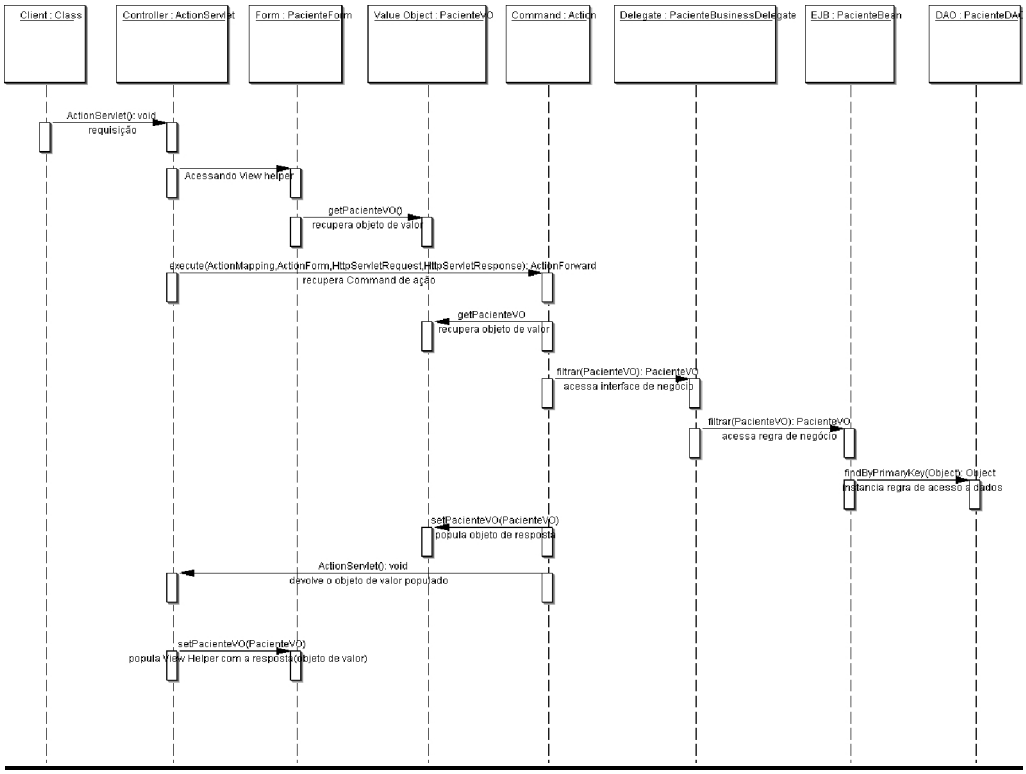

Figura 37 Seqüência de ações do padrão *Value Object* na aplicação.

Um *Value Object* é popularmente conhecido como "VO", ele deve ser constituído apenas com atributos e métodos de acesso.

5.1.9. *Commands*

Toda requisição é transformada em um objeto do tipo *Action*. A requisição do tipo *WEB* será redirecionada (através do *Controller: Service to Worker)* para uma ação específica da aplicação. Neste caso, as classes *actions* da aplicação irão responder (Figura 39).

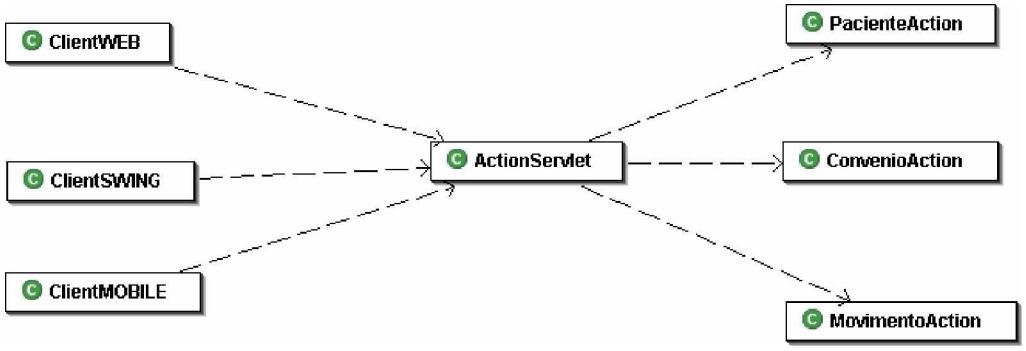

Figura 38 Estrutura do padrão *Command*.

Nesta situação, uma requisição é feita antes de os dados chegarem a regra de negócio da aplicação, ela passa por uma ação específica, um *Action* (*Command*) que irá identificar qual *Business Delegate* está associada com aquela requisição (Figura 39).

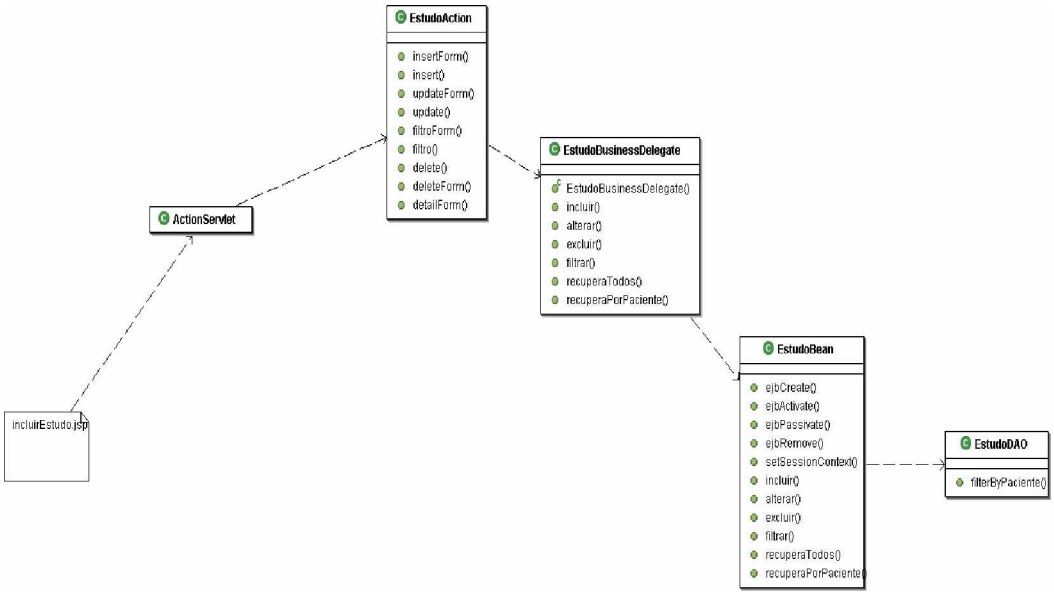

Figura 39 Padrão *Command* na aplicação.

Toda a aplicação está estruturada na seqüência de ações definidas na Figura 39. Pode-se observar que qualquer ação que seja executada na aplicação é transferida para uma *ActionServlet*, que é a peça fundamental neste tipo de desenvolvimento, que é a responsável pela navegabilidade do sistema.

5.2. Padrões da camada de negócio

5.2.1. Session Façade

Um EJB de sessão fornece uma fachada (*Façade)* para isolar os componentes de acesso aos dados (*DAOs*) do acesso direto do cliente.

Como visto anteriormente, a *Business Delegate* mapeia a verdadeira regra de negócio da aplicação, deixando a aplicação em uma situação mais confortável para ser estendida e especializada e com baixo acoplamento. Um *Session Façade*

contêm essa regra mapeada por uma *Business Delegate*, e que podem ser representados pelos *Session Beans* em uma aplicação.

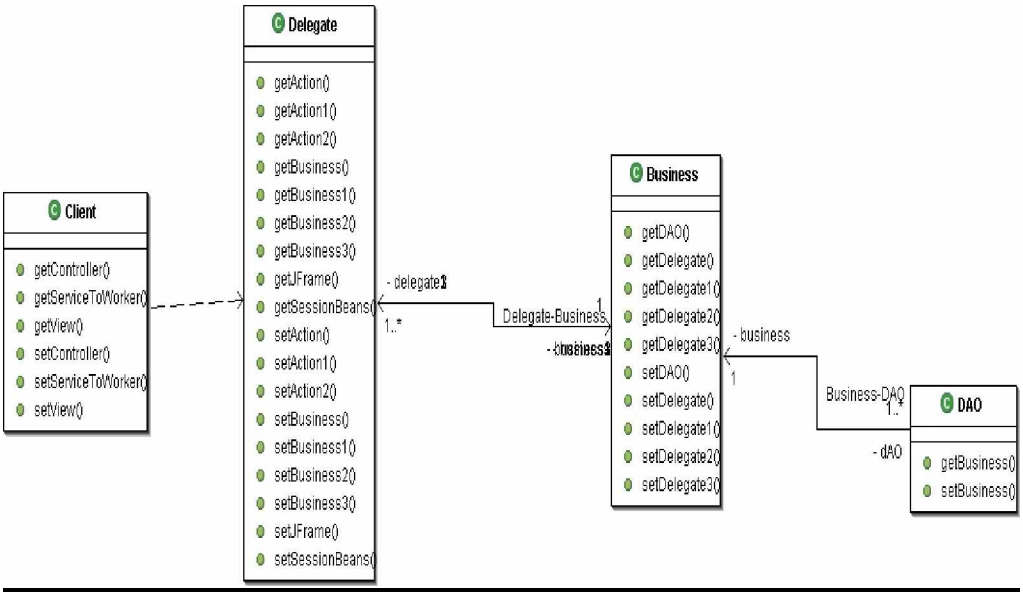

Figura 40 Estrutura do padrão *Session Façade*.

Eles são considerados os principais componentes do projeto. São eles que contêm a regra de negócio do projeto (Figura 40).

Seqüência de ações

A partir do momento em que uma ação é gerada na aplicação, ela irá passar por uma *Business Delegate* que irá "delegar" suas atividades para a verdadeira regra de negócio da aplicação, onde as regras serão executadas e aplicadas. A execução de um *Session Bean* é baseada na ação de uma tecnologia de persistência de dados.

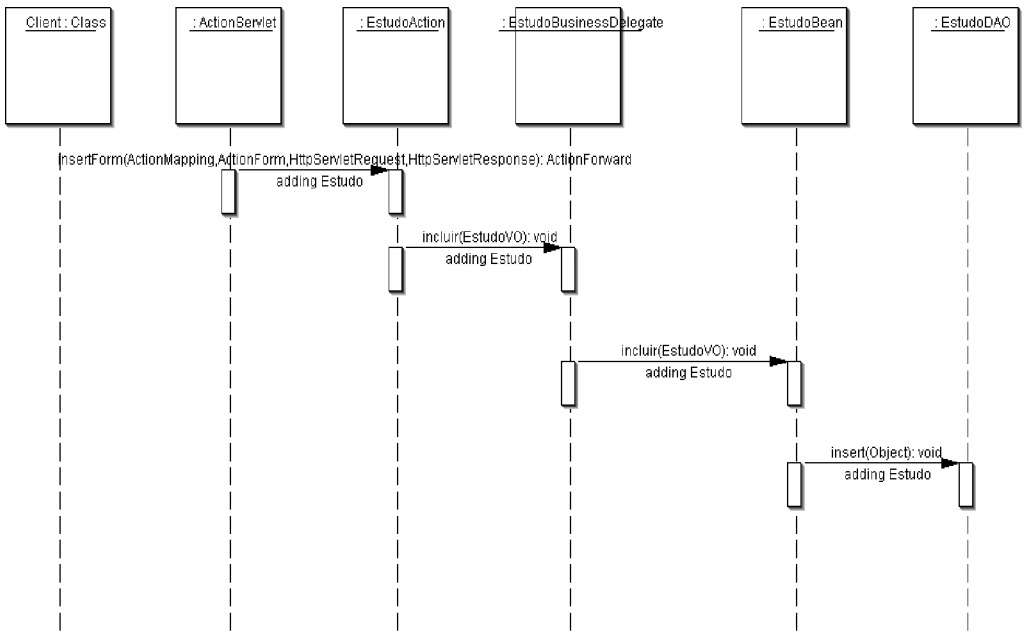

Figura 41 Seqüência de ações do padrão *Session Façade* na aplicação.

Neste projeto utiliza-se o conceito ROM (*Relation Object Model*: Modelo Objeto Relacional), e é aplicado um *Framework* específico que é utilizado para fazer a persistência dos dados na forma objeto-relacional (Figura 41).

5.2.2. Padrões da camada de persistência

5.2.2.1. Data Access Object

Um *Data Access Object* é utilizado para mapear a lógica relacionada à persistência de dados da aplicação.

As DAOs encapsulam o acesso a dados por trás de uma *interface* que pode ser implementada de diferentes maneiras para diferentes fontes de dados. Normalmente utilizadas para acesso a dados em *Servlets* e EJBs de sessão para encapsular acesso direto ao banco de dados (Figura 42).

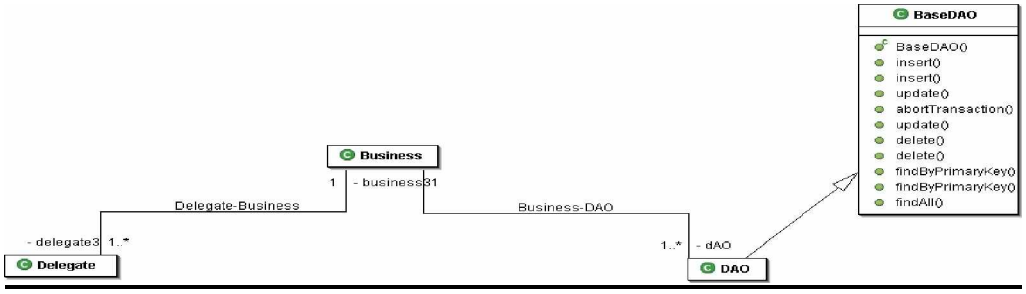

Figura 42 Estrutura do padrão *Data Access Object*.

Este sistema utiliza o conceito MOR (Modelo Objeto Relacional) para persistir seus dados. Um *Framework*, específico para esta característica é utilizado para mapear via objetos XML, os atributos de uma base de dados PostgresSQL.

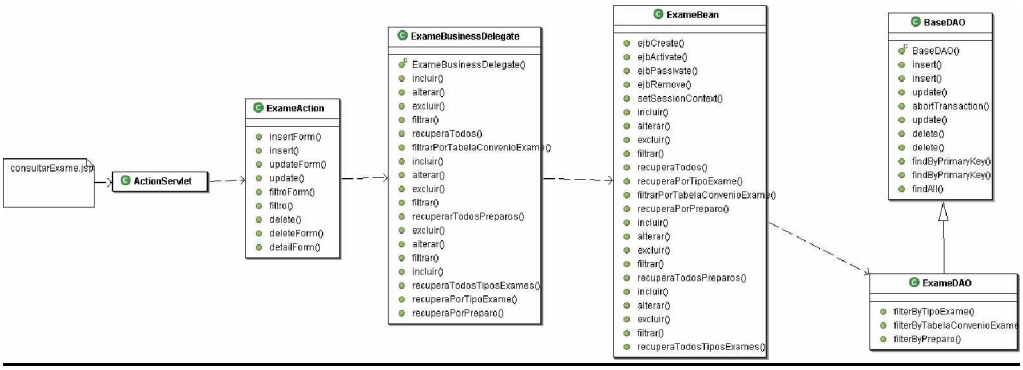

Figura 43 Padrão *Data Access Object* na aplicação.

Uma classe DAO irá possuir implementação referente a utilização deste Framework para persistir, recuperar e atualizar os dados provenientes de uma requisição de uma regra de negócio (Figura 43).

Seqüência de ações

Depois que a regra de negócio propriamente dita for executada, é a vez de uma classe "DAO" (*Data Access Object*) entrar em ação.

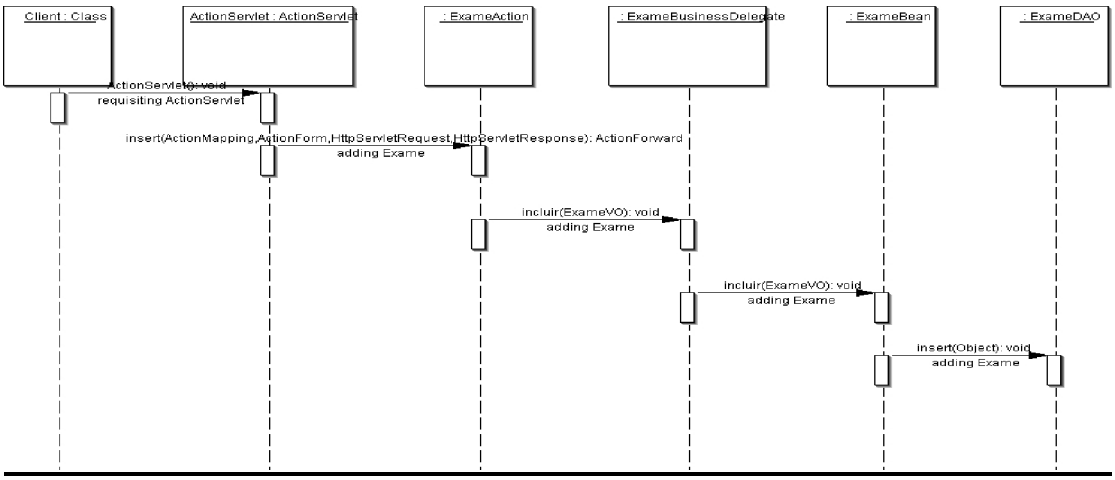

Figura 44 Seqüência de ações do padrão *Data Access Object* na aplicação.

Ela deverá pegar o objeto que recebe como argumento e tratá-lo de acordo com os métodos de negócio que foram invocados na regra de negócio, na qual aciona esta classe (Figura 44).

5.3 Dificuldades Encontradas:

O desenvolvimento de software é uma tarefa que exige dedicação e constante atualização técnica e tecnológica. Com a utilização de Padrões esta tarefa tornou-se mais sistematizada e organizada, porém exige do desenvolvedor a aplicação total a normas e definições de terceiros.

Na maior parte dos casos, já existem sistemas implementados que de uma forma ou de outra suprem as necessidades, quando é necessária a atualização dos mesmos há sempre a comparação com aquilo que já está funcionando, a elaboração e implementação de um novo modelo de software insere nas organizações um grau muito grande de insatisfações e incertezas.

As principais dificuldades enfrentadas na realização deste trabalho estão relacionadas à adoção de técnicas de desenvolvimento de software extremamente atualizadas e que não dispõem ainda de documentação clara e objetiva, exemplos práticos e elucidativos, massa crítica suficientemente formada e treinada.

Como o principal objetivo do trabalho é a criação de software para a área médica, outro fator problemático foi que a conexão entre os equipamentos de radiodiagnóstico por imagens, servidores de banco de dados e aplicação são ainda insipientes, em clínicas do porte das encontradas neste estudo, não contando com documentação adequada e padrões seguidos de fabricante para fabricante.

A equipe de desenvolvimento contou, na sua totalidade, com acadêmicos do curso de Tecnologia em Sistemas de Informação da unidade do UTFPR de Ponta Grossa, o que ocasionou atraso significativo no cronograma do projeto, pois houve a necessidade de se conhecer toda a tecnologia envolvida para depois entrar em ritmo de produção do sistema. E quando a equipe estava afinada e em ritmo bom de trabalho havia a necessidade de troca pois os mesmos acabavam por concluir seu curso, e como adquiriam alto grau de desenvolvimento o mercado absorveu.

A empresa foco passou por várias mudanças, desde a reestruturação de seu quadro de funcionários até a implantação de novos equipamentos e serviços, isto implicou na reorganização do setor de informática e de desenvolvimento.

No início do projeto foi decidido pela adoção da tecnologia denominada de CORBA/Med, como esta tecnologia na maior parte de suas implementações não são baseadas no conceito do software livre, a mesma foi descartada e utilizada a Tecnologia J2EE, ou seja, o tempo de aprendizado e gastos com treinamentos foram descartados.

As principais empresas fabricantes de equipamentos de radiodiagnósticos por imagens ainda não dispõem de pessoal técnico de campo preparado para a interconexão de seus equipamentos com as redes do padrão *ethernet*, dificultando o acesso às imagens no formato DICOM, que é um dos objetivos principais do trabalho.

A limitação orçamentária para projetos desta natureza é um fator que leva a adoção de práticas que atrasam sobremaneira a execução do cronograma, pois inviabiliza na maioria das vezes pelo seguinte: treinamentos constantes, aquisição de material bibliográfico, participação em congressos técnicos e aumento da equipe de desenvolvimento.

5.4. Considerações Finais do Capítulo

Neste capítulo, descreveu-se como foi implementado o programa através da utilização dos padrões definidos para a utilização com J2EE. Esta descrição foi baseada na apresentação de exemplos de algumas classes do sistema.

Foi exemplificada também algumas seqüências de ações que descrevem o comportamento do sistema em relação ao seu fluxo de informação.

E, por fim, foram feitas considerações a respeito das dificuldades enfrentadas na realização do estudo, aonde foram levantadas os principais problemas.

No próximo capítulo serão apresentados os resultados obtidos no trabalho através da adoção dos padrões descritos anteriormente.

Capítulo 6

6. Resultados Obtidos

Após a adoção das diversas tecnologias descritas anteriormente, como: software livre, Java, J2EE, EJB, *Struts* entre outros, e aliada à implementação de padrões, foi possível desenvolver um modelo mais estável para a aplicação do que existe atualmente na empresa.

6.1. Estrutura do Sistema

Deste ponto em diante serão apresentadas as partes que compõem o software. A preocupação com a utilização de todas as tecnologias antes descritas levou ao desenvolvimento de um sistema flexível e de fácil manutenção, desta forma todo seu funcionamento está baseado em um navegador, da internet, e a primeira preocupação é com a segurança de acesso, baseada em criptografia, garantida pelos padrões definidos como SSL (*Secure Socket Layer*), conjunto de regras e procedimentos de segurança definidas por padrões internacionais e que é baseada em árvore hierárquica de segurança, conforme apresentado na Figura 45, que mostra a tela de *login* do sistema.

Figura 45 Tela de *login* do sistema.
Fonte: (SIGECLIM, 2005).

Após autenticação no sistema, o usuário habilitado poderá navegar através do menu principal, que conterá o fluxo operacional da empresa e está dividido em:

Recepção: módulo responsável pela entrada do paciente na clínica, é através deste módulo que serão agendados exames, pacientes serão cadastrados no sistema, serão definidas quais especialidades dos médicos que requisitam serviços de radiodiagnósticos e após confirmação do paciente para a realização do exame, que poderá ser feita através do telefone ou por mensagem SMS enviada ao paciente, o mesmo será registrado como exame. (Figura 46)

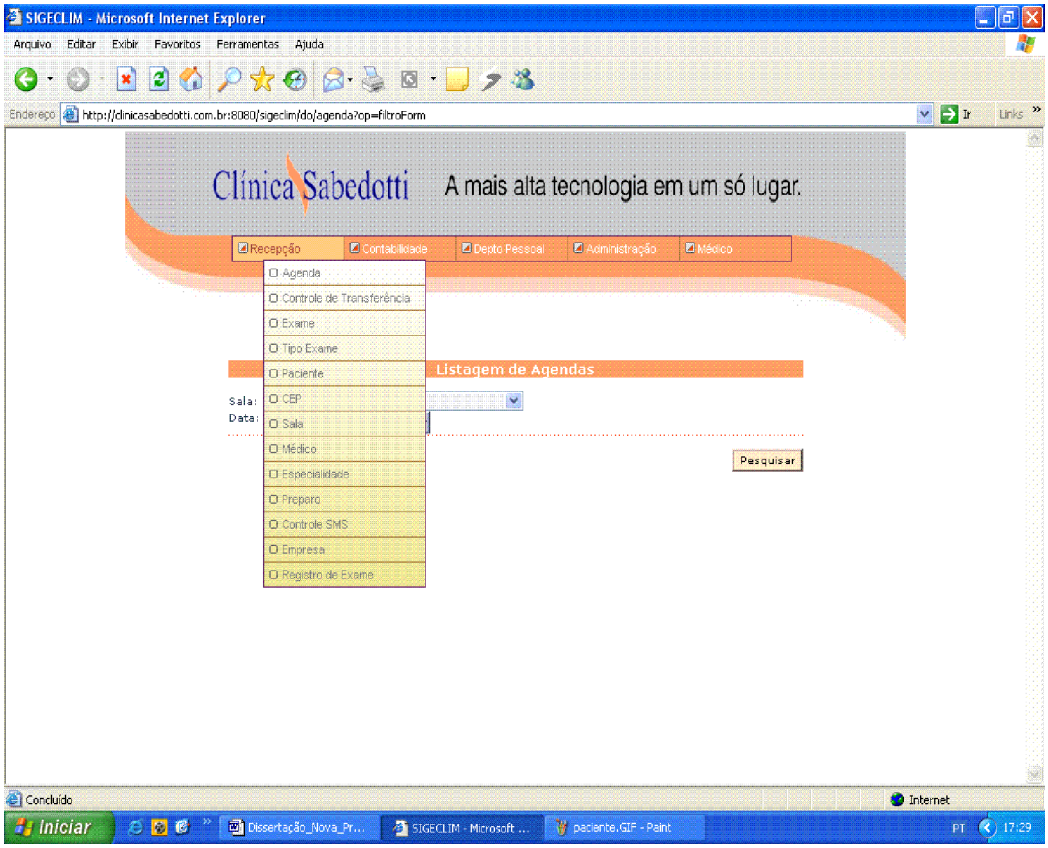

Figura 46 Módulo de Recepção
Fonte: (SIGECLIM, 2005)

Contabilidade: Módulo responsável pela centralização financeira da clínica. (Figura 47)

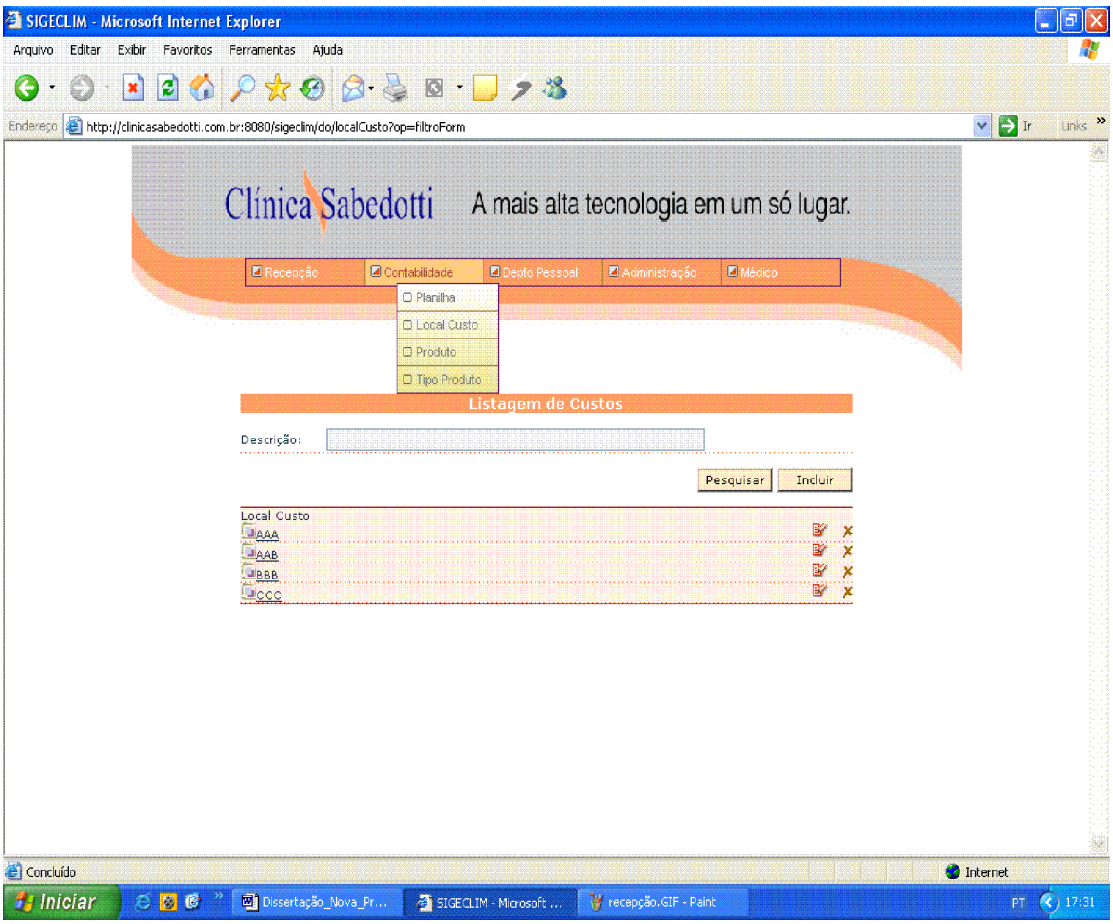

Figura 47 Módulo de Contabilidade
Fonte: (SIGECLIM, 2005)

Departamento de Pessoal: Este módulo é o responsável por organizar a empresa em relação aos seus funcionários, é através dele que os colaboradores da empresa receberão permissões de acesso ao sistema e terão todos seus dados cadastrais ali armazenados. (Figura 48)

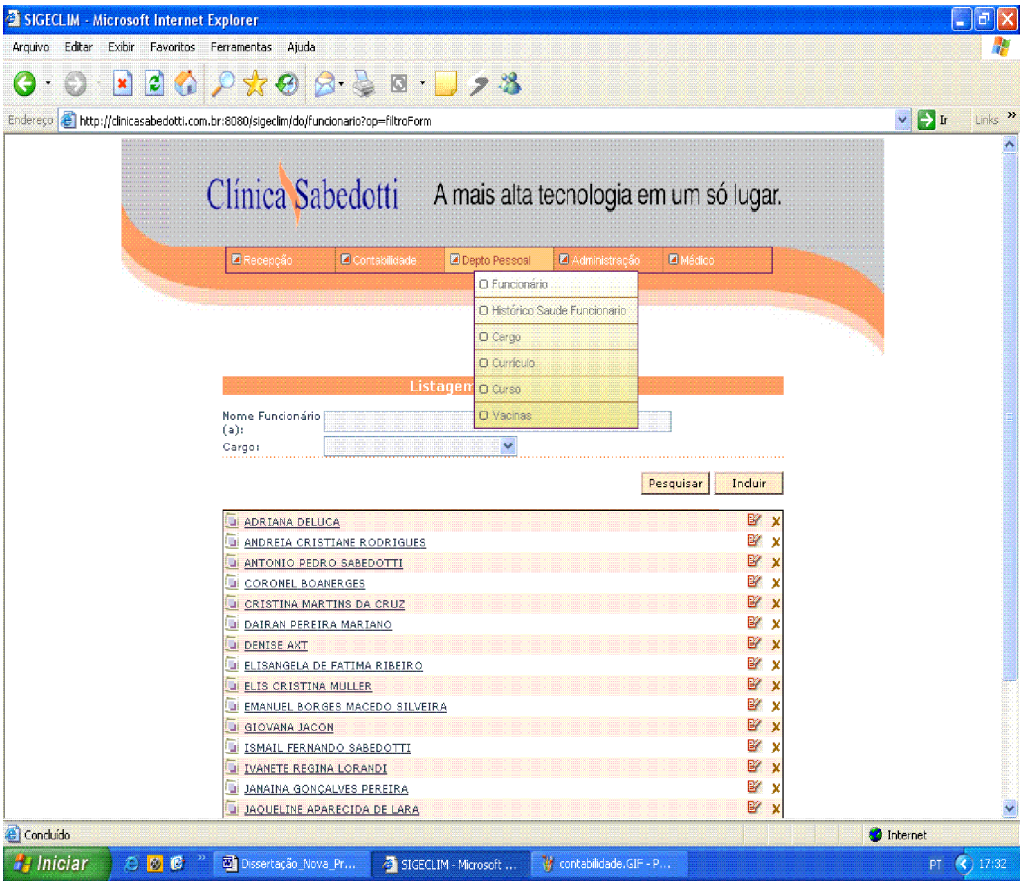

Figura 48 Módulo de Departamento de Pessoal
Fonte: (SIGECLIM, 2005)

Módulo de Administração: É neste módulo que serão realizadas as movimentações de fatura e controle de convênios da empresa, pois a grande maioria dos exames das clínicas de radiodiagnósticos por imagens são provenientes de convênios, desta forma há a necessidade de criação de um módulo específico. (Figura 49)

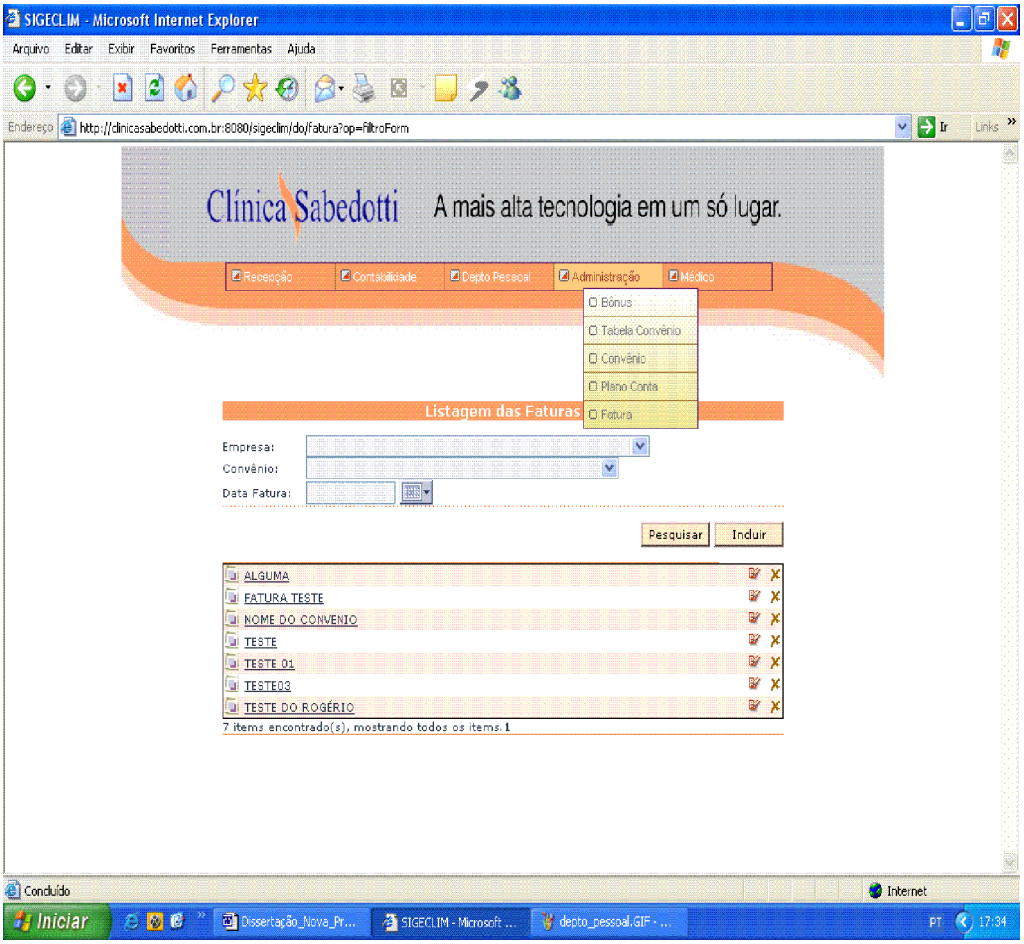

Figura 49 Módulo de Administração
Fonte: (SIGECLIM, 2005)

Módulo Médico: Este é o módulo principal do sistema, pois é nele que serão realizados os laudos, é através deste módulo que será realizada a união e organização dos dados dos pacientes (nome, idade etc), com os laudos referentes aos seus diversos exames e as imagens geradas nos equipamentos de radiodiagnósticos. (Figura 50)

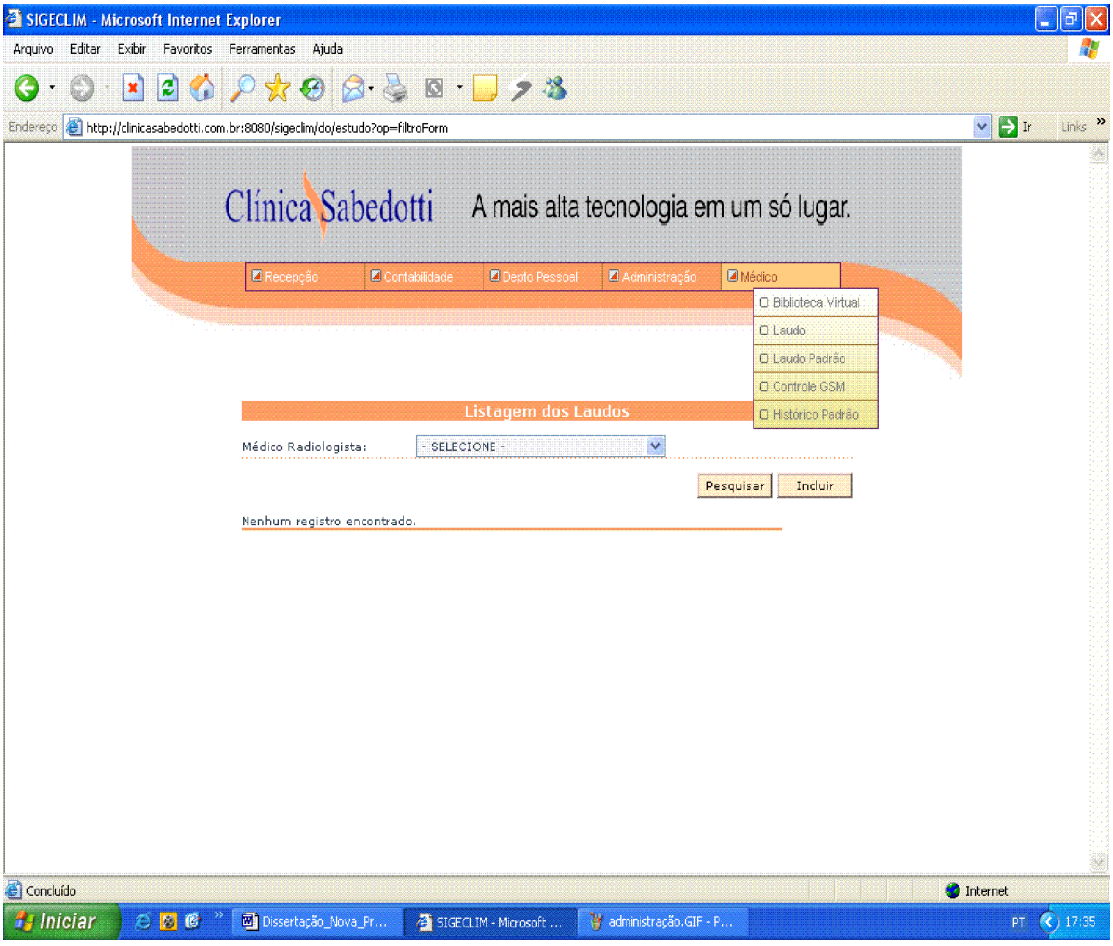

Figura 50 Módulo Médico
Fonte: (SIGECLIM, 2005)

6.2. VISUALIZAÇÃO DAS IMAGENS MÉDICAS – O JAVA APPLET

Com o sistema baseado em aplicações *Web,* a preocupação com a visualização das imagens armazenadas no servidor também tem de atender este requisito. Obviamente, se o problema fosse apenas a visualização destas imagens, a solução seria muito simples, afinal, páginas *Web* são formadas de textos, e também de figuras.

No entanto, para se chegar a um bom diagnóstico pelas imagens, o médico precisa dispor de algumas ferramentas de auxílio, como ajustes de contraste, brilho, *zoom*, ajustes estes que eles já possuem através de alguns aplicativos *desktop*, como o *eFilm* (MERGE HEALTHCARE CO.). O problema é que estas soluções *desktop* não oferecem a flexibilidade que a *Web* disponibiliza, acessando diretamente as imagens no banco de dados, por exemplo, ou permitindo o acesso de qualquer máquina em qualquer lugar através de uma conexão com a internet.

Para alcançar esta flexibilidade, foi pensado numa solução que, além de já ser bem difundida na *Web*, também dá seguimento à linha de desenvolvimento do projeto, por se tratar de uma tecnologia *Java*: o *Applet*.

6.2.1 Visualizando as Imagens Médicas

Um dos aspectos mais importantes do projeto é justamente a elaboração dos laudos médicos (através das imagens de radiologia) usando a flexibilidade que a plataforma *Web* proporciona. Uma vez que as imagens já estão armazenadas no banco de dados (como descrito anteriormente no capítulo sobre o servidor DICOM), o sistema é capaz de buscá-las normalmente como qualquer outra entidade do banco de dados, já que cada imagem possui seu código identificador e outros atributos que podem ser usados para filtrá-las e agrupá-las, conforme necessário.

Antes de entrar em detalhes do funcionamento do *Applet*, é conveniente uma breve descrição do cenário de elaboração de laudos, para uma melhor compreensão de algumas medidas tomadas na confecção do visualizador das imagens.

6.2.2 Controle dos laudos

Anteriormente, já foram apresentadas superficialmente algumas características do controle dos laudos, ainda que implicitamente. Voltando ao servidor DICOM, pode-se notar que não há nenhum campo explícito para identificar o paciente, ou a data em que a imagem (exame radiológico) foi gerada, por exemplo.

Estes e outros atributos importantes estão identificados através da entidade Movimento, que possui ligação com a entidade Imagem. Esta entidade Movimento não é nada mais do que o exame que foi feito pelo paciente, e guarda informações como a data/hora do exame, o exame que foi feito (tomografia, por exemplo), o médico solicitante, informações sobre faturamento (preço, convênio etc.) e, naturalmente, o paciente que fez o exame. De posse do relacionamento existente entre a Imagem e o Movimento, pode-se facilmente recuperar qualquer destas informações para serem apresentadas ao médico no momento da elaboração do laudo.

Da mesma maneira que a Imagem, a entidade Laudo também possui uma ligação (obrigatória) para Movimento. Assim sendo, no momento da inserção de um novo laudo, o médico deve apenas identificar a qual Movimento diz respeito este novo laudo.

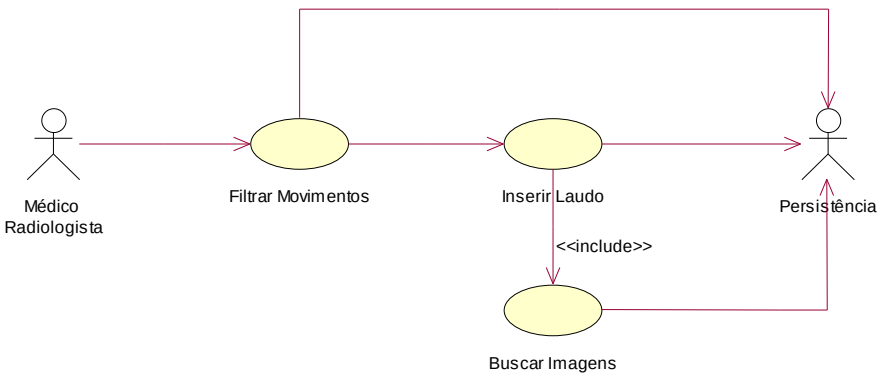

Figura 51 Diagrama use-case Inserir
Fonte: Adaptado de (MATOS, 2002, p.39)

O diagrama *use-case* acima (Figura 51) demonstra, de maneira simples, o funcionamento geral da inserção de laudos. A relação das entidades Movimento e Laudo, no banco de dados, são de cardinalidade 1:0..N (um movimento para

nenhum ou muitos laudos). Desta maneira, os médicos podem elaborar vários laudos para o mesmo exame (Movimento). A relação entre Movimento e Imagem se dá da mesma maneira, uma vez que as máquinas de radiologia geram inúmeras imagens do mesmo exame.

Depois de selecionado o movimento, o sistema busca no banco de dados as imagens relacionadas a ele. Estas imagens são agrupadas (na página de inserção de laudo) pelo atributo *SeriesDescriptor*, já descrito anteriormente. Este agrupamento pode ser observado na Figura 52.

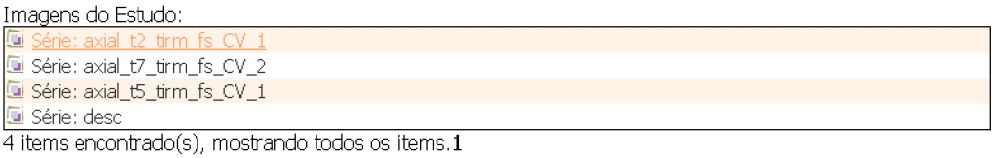

Figura 52 Listagem das imagens agrupadas pelo *SeriesDescriptor*
Fonte: (SIGECLIM, 2005)

Esta lista de *links* possui os dados necessários para carregar as imagens referentes ao movimento selecionado.

6.2.3 Carregando as imagens

Agora que as imagens estão agrupadas na página *Web* (obviamente que somente referências de código identificador para as imagens, por questões de desempenho), o *Applet* pode ser acionado para trabalhar com as mesmas.

Quando o médico escolhe o *SeriesDescriptor* (que está agrupando as imagens), o programa *Applet* é chamado. Como parâmetros, são passados para ele uma coleção de códigos (identificadores – chaves primárias) das imagens referentes a este *SeriesDescriptor* e ao movimento escolhido.

Dentro do método *init()* do *Applet* é instanciado um novo objeto do tipo *ImageViewer* (que estende uma *JFrame*). Assim, o usuário pode abrir quantos *Applets* forem necessários.

Por questões claras de desempenho, neste primeiro momento em que o *Applet* é aberto, apenas a primeira imagem é carregada e mostrada ao usuário, e é criada uma lista contendo todas as imagens (códigos identificadores) que foram passadas por parâmetros. Agora, esta lista é formada por uma coleção de objetos do tipo *ImagemCarregada*. Esta classe é responsável por fazer o controle para saber se

a imagem já foi carregada ou não, pensando novamente no desempenho. Ela possui três atributos: o código da imagem, a imagem JPEG (que inicialmente é um campo nulo) e um campo *boolean* que é mudado para *true* quando a imagem é carregada. Obviamente, se uma imagem já foi carregada, não há necessidade de recorrer novamente ao banco de dados para recuperá-la.

Obter um arquivo de imagem pode ser um processo lento, dependendo da velocidade da conexão e do tamanho da imagem. E neste meio tempo, enquanto a imagem é carregada do banco de dados para o *Applet,* é conveniente que o usuário tenha conhecimento do que está acontecendo. Para isso, fez-se oportuna a utilização de um componente para mostrar o progresso da tarefa de carregamento da imagem. O padrão adotado para controlar esta tarefa foi o *Observer.* Este padrão identifica dois objetos: o observador e o observado. Como os próprios nomes sugerem o objeto observador monitora o objeto observado, a fim de detectar qualquer mudança no estado deste último (MAGALHÃES, 2005), A tarefa de carregar a imagem é desempenhada por uma classe chamada de *ImageGetter*. Esta classe estende de *Java.util.Observable* e implementa *Runnable*. É o objeto observado, enquanto *ImageViewer* (implementa *Observer*) é observador, que controla a exibição da barra de progresso à medida que a imagem está sendo carregada pelo *ImageGetter*. Este objeto é instanciado por *ImageViewer* dentro de uma *thread*, impedindo que a janela do *Applet* (o *ImageViewer*) fique travando enquanto esta tarefa é executada. Obviamente, este controle é executado apenas nos casos em que a imagem ainda não foi carregada. Nos demais casos, a imagem é mostrada instantaneamente pelo programa.

No banco de dados, a imagem é armazenada como um vetor de *bytes*. O método responsável por buscar esta imagem no banco de dados e enviá-la para o *Applet* como um arquivo JPEG é o *getImage()*, da classe *DicomAction*, no servidor. Este método é acessado pelo *Applet* através de uma requisição HTTP, disponível em *HTTP://localhost:8080/sigeclim/do/dicom?op=getImage&codImg=78.* O acesso ao método fica protegido, uma vez que a segurança e restrições de acesso estão sendo tratados na aplicação.

Como se pode notar, o código da imagem é informado na chamada. De poder deste código, o método *getImage()* localiza a imagem correspondente, *contentType* (o tipo de resposta da requisição) para *image/jpeg*, e devolve o objeto

(imagem JPEG) por *outputStream*. De volta no *Applet*, esta imagem pode ser trabalhada como um objeto *Image* normalmente.

6.2.4 Trabalhando com as imagens

Uma vez que as imagens estão carregadas no *Applet*, faz-se necessário, para um diagnóstico mais preciso através da imagem, que sejam disponibilizadas algumas ferramentas de controle sobre elas. Ajustes como brilho, contraste, *zoom*, podem ser de grande auxílio para o médico na elaboração de um laudo.

Para prover estas funcionalidades, foi usada uma API *open-source* que já possui estes controles implementados. O projeto *E-view Box* (E-VIEW BOX PROJECT, 2005) é uma aplicação, também em *Java Applet*, que dá suporte a arquivos DICOM. Os motivos dela não ser usada completamente no projeto são algumas incompatibilidades com os propósitos do mesmo. Por exemplo, a falta do agrupamento das imagens pelo *SeriesDescriptor*, e a maneira como as imagens são abertas por este aplicativo.

A implementação destas funcionalidades no *E-view Box* é feita através de objetos que implementam a classe *ImageFilter*, do pacote *Java.awt.image*. Cada alteração na imagem provoca uma chamada ao método *repaint()* do *Applet,* para que a atualização seja transmitida imediatamente ao usuário.

Para efetuar estas funcionalidades, o usuário usa controles do próprio *mouse*, clicando e arrastando com o botão esquerdo, ou opções posicionadas no menu da janela do *Applet*. Algumas adaptações também foram feitas na maneira como essas funcionalidades são acionadas, no sentido de facilitar a utilização do programa. Um exemplo disto é a navegabilidade entre as imagens utilizando o botão de rolagem, muito comum nos *mouses* atuais, ou através de teclas de atalho no teclado, ou mesmo pelo próprio menu do programa.

Esses requisitos foram elaborados em conjunto com o usuário do *Applet*, no caso, o médico responsável, de modo a estar adequando o sistema da melhor maneira possível às necessidades identificadas por ele, segundo o que os médicos já estão acostumados a utilizar e trabalhar em outras ferramentas e aplicações do tipo *desktop* para tal.

O sistema também guarda uma versão original da imagem que está sendo tratada, para poder ser utilizada através da opção de restauração, presente no programa.

A Figura 53 ilustra uma imagem médica sendo mostrada pelo *Applet*.

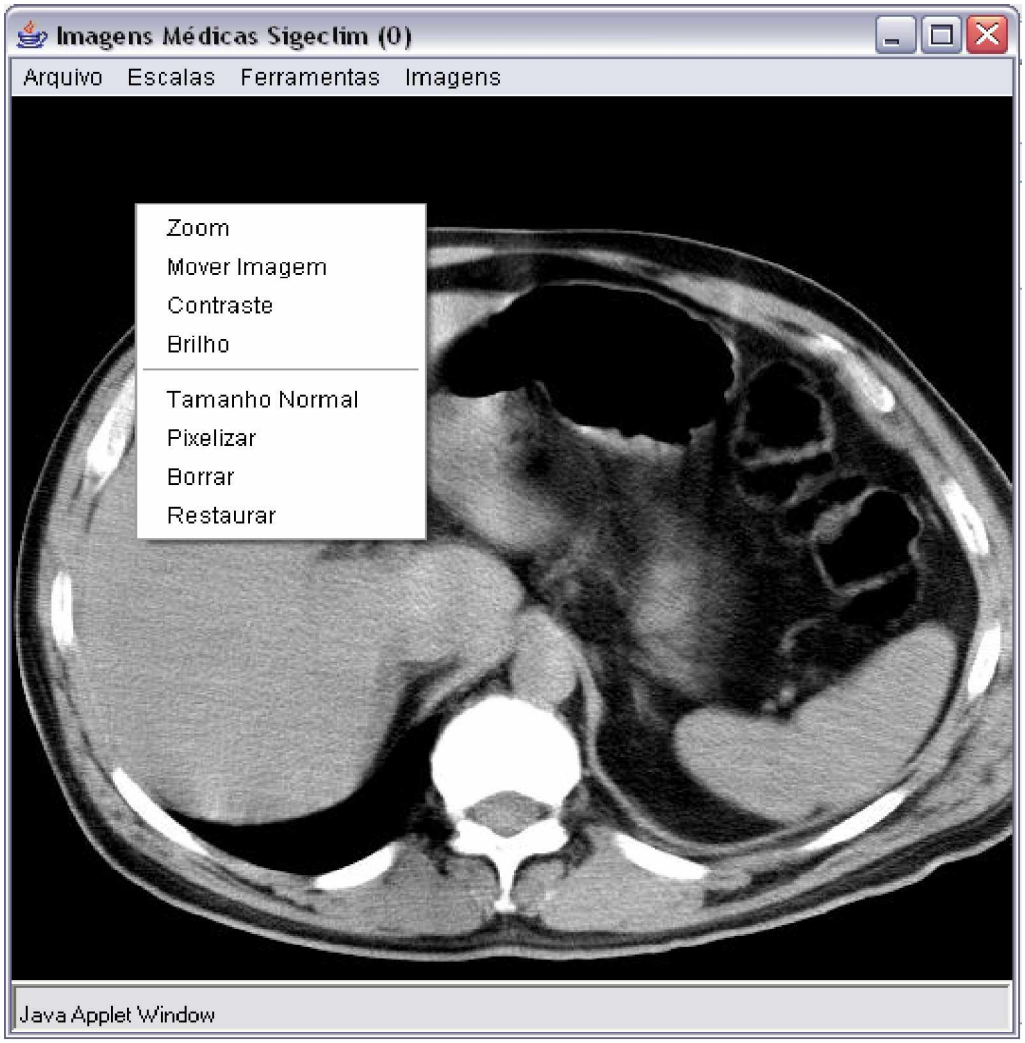

Figura 53 A janela do *Applet*

Na Figura 53, pode-se observar o menu resumido que se abre quando o botão direito do mouse é acionado. Logo acima, na barra de menus, estão dispostas todas as funcionalidades do programa. Outro detalhe está no menu Imagens, que possui as ações de avançar/retroceder e também uma listagem de todas as imagens referentes ao movimento e à *SeriesDescriptor* selecionados.

6.2.5 Problemas e Considerações Sobre a Confecção do Applet

O fato de se utilizar parte do projeto *open-source Eview Box* (E-VIEW BOX PROJECT, 2005) trouxe agilidade na implantação dos recursos de brilho, contraste, *zoom*, o que facilitou bastante o desenvolvimento do programa *Applet*, já que foi implementado utilizando recursos da classe de manipulação de interface gráfica *JFrame*, para manipulação de janelas, já empacotada com a distribuição *Java*. No entanto, algumas restrições e características próprias da tecnologia *Java Applet* tiveram que ser trabalhadas durante o processo.

Como já foi citado anteriormente, o *Applet* possui algumas restrições de segurança. Uma delas é que ele não pode ter acesso à base de dados diretamente. Para que isso fosse possível, seria necessário implementar a assinatura do *Applet*, e configurações nos arquivos de *policy*.

Com o objetivo de agilizar o processo de desenvolvimento, e até mesmo de separar esta tarefa de acesso ao banco do programa *Applet*, a lógica de comunicação e consulta ao banco de dados e o retorno da imagem para o cliente foi implementada ainda no servidor da aplicação. Assim sendo, o programa *Applet* apenas faz a chamada para uma *url* passando o valor identificador da imagem.

Esta *URL* invoca o método *getImage()* no servidor, que devolve a imagem encontrada na consulta para o cliente que a requisitou (o programa *Applet*). Poderia até mesmo ser citado o benefício de deixar o programa *Applet* um pouco mais leve, com a economia das linhas de código desta tarefa, no entanto, este não é o caso, já que esta economia é praticamente irrisória.

Um outro requisito de auxílio para a elaboração dos laudos médicos diz respeito à quantidade de imagens que o usuário pode visualizar simultaneamente, para fins de comparação entre um *SerieDescriptor* e outro do exame. Parece óbvio que a maneira de se fazer isso é apenas abrindo outra janela do programa *Applet*. No entanto, o funcionamento do *Applet* é vinculado à página que o chamou. Assim, da maneira como é geralmente implementado, quando ocorre o acionamento do *link* referente ao *SeriesDescriptor* a página é atualizada para comportar os novos valores de parâmetros (de acordo com o *SeriesDescriptor* escolhido). Mas essa atualização da página acarreta o fechamento da janela do *Applet* anterior (a página com os dados anteriores já não existe mais, apenas a nova, com os valores atuais). Depois

disso, obviamente, é aberto outro *Applet* com as novas informações. Para reverter esta situação, foi utilizada, para cada grupo de *SeriesDescriptor* da página, uma pequena janela *IFRAME* que corresponde ao seu respectivo *Applet*. Assim, o acionamento do link é direcionado para este *frame* (com o atributo *target* do *link*). Esta *tag IFRAME* especifica um local dentro da página principal que funciona como uma espécie de página à parte, onde pode ser manipulada qualquer outra *URL*. E é nesta página que os parâmetros do Applet estão sendo trabalhados, individualmente para cada registro de *SeriesDescriptor* que foi encontrado para o exame. Agora, o usuário pode abrir diversos *Applets* simultaneamente, e ir posicionando-os e redimensionando-os conforme lhe seja conveniente.

6.3 Servidor DICOM

A implementação do servidor de imagens médicas considera a adoção do padrão DICOM, que não apenas define o formato de armazenamento, mas também o protocolo de comunicação entre o servidor e estações clientes. Toda comunicação com o mundo exterior é realizada através da interface DICOM, que faz seu fluxo de dados sobre o protocolo TCP/IP conforme a especificação ACR-NEMA (2005). O servidor desenvolvido visa ser um centro de armazenamento e distribuição de imagens médicas dos exames feitos dentro e fora da clínica pelos equipamentos radiodiagnóstico. Toda a implementação do servidor foi feito usando *Java*.

6.3.1 Arquitetura do Servidor de Imagens Médicas

O servidor de imagens é peça central da nossa aplicação, ele que recebe as imagens e as monta no banco de dados em um formato legível para que possa ser publicado no servidor *Web*. O servidor DICOM foi implementado utilizando uma biblioteca de funções escrita na linguagem *Java* chamada de JDT (*Java* DICOM *ToolKit*) produzida pela empresa *SoftLink*

O objetivo de usar a API JDT é pelo fato de ela facilitar em vários aspectos. Principalmente na parte de comunicação entre as máquinas SCP e SCU, definido pela especificação ACR-NEMA (2005). Foi implementado o serviço de *Store SCP*, que no mundo do DICOM indica o envio das imagens médicas pelos equipamentos de radiodiagnóstico para o servidor permitindo seu armazenamento. Apesar do

padrão DICOM definir o meio e como deve ser armazenadas as imagens, essa fase foi especializada sua implementação para que pudesse ser integrado ao sistema gerencial da clínica.

O servidor de imagens fica disponível no sistema 24 horas por dia, quando um exame é feito, logo em seguida o médico/operador da máquina envia as imagens do exame para o servidor através de uma opção pré-configurada. O servidor de imagens as recebe e, através do protocolo busca o código do exame e a descrição da série, que está contida na imagem DICOM. As imagens são extraídas do objeto DICOM em formato JPEG com compressão, para que assim possa haver um melhor aproveitamento de espaço e recursos da rede, e então as imagens do exame são armazenadas no banco de dados com suas respectivas identificações.

Após as imagens estarem armazenadas de forma adequada no servidor de banco de dados, elas podem ser acessadas pelo servidor *Web*, que será descrito no próximo capítulo. O servidor mostra as imagens para que os médicos possam acessar de qualquer cliente HTTP e a partir delas gerar os laudos e estudos necessários sobre o exame.

6.3.2 Implementação do servidor DICOM

Com a utilização da API JDT para implementação do servidor de imagens médicas que utiliza o protocolo DICOM, o desenvolvimento foi facilitado, pois a mesma implementa a comunicação entre as máquinas. Sendo necessário desenvolver as rotinas para fechar os acordos com os equipamentos médicos (SCUs) e desmembrar o objeto DICOM para que possa ser armazenado no banco de dados.

O Servidor é composto por duas classes construídas pela equipe de desenvolvimento. A principal é a classe *DicomServer.class* que negocia os parâmetros com as SCUs e atribui as imagens dos exames feitos por elas. Para isso é muito usado o *ToolKit*. Primeiramente o servidor é iniciado em uma *thread* principal que fica disponível na porta 104 (porta padrão definido pela especificação DICOM), que aguarda conexões de máquinas radiológicas com exames pronto para serem enviados. As máquinas radiológicas fazem o papel de SCUs. Quando o operador da máquina faz o envio das imagens para o servidor, essas máquinas se conectam ao

servidor na porta 104 e enviam um objeto DICOM composto somente de um *DataSet* com informações necessárias de conexão. Nesse *DataSet* existe informações definindo que a requisição feita pela máquina é um *SotreSCP* e outros campos auxiliares que informam o tipo de compreensão da imagem e protocolo a ser usado na transmissão dos próximos dados (ACR-NEMA). O servidor lê esses dados e passa para API que monta objetos *Java* para que possa ser manipulado pelo sistema. O servidor identifica o tipo de requisição e se for compatível os protocolos solicitados pela máquina, ele envia uma resposta dizendo para que comece enviar as imagens. Se não, nega a associação e a conexão é encerrada, como ilustrado na Figura 54. A SCU recebendo uma resposta positiva inicia a compressão das imagens e monta os pacotes DICOM e os envia para o servidor que, por suas vez, monta os objetos DICOM.

Depois de feito isso, é chamada a classe responsável pelo armazenamento das imagens, e passado a ela os objetos DICOM (objetos Java).

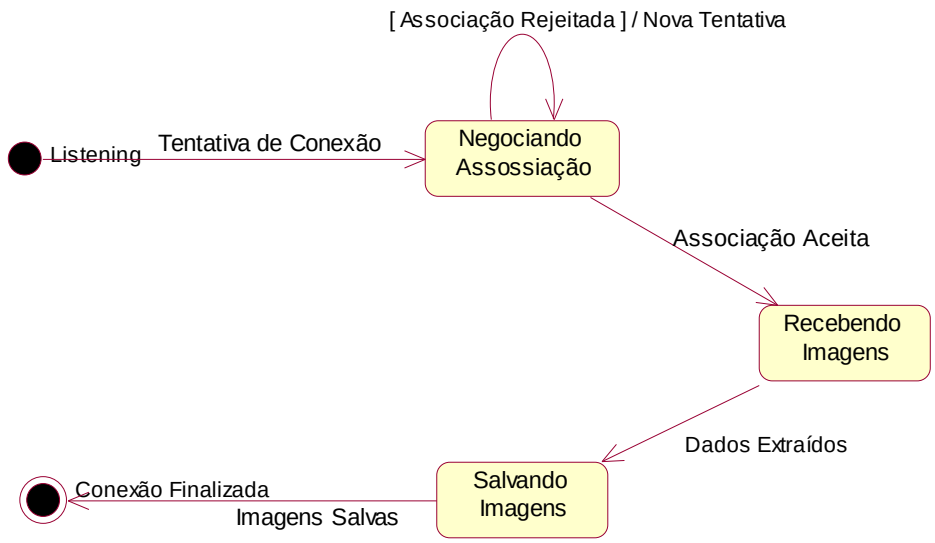

Figura 54 Diagrama representando o fluxo do Servidor DICOM
Fonte: (MATOS, 2002, p.88)

A segunda classe é a *SaveImage.class* (responsável pelo armazenamento) que tem a função de extrair as imagens em formato JPEG do objeto DICOM e armazená-las no banco de dados. Para isso, é necessário que nos objetos DICOM haja pelo menos dois campos obrigatórios preenchidos. Esse campo define o código

do movimento, que nada mais é do que o exame que foi feito pela máquina. Outro campo somente descritivo e não obrigatório é o *SeriesDescription* que é também extraído da imagem, pois alguns exames o utilizam para definir o corte[2] utilizado no exame. E o principal, e também obrigatório, é campo que contém as imagens relacionadas ao exame em questão. Por existirem várias máquinas usando protocolo DICOM e nem todas respeitarem o protocolo como a especificação padrão, o campo escolhido para carregar o código do movimento foi o mesmo campo do nome do paciente. Sendo assim, o campo *PatientName* foi montado da seguinte forma: A primeira parte é composta pelo código do movimento seguido de um espaço (*Caracter* ASCII 32) e a segunda parte é o nome do paciente. (Figura 55)

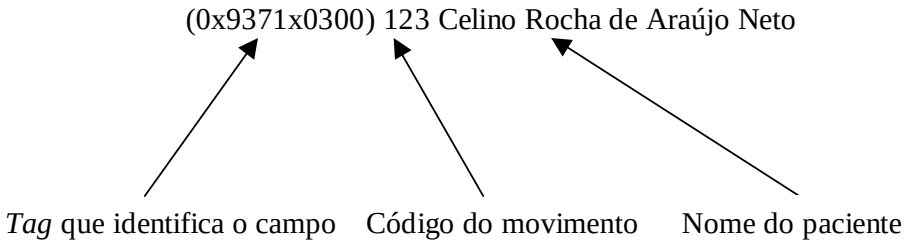

Figura 55 Campo *PatientName* do protocolo DICOM
Fonte: Autor.

O campo *SeriesDescription* não é enviado por todas as máquinas e por isso para uma melhor visualização posterior da imagem, foi definido que quando a máquina não enviasse o campo preenchido seria definido no banco de dados como "Não_Informado". O campo de imagem traz várias imagens em vários formatos, que são definidos no acordo de associação entre as máquinas que é feito para que estas possam se entender na troca de mensagens. Essas imagens são extraídas usando uma rotina que traz as imagens em formato *Java.awt.BufferedImage* que usando a *JPEGImageEncoder* são codificadas para o formato JPEG. Tendo em mãos o código do movimento, o descritor da série e as imagens, elas são, então, armazenadas no banco de dados.

Além dos problemas de as máquinas não seguirem a especificação DICOM ACR-NEMA V3.0 à risca e assim não permitir que fosse criado um campo

[2] Ângulo aplicado na extração da imagem (Radiology, 2003).

customizado para o código do movimento, foi encontrada alguma dificuldade com a implementação da classe *SaveImage.* Primeiramente usamos uma classe fornecida pela API que transformava as *BufferedImage* em JPEG, pois essa classe lançava erros de conversão na maioria das imagens. Para resolver isso foi usada a *JPEGImageEncoder* que é uma classe nativa da API padrão do *Java* que conseguiu converter todas imagens testadas sem nenhum problema.

Para evitar que o servidor seja derrubado por uma exceção ou entre em *DeadLock* por causa e uma operação mal sucedida, no servidor de imagens, cada transação é uma *thread* independente, ou seja, se uma conexão por qualquer motivo tiver lançado uma exceção nenhuma outra conexão ativa será afetada, e nem o serviço de espera de nova imagens será abalado.

6.4 Conclusão

Neste momento serão realizadas as reflexões sobre os objetivos propostos neste trabalho e comparados com os resultados obtidos, também serão analisadas as tecnologias adotas no seu desenvolvimento e como e qual o comportamento da modelagem e da implementação em face de tais tecnologias.

Como objetivo principal este estudo apresenta: Desenvolver e implementar um modelo de Sistema de Informação para otimização de exames de diagnósticos por imagens.

Segundo Terra (2001) “As informações e os sistemas de informações tem de ser precisos, personalizados e imediatamente disponíveis em um formato que facilite o espaço, e têm de estar em formato que facilite o uso”.

Desta forma, o modelo proposto deveria seguir os padrões da área médica e os padrões mais atualizados e utilizados em todo o mundo. Foi projetado então um modelo que fosse flexível, robusto e interoperável, ou seja, que pudesse rodar em qualquer plataforma de sistema operacional.

Como desafio, ainda havia a integração de equipamentos de radiodiagnósticos por imagens e a rede coorporativa da empresa. Após um levantamento das tecnologias disponíveis e visando à redução dos custos como desenvolvimento e implantação, foi adotada a utilização de software livre como padrão de desenvolvimento.

Toda a modelagem foi estruturada utilizando-se o padrão Modelo-Visão-Controle, para que o sistema fosse preciso. Este padrão imprime no desenvolvimento de software a organização e independência entre os diversos módulos. Para implementação do padrão foi utilizada a linguagem de programação JAVA, além de a mesma ser interoperável está fortemente orientada a aplicações para a internet, ou seja, deixando o sistema imediatamente disponível. Um dos grandes impactos de sistemas que armazenam imagens é o tamanho e a robustez da base de dados, pensando nisto foram utilizados diversos *Frameworks* como: J2EE, EJB, *Struts*, OJB entre outros, para que seu formato facilitasse o espaço, pois com a utilização destes é possível distribuir a aplicação, colocando partes do sistema em equipamentos distribuídos, devido ao grande volume de imagens trafegadas na rede, a tecnologia de objetos distribuídos é um fator fundamental para

o bom desempenho da aplicação, pois reduzirá o volume de dados tratado por cada equipamento distribuído.

A integração entre os equipamentos de diagnóstico e a rede coorporativa da empresa foi feita com a adoção do padrão DICOM, baseado no protocolo TCP/IP e disponível na maioria dos equipamentos de imagens, o protocolo tem se transformado em um padrão para a indústria médica.

Para a facilidade de uso da aplicação conta com páginas em HTML, que poderá ser configurada a qualquer momento pelo próprio usuário bastando escolher a melhor maneira de utilização de seu navegador, e até mesmo qual o sistema operacional será melhor para sua organização.

O grande desafio do trabalho foi desenvolver uma aplicação para área médica de baixo custo, pois as iniciativas analisadas são normalmente caras e estão integradas a sistemas completos de PACS, oferecidos por empresas como Fuji, Agfa e Philips. Tais aplicações são extremamente onerosas para clínicas do porte da estudada aqui.

Com a adoção do sistema proposto, projetos de implantação de PACS em pequenas organizações passam a se tornar viáveis, pois a ferramenta proposta pode ser integrada a qualquer tipo de equipamento de radiodiagnóstico por imagens, e os projetos podem ser desmembrados em módulos para cada segmento de exame.

Como a plataforma é interoperável, as clínicas podem optar por equipamentos de radiodiagnóstico de fornecedores diferentes, isso ajuda na redução de custo de aquisição dos equipamentos.

A telemedicina foi outro fator explorado no modelo, pois as informações clínicas dos pacientes poderão ser disponibilizadas a qualquer momento e em qualquer lugar, permitindo a criação da sala de laudos virtual e até mesmo buscando uma segunda opinião sobre um determinado diagnóstico.

Desta forma pode-se concluir que o modelo proposto segue os mais rígidos padrões de desenvolvimento e que os objetivos propostos foram alcançados através da implementação de um software interoperável e flexível, fornecendo aos usuários (médicos radiologistas, funcionários, pacientes e especialistas) informações seguras e instantâneas.

6.5 Trabalhos futuros

A área de telemedicina é extremamente promissora, e a partir desta iniciativa pode-se citar que este produto poderá ser utilizado em:

- Salas de laudos virtuais;
- Como este sistema ajudará na redução da poluição,
- Comportamento do sistema em relação ao uso de software livre;
- Implementação do conceito de *worklist*.

REFERÊNCIAS

ACR-NEMA. **Site Oficial do comitê de trabalho ACR-NEMA**. Disponível em <www.nema.org/nema/medical/dicom>. Acessado em: 27 de maio de 2005.

ALEXANDER, C. **Some notes on** Disponível em <http://math.utsa.edu/sphere/salingar/Chris.text.html>. Acessado em: 23 agosto 2004.

ALEXANDRINI, F. **Desenvolvimento de uma Metodologia de Interpretação, recuperação & codificação inteligente de laudos médicos independente de idioma**. Tese de doutorado. Florianópolis, 2005.

ALTER, S. L. **Information System: a managament pespective**. 2 Ed. , Melon Park: Benjamin Cumings. 1996.

ALTMAN, R. B. AI in Medicine: **The Spectrum of Challenges from Managed Care to Molecular Medicine**. Fall: AI Magazine, 1999.

BITTENCOURT, M. V. S. **Estudo Comparativo entre Frameworks Java para Construção de Aplicações Web – Universidade Federal de Santa Maria, RS (2004).** Disponível em <www-usr.inf.ufsm.br/~marvin/> Acessado em: 30 de junho de 2005.

BOND, M. et al.: **Aprenda J2EE com EJB, JSP, Servlets, JNDI, JDBC e XML em 21 dias**. São Paulo: MAKRON Books, 2003.

CARITÁ, E. C. **Visualisador de Imagens DICOM para um ambiente Hospitalar**. São Paulo. USP, 2004.

CHAVENATO, I. **Introdução à teoria geral da administração.** 6ª ed. Rio de Janeiro: Campus, 2000.

CHILD, J; SMITH, C. **The context and process of organizational Transformation: Cadbury Limited in its Sector**. Journal of Managament Studies, v.24, n. 6, nov. 1987.

COSTA, C. G. A. **Desenvolvimento e avaliação tecnológica de um sistema de prontuário eletrônico do paciente, baseado nos paradigmas da World Wide Web e da engenharia de software**. Campinas, São Paulo, 2001.

COSTA, H. F. D; et al. **Soluções Open source para Desenvolvimento de Aplicações Web.** São Paulo, 2004.

DESIGN PATTERNS. **Design Patterns Fundamentais do J2EE (2003)**. Disponível em <www.fundao.pro.br> Acessado em: 30 de junho de 2005.

DUPAS, G. **Alca e os interesses do Mercosul.** São Paulo: Edição Fundação Memorial da América Latina, 1997.

DAVENPORT, T. H. **Reegenharia de processos: Como inovar na empresa através da tecnologia da informação**. 5ª Ed. Rio de Janeiro: Campus, 1994.

DAVIS, G**. Management Information Systems**. São Paluo: Mcgraw Hill, 1989.

E-VIEW BOX PROJECT. **Java Medical Imaging Software**. Disponível em <http://eviewbox.sourceforge.net/>. Acessado em: 01 de Maio de 2005.

ENTERPRISE JAVA BEANS. EJB. Disponível em <http://arquivosweb.lncc.br/pdfs/> Acessado em: 30 de junho de 2005.

TEIXEIRA FILHO, J. **Tecnologia da Informação para a Gestão do Conhecimento**. Disponível em <http://www.informal.com.br/artigos>. Acessado em: 22 de outubro de 2005.

GAMMA E.; et al. **Design P: Elements Of Reusable Object-Oriented Software**. United States: Addison-Wesley, 2000.

GARVIN, D. **A. Construindo a organização que aprende**: In: Gestão do conhecimento. Harvard Bussiness Review. Rio de Janeiro: Campus, 2000.

GEARY, D. M.; **JavaServer Pages Avançado: Plataforma Java 2 Enterprise Edition**. Rio de Janeiro: Ciência Moderna, 2002.

GIL, A. C. **Como elaborar projetos de pesquisa social.** São Paulo: Atlas, 1999.

GONÇALVES, J. E. L. A Tecnologia e a realização do Trabalho. **Revista de Administração de Empresas,** São Paulo, jan/fev. 1993.

GRAEML, A. R. **Sistemas de Informação: o alinhamento da estratégia de TI com a estratégia corporativa.** 2ed. São Paulo: Atlas, 2003.

HIRSCHHEIM, R; KLEIN, H. K.; LYYTINEM, K. **Information System Development and Data Modeling: Conceptual and Phylosophical Foundations.** Cambridge: Cambridge Unniversity Press, 1995.

Java BluePrints. **Model-View-Controller.** Disponível em <http://java.sun.com/blueprints/patterns/MVC-detailed.html> Acessado em 21 de outubro de 2004.

KAPLAN, R.CS.; NORTON, D. **Estratégia em ação: balanced scorecard**. Rio de Janeiro: Campus, 1997.

KIM, D.CH. **Gestão Sistêmica da Qualidade: Melhorando a qualidade do agir e do pensar**. São Paulo: Futura, 1996.

LAGO, N. P. **Processamento de Áudio em Tempo Real**. Dissertação de Mestrado. São Paulo, 2004.

LAUTERT, F., OLIVEIRA, R. A. **Paradigma de desenvolvimento MVC (2004)**. Disponível em < www.jelb.org.br/~filipe/artigos/ >. Acessado em:30 de junho de 2005.

LUCAS, P. **Computer-based Decision Support in the Management of Pirmary Gastric non-Hodgkin Lymphoma**. 1997. Disponível em: <http://www.cs.ruu.nl/people/lucas>. Acessado em 18 de novembro de 2004.

LUCAS, P. **Prognostic Methods in Medicine**. 1999. Disponível em: <http://www.cs.ruu.nl/people/lucas>. Acessado em 18 de novembro de 2004.

LAUDON, K. C.; LAUDON, J. P. **Sistemas de Informação**. 4 Ed. Rio de Janeiro: LTC, 1999.

KIMURA, M. **A patient information exchange guidiline using MML, HL7, and DICOM.** International Journal of medical Infotmatics. Diponível em: <http://www.icml9.org/channel.php?lang=en&channel=52&content=170>. Accessado em: 22 de outubro de 2005.

MAGALHÃES, K. **Observando Objetos**. Disponível em <http://www.cafeh.com.br/artigos/13/Observando%20Objetos.pdf>. Acessado em: 15 de Junho de 2005.

MARTINELI, R. M. F. **Tecnologia da Informação na construção do conhecimento: Uma abordagem a partir do modelo de Nonaka & Takeuchi**. Dissertação de Mestrado: Florianópolis, 2001.

MATOS, A. V. **UML: Prático e descomplicado**. São Paulo: Editora Érica, 2002.

MANÃS, A. V. **Administração de Sistemas de Informação**. São Paulo: Érica. 1999.

MAUDONNET, R. **Administração Hospitalar**. Cultura Médica: Rio de Janeiro, 1988.

MCDONALD, C. J., BARNETT, G. O. Medical-Record System, In: Shortliffe, E. H., Perreault, L. E. (eds). **Medical Informatics: Computer Application in Health Care.** New York: Addison-Wesley Publishing. 1990.

MVC, **Web-Tier Application Framework Design.** Disponível em <http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/web-tier/web-tier5.html >. Acessado em: 22 de outubro de 2004.

NONAKA, I. & TAKEUCHI, H. **Criação de conhecimento na empresa – como as empresas japonesas geram a dinâmica da inovação**. Rio de Janeiro: Editora Campus, 1997.

O'BRIEN, J. **Sistemas de Informação – as decisões gerenciais na era da internet**. 9 Ed. São Paulo: Saraiva, 2002.

PAIS, A. P. V. **Arquitetura de Controle Hércules: A base para a Geração Automática de Sistema de Informação com Ênfase na Camada de Controle**. Dissertação de Mestrado. Rio de Janeiro. 2004.

RADIOLOGY. Site de Imagens Médicas – Artigos/notícias. **TC de Cortes Multiplos: Como funciona e quais as aplicações**. Disponível em <http://www.radiology.com.br/materias/rad_materias.asp?flag=1&id_materia=183>. Acessado em: 5 de junho de 2005.

REIS, D. R. **Gestão da Inovação Tecnológica**. Barueri, São Paulo: Manole, 2004.

RICARTE, I. L. M. **Applets**. DCA/FEEC/UNICAMP. Disponível em <http://www.dca.fee.unicamp.br/cursos/PooJava/applet/>. Acessado em: 05 de Maio de 2005.

RIESCO, A. M.; TOMAS, R. M.; MIRA, J. M. **A customisable model for assessment of therapies in the solution of therapy decision tasks**. **Artificial Intelligence in Medicine.** Elsevier, 2000.

RODRIGUES, B. **A hora da gestão do conhecimento.** Disponível em: <http://www.informal.com.br/artigos>. Acessado em 22 de outubro de 2005.

SALEH, A. M. **Adoção de Tecnologia: Um estudo sobre o uso de software livre nas empresas**. Dissertação de Mestrado. São Paulo. 2004.

SANTOS, R. R **Um linguagem de definição e recuperação de imagens baseada em conteúdo em base de dados orientado a objeto**. Anais do XI Simpósio Brasileiro de banco de dados. São Carlos: 14 a 16 de outubro de 1996.

SIGECLIM – **Sistema de Gerenciamento de Clínica Médicas**. Disponível em: <https://clinicasabedotti.com.br:8443/sigeclim/do/loginSistema?op=logarForm>. Acessado em: 10 de outubro de 2005.

SILVA, E. L. **Metodologia da Pesquisa e elaboração de dissertação**. 3.ed. ver. atual. Florianópolis: Laboratório de Ensino a Distância da UFSC. 2001

SILVEIRA, S. A. ; CASSIANO, J. **Software Livre e inclusão digital. São Paulo**.: Conrad Editora do Brasil. 2003.

SOUZA, W. B. **Struts Framework – J2EE Brasil (2004).** Disponível em < http://www.jspbrasil.com.br/jsp/artigos/artigo.jsp?idArtigo=0011> Acessado em: 2 de junho de 2005.

SUN DEVELOPER NETWORK. **Code Samples and Apps – Applets**. Disponível em <http://java.sun.com/applets/>. Acessado em 25 de Maio de 2005.

SUN MICROSYSTEMS SECURITY. **Propriedade Privada – Softwares da Sun ajudam as empresas a impedirem brechas de alto risco na segurança interna**. Disponível em <http://br.sun.com/produtos-solucoes/software/security.html>. Acessado em: 07 de Junho de 2005.

TEMPLE, **A. Jsp, Servlets e J2EE (2004)**. Disponível em <www.inf.ufsc.br/~bosco/downloads/>. Acessado em: 2 de junho de 2005.

TERRA, J. C. C. **Gestão do Conhecimento:** o grande desfio empresarial. São Paulo: Negócios Editora, 2001.

UNIFESP – Universidade Federal de São Paulo (Escola Paulista de Medicina). **O que é a telemedicina**. Disponível em: <http://www.unifesp.br/dis/set/oquee.html>. Acessado em: 22 de outubro de 2005.

WAEGEMANN, P. - **The Five Livels of Eletronic Health Records. M. D. Computing**, v13, n3. 1996.

ZANETI JUNIOR, L. A. **Sistemas de Informação baseados na tecnologia WEB:** um estudo sobre seu desenvolvimento**.** Dissertação de Mestrado. São Paulo FEA/USP. 2003. 189p.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo