

**CILAS DE FREITAS**

**UMA ARQUITETURA BASEADA EM PADRÕES ABERTOS PARA  
VISUALIZAÇÃO CIENTÍFICA VIA INTERNET APLICADA À MEDICINA**

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Informática, Curso de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.

Orientador: Prof. Dr. Klaus de Geus

**CURITIBA**

**2002**



Dedico este trabalho à minha família que mesmo à distância sempre me apoiou. Ao meu pai Hilton Cândido de Freitas, a minha mãe Cleuza Mara Carvalho e aos meus irmãos Marco André de Freitas e Paulo Rodrigo de Freitas.

Também dedico esta dissertação ao meu “mestre” Klaus de Geus, cuja orientação, apoio e incentivo foram fundamentais para realização deste trabalho.

## AGRADECIMENTOS

A

Todos que, direta ou indiretamente, contribuíram para a realização deste trabalho. A Deus, por me dar forças e sabedoria para continuar mesmo nos momentos mais difíceis.

Klaus de Geus, por sua orientação, exigência e apoio necessários na medida certa e nos momentos exatos em que eu precisava. Sem dúvida alguma é um excelente orientador e amigo.

Cássio Rúbio e José Henrique Dometerco, pelas valiosas informações que trocamos durante o desenvolvimento deste trabalho. Em especial ao Henrique por ter gentilmente cedido a aplicação de renderização de volumes utilizado neste trabalho.

Professores da UFPR, os quais me ensinaram a pesquisar e persistir sempre para alcançar os objetivos traçados, além de ministrar com excelência as disciplinas deste curso.

Professor Dr. Sérgio Shiguemi Furuie e Professora Dra. Olga R. P. Bellon por aceitarem o convite para participar da banca examinadora e terem contribuído com sugestões para o refinamento deste trabalho.

# SUMÁRIO

<b>1</b>	<b><u>INTRODUÇÃO</u></b> .....	<b>1</b>
1.1	<u>MOTIVAÇÃO</u> .....	2
1.2	<u>OBJETIVOS</u> .....	3
1.3	<u>ESTRUTURA DO TRABALHO</u> .....	4
<b>2</b>	<b><u>VISUALIZAÇÃO CIENTÍFICA EM MEDICINA</u></b> .....	<b>5</b>
2.1	<u>TÉCNICAS DE AQUISIÇÃO DE IMAGENS</u> .....	5
2.1.1	<u>TOMOGRAFIA COMPUTADORIZADA (CT)</u> .....	6
2.1.2	<u>RESSONÂNCIA MAGNÉTICA (MRI)</u> .....	8
2.2	<u>O PADRÃO DICOM</u> .....	9
2.3	<u>RENDERIZAÇÃO DE VOLUMES</u> .....	11
2.4	<u>RENDERIZAÇÃO DIRETA DE VOLUMES</u> .....	12
2.4.1	<u>RAY CASTING</u> .....	13
2.5	<u>RENDERIZAÇÃO POR EXTRAÇÃO DE SUPERFÍCIES</u> .....	17
2.5.1	<u>MARCHING CUBES</u> .....	18
<b>3</b>	<b><u>ESTADO DA ARTE EM VISUALIZAÇÃO CIENTÍFICA VIA INTERNET</u></b> .....	<b>21</b>
3.1	<u>AS ARQUITETURAS CONVENCIONAIS</u> .....	22
3.1.1	<u>ARQUITETURAS COM ÊNFASE NO CLIENTE</u> .....	25
3.1.2	<u>ARQUITETURAS COM ÊNFASE NO SERVIDOR</u> .....	26
3.2	<u>ARQUITETURAS BASEADAS EM AGENTES</u> .....	27
3.3	<u>VISUALIZAÇÃO COLABORATIVA VIA INTERNET</u> .....	29
3.4	<u>INTEROPERABILIDADE DOS SISTEMAS DE VISUALIZAÇÃO</u> .....	30
<b>4</b>	<b><u>ARQUITETURA PARA VISUALIZAÇÃO CIENTÍFICA BASEADA EM PADRÕES ABERTOS</u></b> .....	<b>32</b>
4.1	<u>PADRÕES ABERTOS PARA COMPUTAÇÃO DISTRIBUÍDA</u> .....	32
4.1.1	<u>CORBA</u> .....	34
4.1.2	<u>WEB SERVICES</u> .....	35
4.2	<u>A INFLUÊNCIA DOS PADRÕES ABERTOS NA ARQUITETURA</u> .....	43
4.3	<u>A ARQUITETURA PROPOSTA</u> .....	46
4.3.1	<u>COMPOSIÇÃO DO ARQUIVO XML COM INFORMAÇÕES DE VOLUMES</u> .....	50
4.3.2	<u>PUBLICAÇÃO DO SERVIÇO E O ACESSO REMOTO</u> .....	53
4.4	<u>ABRANGÊNCIA DA ARQUITETURA</u> .....	56
<b>5</b>	<b><u>O DESENVOLVIMENTO DE UM PROTÓTIPO</u></b> .....	<b>57</b>
5.1	<u>PLATAFORMA</u> .....	58
5.2	<u>IMPLEMENTAÇÃO</u> .....	60

<u>5.2.1</u>	<u>SERVIDOR DE APLICAÇÃO INTEGRADORA</u> .....	60
<u>5.2.2</u>	<u>CLIENTE INTERNET</u> .....	62
<u>5.3</u>	<u>PUBLICAÇÃO E LOCALIZAÇÃO DO SERVIÇO</u> .....	64
<b>6</b>	<b><u>CONCLUSÕES</u></b> .....	<b>67</b>
<u>6.1</u>	<u>TRABALHOS FUTUROS</u> .....	68
<b>7</b>	<b><u>REFERÊNCIAS BIBLIOGRÁFICAS</u></b> .....	<b>69</b>
<b>8</b>	<b><u>APÊNDICE</u></b> .....	<b>72</b>

## LISTA DE ILUSTRAÇÕES

<a href="#">FIGURA 2.1: TOMÓGRAFO CT TWIN ELSCINT. EXTRAÍDO DE: (DAPI, 2002).</a>	6
<a href="#">FIGURA 2.2: UMA FATIA DA CABEÇA DE UM PACIENTE, OBTIDA POR CT.</a>	7
<a href="#">FIGURA 2.3: EQUIPAMENTO DE RESSONÂNCIA MAGNÉTICA. EXTRAÍDO DE: (DAPI, 2002).</a>	8
<a href="#">FIGURA 2.4: UMA FATIA DA CABEÇA DE UM PACIENTE, OBTIDA POR MRI.</a>	9
<a href="#">FIGURA 2.5: MODELO DE INFORMAÇÕES DICOM (<i>DICOM COMPOSITE INSTANCE IOD INFORMATION MODEL</i>) (DICOM, 2001).</a>	11
<a href="#">FIGURA 2.6: ESQUEMA SIMPLIFICADO DO ALGORITMO DE <i>RAY CASTING</i> (WATT, 1993).</a>	14
<a href="#">FIGURA 2.7: FUNÇÃO DE CLASSIFICAÇÃO DE VALORES CT. EXTRAÍDO DE: (GEUS, 1993).</a>	15
<a href="#">FIGURA 2.8: POSSÍVEIS CASOS DO ALGORITMO DE <i>MARCHING CUBES</i> (LORENSEN, 1987).</a>	20
<a href="#">FIGURA 3.1: PIPELINE DE VISUALIZAÇÃO.</a>	22
<a href="#">FIGURA 3.2: CLASSIFICAÇÃO DAS ARQUITETURAS CLIENTE/SERVIDOR CONVENCIONAIS.</a>	24
<a href="#">FIGURA 3.3: ARQUITETURA COM ÊNFASE NO CLIENTE.</a>	25
<a href="#">FIGURA 3.4: ARQUITETURA COM ÊNFASE NO SERVIDOR GERANDO IMAGEM PARA O CLIENTE.</a>	26
<a href="#">FIGURA 3.5: ARQUITETURA COM ÊNFASE NO SERVIDOR GERANDO ARQUIVO VRML PARA O CLIENTE.</a>	27
<a href="#">FIGURA 3.6: ARQUITETURA BASEADA EM AGENTES. OS AGENTES SÃO DISTRIBUÍDOS ENTRE OS CLIENTES E SERVIDORES DISPONÍVEIS EM TEMPO DE EXECUÇÃO.</a>	28
<a href="#">FIGURA 3.7: PIPELINE DE VISUALIZAÇÃO EXPANDIDO PARA SUPORTAR CSCV (WOOD, 1995).</a>	29
<a href="#">FIGURA 4.1: COMUNICAÇÃO CLIENTE/SERVIDOR POR MEIO DO ORB. EXTRAÍDO DE: (ORFALI, 1997).</a>	35
<a href="#">FIGURA 4.2: SOA – <i>SERVICE-ORIENTED ARCHITECTURE</i> (GRAHAM, 2001).</a>	37
<a href="#">FIGURA 4.3: EXEMPLO DE UM DOCUMENTO NO FORMATO XML.</a>	38
<a href="#">FIGURA 4.4: CHAMADA DE UM MÉTODO USANDO O PROTOCOLO SOAP.</a>	40
<a href="#">FIGURA 4.5: RESPOSTA DE UM MÉTODO USANDO O PROTOCOLO SOAP.</a>	40
<a href="#">FIGURA 4.6: HIERARQUIA DOS ELEMENTOS XML EM UDDI. EXTRAÍDO DE: (UDDI, 2000).</a>	42
<a href="#">FIGURA 4.7: DIAGRAMA DOS COMPONENTES DA ARQUITETURA.</a>	46
<a href="#">FIGURA 4.8: HIERARQUIA DAS INFORMAÇÕES DICOM ESTRUTURADAS EM DIRETÓRIOS.</a>	50
<a href="#">FIGURA 4.9: EXEMPLO DAS INFORMAÇÕES DE PRÉ-CALIBRAGEM DEFINIDAS PARA UM VOLUME.</a>	51
<a href="#">FIGURA 4.10: EXEMPLO DO ARQUIVO LISTAVOLUMES.XML</a>	52
<a href="#">FIGURA 4.11: EXEMPLO DA LISTA DE VOLUMES SIMPLIFICADA, NO FORMATO XML.</a>	54
<a href="#">FIGURA 5.1: CENÁRIO DE ATUAÇÃO DO PROTÓTIPO.</a>	57

<a href="#"><u>FIGURA 5.2: DIAGRAMA DE CLASSES DO PROTÓTIPO.</u></a> .....	61
<a href="#"><u>FIGURA 5.3: ACESSO DO <i>THIN CLIENT</i> AO SERVIÇO DE RENDERIZAÇÃO POR MEIO DE UM <i>WEB BROWSER</i>.</u></a> .....	63
<a href="#"><u>FIGURA 5.4: ACESSO DO <i>THIN CLIENT</i> AO SERVIÇO DE RENDERIZAÇÃO POR MEIO DE UM <i>WEB BROWSER</i> COM ROTAÇÃO DE 45 GRAUS NO EIXO Y DO VOLUME.</u></a> .....	64
<a href="#"><u>FIGURA 5.5: ACESSO AO SERVIÇO UDDI DA MICROSOFT.DISPONÍVEL EM &lt;HTTP://UDDI.MICROSOFT.COM&gt; ACESSO EM 10/07/2002.</u></a> .....	65

## RESUMO

A visualização científica aplicada à medicina tem sido muito explorada nos últimos anos. O uso da Internet como meio de comunicação impulsionou o desenvolvimento de aplicações distribuídas de visualização de volumes para imagens médicas. A maioria dessas aplicações é baseada em arquiteturas fortemente acopladas, as quais dificultam sua acessibilidade e integração com outras aplicações. Este trabalho apresenta uma arquitetura para desenvolvimento de serviços de renderização de volumes de imagens médicas, utilizando padrões abertos para computação distribuída via Internet. Esta arquitetura permite uma maior acessibilidade e integração entre as organizações provedoras de serviços de renderização volumétrica e seus clientes.

**PALAVRAS-CHAVE:** Visualização Científica, Imagens Médicas, Internet, Serviços Web.

## ABSTRACT

Scientific visualization applied to medicine has been strongly explored in the last years. The use of the Internet as a means of communicating impelled the development of distributed volume visualization applications for medical images. Most of these applications are based on strongly coupled architectures, which implies in difficulties on issues such as accessibility and integration with other applications. This work presents an architecture for the development of volume rendering services for medical images, using open patterns for distributed computing via Internet. This architecture allows for larger accessibility and integration between the volumetric rendering services providers and their customers.

**KEYWORDS:** Scientific Visualization, Medical Images, Internet, Web Services.

## 1 INTRODUÇÃO

A visualização científica, disciplina da área de computação gráfica, tem tido um papel muito importante dentro do campo da ciência, tendo como objetivo prover recursos visuais que forneçam ao cientista uma maior compreensão do fenômeno estudado.

De acordo com MCCORMICK (1987), “Visualização é uma ferramenta para a interpretação de dados representados em computador e para a geração de imagens a partir de conjuntos de dados complexos e multidimensionais. Caracteriza-se visualização científica quando estes conjuntos de dados representam fenômenos complexos e o objetivo é a extração de informações científicas relevantes”.

A ampla utilização das técnicas de visualização científica, aliada à necessidade de acessibilidade dessas informações por meio de navegadores Web – *World Wide Web* na Internet, contribuiu para criação dos *Sistemas de Visualização Científica via Internet*.

As arquiteturas convencionais destes sistemas normalmente têm seus módulos distribuídos na Internet em clientes e servidores. Alguns trabalhos procuram explorar mais fortemente os recursos do servidor, enquanto em outros é dada mais ênfase no lado do cliente (BRODLIE, 2000).

Trabalhos mais recentes têm utilizado tecnologias, tais como agentes móveis, para explorar os recursos disponíveis na rede aproveitando o poder de processamento dos equipamentos heterogêneos distribuídos entre o cliente e o servidor (TSOI, 2002). Além disso, muitas pesquisas têm explorado funcionalidades para visualização colaborativa, a qual permite que vários usuários remotos participem de uma mesma sessão de visualização, permitindo que eles atuem como colaboradores no processo de visualização (MANSSOUR, 2000).

## 1.1 MOTIVAÇÃO

Muitas arquiteturas foram criadas para trabalhar com renderização de volumes utilizando a Internet como meio de comunicação, porém a maioria delas tem implementações que não obedecem a um padrão definido para desenvolvimento de sistemas distribuídos via Internet.

Como os sistemas baseados nestas arquiteturas utilizam tecnologias proprietárias, eles têm dificuldades para comunicação com aplicações em ambientes heterogêneos, ou seja, são limitados a funcionar em ambientes específicos. Esta situação é observada nos casos em que o usuário tenta acessar um sistema de renderização de volumes via Internet e recebe uma mensagem advertindo-o de que seu ambiente não é compatível com o sistema.

Sistemas de renderização de volumes via Internet devem ser capazes de se comunicar uns com os outros de forma simples em ambientes heterogêneos, proporcionando acesso ao maior número de usuários possível. Trabalhos em Telemedicina e Teleradiologia (JOHN, 2000) (MATHIESEN, 2001) relatam a importância da integração dos sistemas para a área médica.

Ainda na área médica são notados esforços para padronização das imagens no formato DICOM (*Digital Imaging and Communications in Medicine*) (DICOM, 2001), cujo detalhamento é apresentado no capítulo 2, possibilita o intercâmbio destas imagens entre diferentes equipamentos e ambientes.

Tendo em vista as limitações das arquiteturas convencionais e a necessidade pela integração dos sistemas que utilizam tecnologias proprietárias, torna-se viável a criação de uma arquitetura baseada em padrões abertos para desenvolvimento destes sistemas.

## 1.2 OBJETIVOS

O objetivo principal deste trabalho é estabelecer os fundamentos e as diretrizes básicas para a criação de uma Arquitetura que utilize padrões abertos para criação de sistemas de renderização volumétrica via Internet aplicados à área médica. Além disso, objetiva-se demonstrar os benefícios de integração e de acessibilidade que podem ser alcançados por meio desta arquitetura.

Para permitir acessibilidade adequada, é proposto o uso de mecanismos de publicação, busca e acesso destes sistemas, mesmo em ambientes heterogêneos. Isso possibilita o acesso de usuários independentemente de sua plataforma de trabalho.

O objetivo da integração em ambientes heterogêneos é permitir que os sistemas possam interagir entre si, como se estivessem trabalhando em um ambiente único. Essa característica é alcançada por meio da utilização de padrões abertos na forma de comunicação entre estes sistemas.

Este trabalho também apresenta a aplicação desta arquitetura na construção de um protótipo, para demonstrar a viabilidade do desenvolvimento destes sistemas.

### 1.3 ESTRUTURA DO TRABALHO

Os capítulos subseqüentes deste trabalho estão organizados da seguinte forma: O capítulo 2 apresenta uma introdução sobre visualização científica aplicada à área médica. O capítulo 3 apresenta os trabalhos relacionados a arquiteturas de visualização científica via Internet. O capítulo 4 faz uma breve revisão de padrões abertos para sistemas distribuídos e apresenta a arquitetura proposta. O capítulo 5 apresenta detalhes da implementação do protótipo baseado nesta arquitetura. O capítulo 6 apresenta as conclusões e trabalhos futuros.

## 2 VISUALIZAÇÃO CIENTÍFICA EM MEDICINA

Uma das grandes áreas de atuação da visualização científica é a medicina. A evolução dos equipamentos e das técnicas de processamento de imagens e computação gráfica passou a permitir diagnósticos precisos por meio de imagens com alta fidelidade. Isso contribuiu para tornar a visualização científica uma ferramenta essencial na área médica.

Serão apresentados, neste capítulo, técnicas e padrões de imageamento médico, assim como técnicas de visualização aplicadas à medicina.

### 2.1 TÉCNICAS DE AQUISIÇÃO DE IMAGENS

Muitos exames têm se beneficiado das técnicas de aquisição e tratamento de imagens para visualizar tecidos internos do corpo do paciente, sem a necessidade de procedimento cirúrgico.

Existem várias modalidades de aquisição de imagens onde a visualização volumétrica pode atuar. Na área de medicina, as modalidades de imageamento mais utilizadas são as seguintes:

- Tomografia Computadorizada (CT);
- Ressonância Magnética (MRI);
- Ultra-som;
- Microscopia confocal.

Dentre elas, a tomografia computadorizada e a ressonância magnética são as modalidades utilizadas nas aplicações médicas que mais têm recebido os benefícios da renderização tridimensional, e são descritas brevemente a seguir.

### 2.1.1 TOMOGRAFIA COMPUTADORIZADA (CT)

A Tomografia Computadorizada (CT) é uma técnica de aquisição de imagens médicas, na qual utiliza-se a tecnologia de computação associada a emissão de raios X para produção de uma imagem que representa uma fatia do corpo do paciente.

Com a Tomografia Computadorizada (CT) uma estrutura interna de um objeto pode ser reconstruída a partir de múltiplas projeções deste objeto. Durante a aquisição de cada fatia, um tubo de raios X circundando o paciente produz um feixe de raios X que são emitidos na direção do corpo do paciente, sendo que parte destes raios são absorvidos por um anel de detectores localizados na extremidade oposta a este tubo (OLIVEIRA JR., 1999). A Figura 2.1 apresenta um tomógrafo utilizado no DAPI<sup>1</sup> – Diagnóstico Avançado Por Imagem (DAPI, 2002).



Figura 2.1: Tomógrafo CT Twin Elscint. Extraído de: (DAPI, 2002).

---

<sup>1</sup> DAPI – Diagnóstico Avançado Por Imagem é um instituto de radiologia.

A intensidade dos raios X que chegam aos detectores é dependente das características de absorção dos tecidos pelos quais eles passam. Usando métodos matemáticos, o computador utiliza as informações obtidas nos detectores para reconstruir os coeficientes de absorção e produzir a imagem de uma fatia (OLIVEIRA JR., 1999).

O resultado de um exame por CT é uma série de imagens representando fatias transversais do paciente. Cada fatia representa uma faixa do corpo do paciente com uma espessura normalmente entre 1 e 10 milímetros. A maioria dos *scanners* CT tradicionais geram imagens de 512x512 *pixels* para representar cada fatia (PAIVA, 1999). A Figura 2.2 apresenta uma fatia transversal da cabeça de um paciente, obtida por CT.

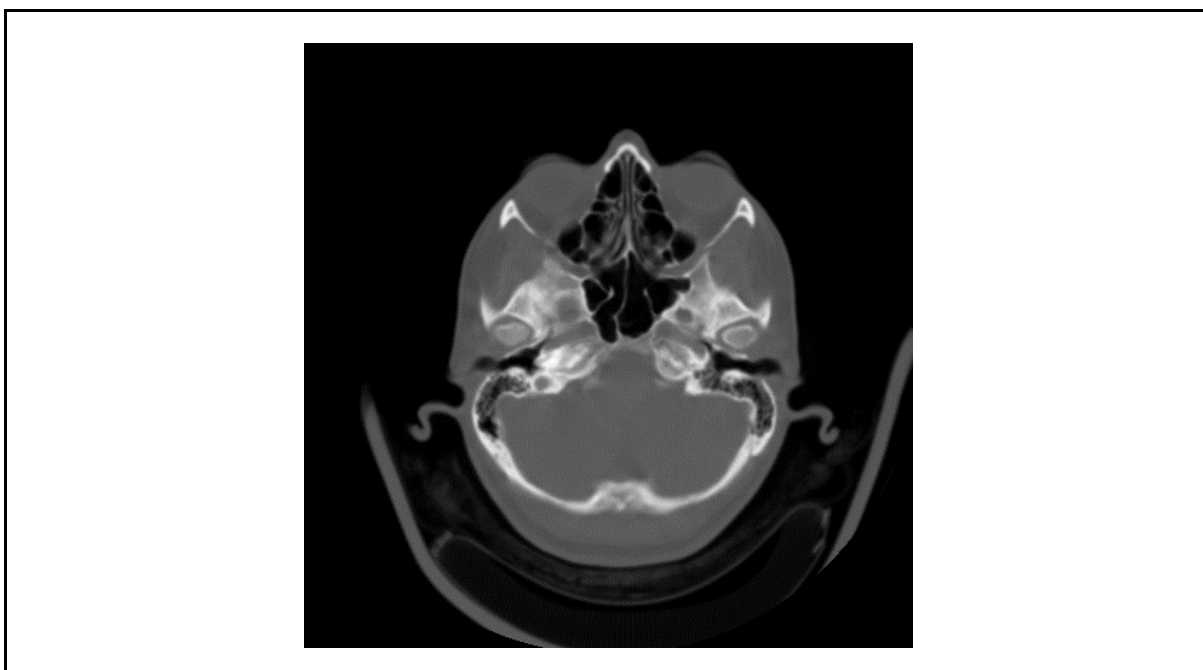


Figura 2.2: Uma fatia da cabeça de um paciente, obtida por CT.

### 2.1.2 RESSONÂNCIA MAGNÉTICA (MRI)

A ressonância magnética (MRI) possibilita a visualização seletiva das características de tecidos diferentes. O scanner de MRI possui grandes rolos magnéticos que produzem campos magnéticos com magnitudes de vários tesla. A Figura 2.3 apresenta um equipamento *scanner* de MRI utilizado no DAPI (DAPI, 2002).



Figura 2.3: Equipamento de Ressonância Magnética. Extraído de: (DAPI, 2002).

O paciente é colocado em um forte campo magnético. Os núcleos dos átomos de hidrogênio nos tecidos se alinham com este forte campo magnético. Durante este processo de alinhamento, o núcleo tende a variar no campo magnético. Esta oscilação de ressonância é conhecida como fenômeno de ressonância magnética.

Uma imagem de MRI é um mapa de intensidades de radio freqüência emitidas pelos tecidos. Quanto mais brilhante for uma área, maior é a intensidade do sinal naquele ponto. As áreas escuras nas imagens indicam onde os sinais não são

produzidos (LICHTENBELT, 1998). A Figura 2.4 apresenta uma fatia da cabeça de um paciente, obtida por MRI.

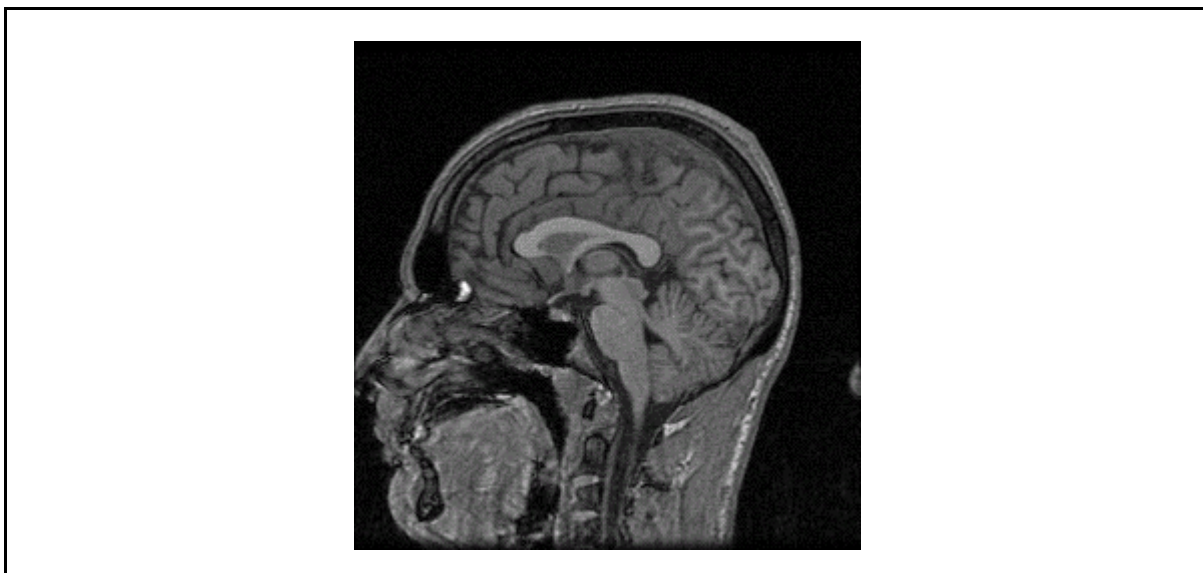


Figura 2.4: Uma fatia da cabeça de um paciente, obtida por MRI.

Com o controle do espaço e amplitude dos gradientes, os técnicos podem programar perfis únicos para isolar planos de dados arbitrários. Características de tecidos específicos podem ser isoladas, permitindo que técnicos e físicos identifiquem um tumor ou inflamação no tecido. Uma das vantagens dos scanners de MRI sobre os de CT é a ausência da radiação ionizante (LICHTENBELT, 1998).

## 2.2 O PADRÃO DICOM

Em 1983, o ACR – *American College of Radiology* e o NEMA – *National Electrical Manufacturers* formaram um comitê chamado ACR-NEMA, com o objetivo de desenvolver um padrão para imagens digitais e transmissão destas imagens entre os equipamentos em medicina. Este padrão recebeu o nome de DICOM – *Digital Imaging and Communication in Medicine* (DICOM, 2001).

O DICOM 3.0, além de encapsular as imagens digitais, também possibilita o armazenamento de informações, tais como dados do paciente, dados dos processos e configurações utilizadas na aquisição das imagens. Um dos maiores benefícios deste padrão é a independência de plataforma, a qual contribuiu para o desenvolvimento dos PACS<sup>2</sup> - *Picture Archiving and Communication Systems*.

Com a utilização da orientação a objetos para modelagem e implementação de seus módulos, o padrão DICOM possibilita uma evolução constante no seu desenvolvimento. Ele especifica a definição de objeto de informação (*IOD – Information Object Definition*) para modelar processos do mundo real, proporcionando a representação dos ambientes médicos. A Figura 2.5 ilustra esta modelagem, utilizando o diagrama de entidade-relacionamento.

A entidade Paciente (*Patient*) contém dados que descrevem o paciente, tais como nome, idade. A entidade Estudo (*Study*) possui as informações de todos os estudos feitos sobre este paciente. A entidade Série (*Series*) representa um grupo lógico de imagens e é composta por duas entidades: A Referência de Imagem (*Frame of Reference*) que possui informações sobre o relacionamento espacial das imagens com as séries, e a entidade Equipamento (*Equipment*) que armazena informações, tais como fabricante e revisão do software. E finalmente as entidades de imagem (*Image*) que armazenam os dados do tipo pixel de uma imagem.

---

<sup>2</sup> PACS são Sistemas para Armazenamento e Comunicação de Imagens Médicas. Eles possibilitam a visualização de imagens para diagnósticos, laudos e consultas remotamente.

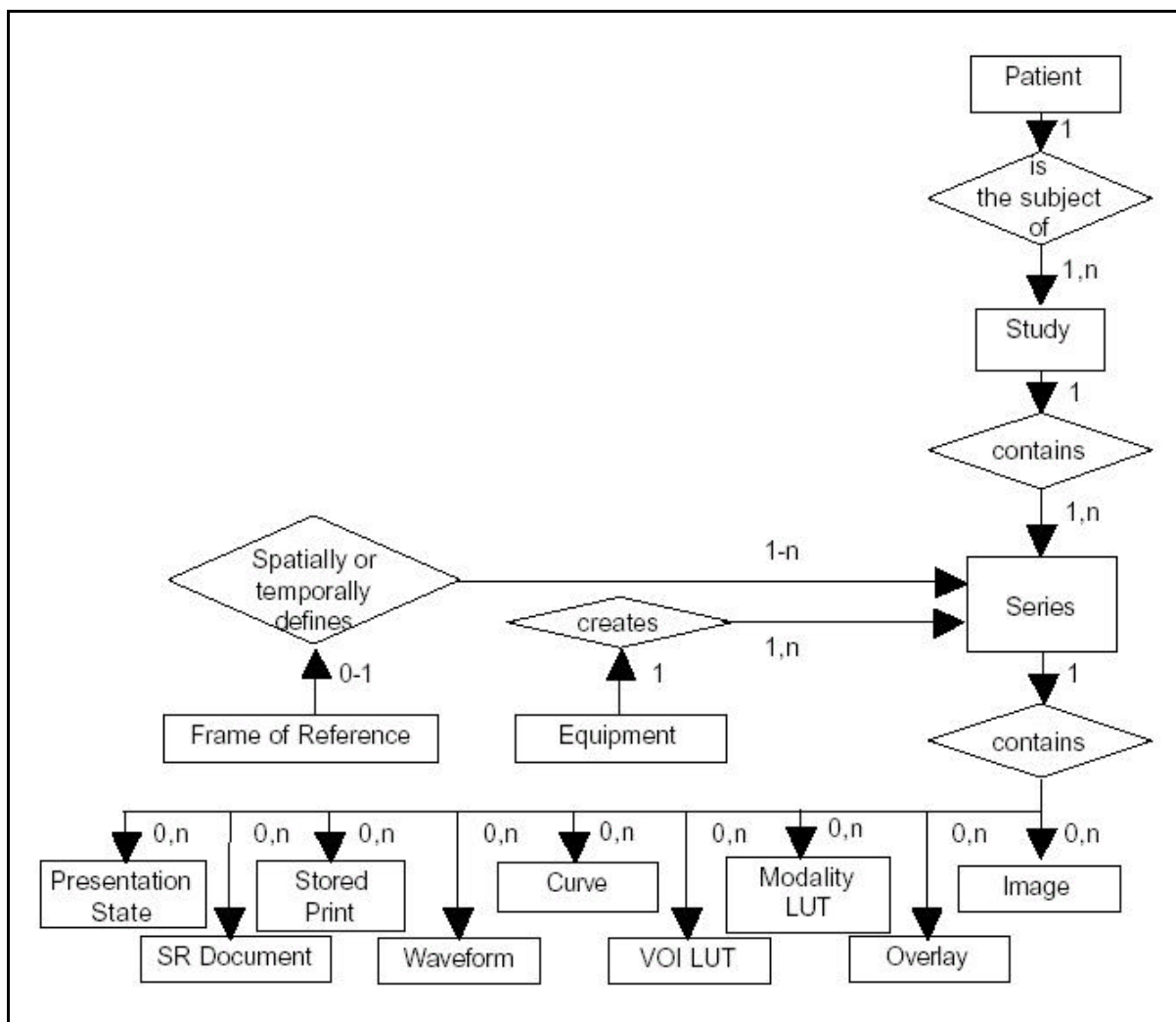


Figura 2.5: Modelo de Informações DICOM (*DICOM composite instance IOD Information Model*) (DICOM, 2001).

### 2.3 RENDERIZAÇÃO DE VOLUMES

Em áreas como a medicina, as aplicações envolvendo imagens têm caráter tridimensional, exigindo técnicas que permitam que estruturas internas ao volume de dados sejam visualizadas. Estas imagens tridimensionais (volumes) são criadas a partir das imagens bidimensionais (fatias) que foram geradas pelos equipamentos de tomografia computadorizada (CT), ressonância magnética (MRI), e outros usados para aquisição de imagens médicas.

Renderização de Volumes (*volume rendering*) normalmente é definido como um processo que opera diretamente no conjunto de dados para produzir uma imagem sem a geração de representação geométrica intermediária.

Com os avanços da computação gráfica, conseguiu-se usar primitivas geométricas para produzir imagens com a mesma qualidade daquelas geradas pelas técnicas de renderização direta de volumes. Devido a estes avanços, a Renderização de Volumes pode ser definida, de forma mais abrangente, como: Um método que opera sobre dados volumétricos para produzir uma imagem (SCHROEDER, 1997).

As técnicas mais conhecidas de renderização de volumes são a extração de superfícies, a qual usa primitivas geométricas para compor o volume, e a renderização direta de volumes (*direct volume rendering*). Essas duas técnicas serão descritas em mais detalhes nas próximas seções.

## 2.4 RENDERIZAÇÃO DIRETA DE VOLUMES

A renderização direta de volumes é uma técnica que processa o volume como um todo, sem a necessidade de processos intermediários. Esta técnica, considera todos os pontos no interior do volume necessários para compor cada *pixel* da imagem resultante.

Por meio do processo de classificação, é possível alterar propriedades de porções de dados do objeto tridimensional para que estas fiquem transparentes. Isto permite que regiões de interesse, sejam vistas mais facilmente (LICHTENBELT, 1998). Uma das técnicas mais utilizadas, especialmente em aplicações da área de medicina, é a técnica chamada *Ray Casting* (LEVOY, 1988).

### 2.4.1 RAY CASTING

O processo de renderização volumétrica por *ray casting* é muito similar à produção de uma chapa de raio X. Raios imaginários são passados através de um corpo tridimensional. Os raios viajam através dos dados acumulando o valor (intensidade) de cada dado por onde passam.

Ao deixar os dados, os raios compreendem a extensão de valores acumulados. No caso da radiografia, cada valor acumulado de absorção de raios X é projetado numa chapa. No caso de renderização de volumes, cada amostra computada dentro do volume contribui para a geração de uma imagem tridimensional (LICHTENBELT, 1998).

No algoritmo de *ray casting*, para cada pixel da imagem final, são lançados raios na direção do volume. Se o raio interceptar o volume, o conteúdo do volume ao longo do raio é amostrado, transformando-se o valor em cor e opacidade e resultando num valor que é atribuído ao *pixel* (PAIVA, 1999). A figura 2.6 ilustra um esquema simplificado do algoritmo de *ray casting*.

O algoritmo de *ray casting* produz imagens de boa qualidade para a área médica porque pode utilizar sem grandes perdas um modelo de iluminação mais elaborado, como o de PHONG (1975) (OLIVEIRA JR., 1999).

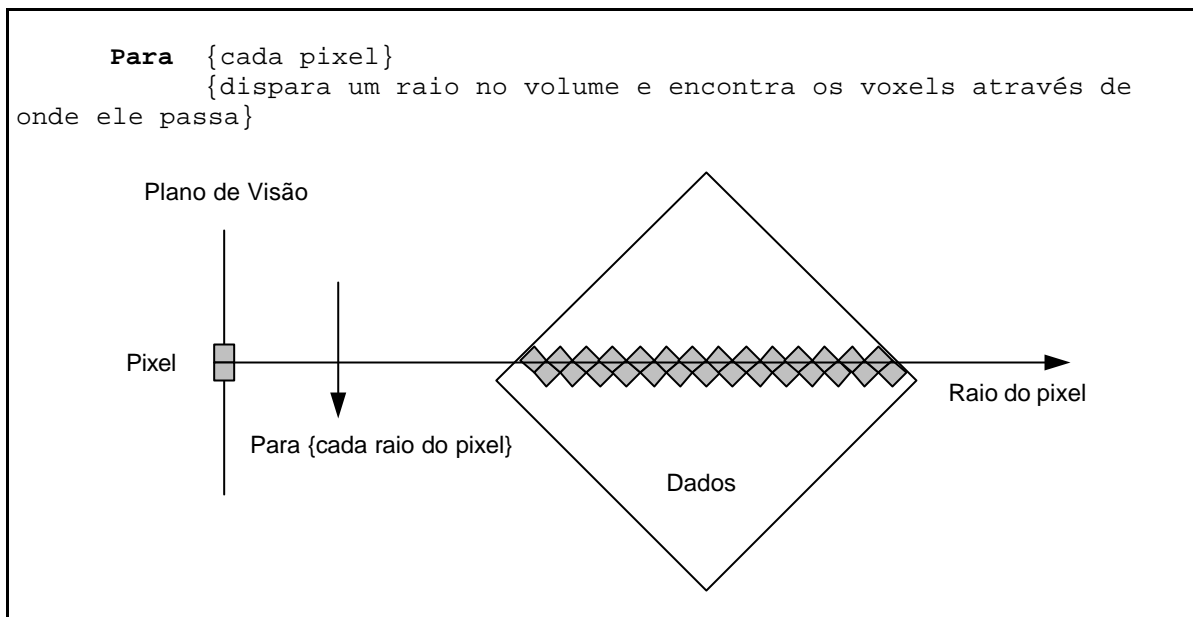


Figura 2.6: Esquema simplificado do algoritmo de *ray casting* (WATT, 1993).

O processo de renderização de volumes pode ser dividido em 4 passos (DREBIN, 1988) (GEUS, 1993): Classificação, detecção de superfícies, modelo de iluminação e projeção.

#### 2.4.1.1 CLASSIFICAÇÃO

O processo de classificação determina percentuais de diferentes materiais que podem estar presentes em cada *voxel*. No caso da tomografia computadorizada, a classificação utiliza quatro tipos de materiais, de acordo com seus valores de absorção de raio-X: Ar, gordura, tecido e osso. Portanto, cada *voxel* pode ser classificado como representando uma das seguintes configurações:

- Ar
- Mistura entre ar e gordura
- Gordura
- Mistura entre gordura e tecido

- Tecido
- Mistura entre tecido e osso
- Osso

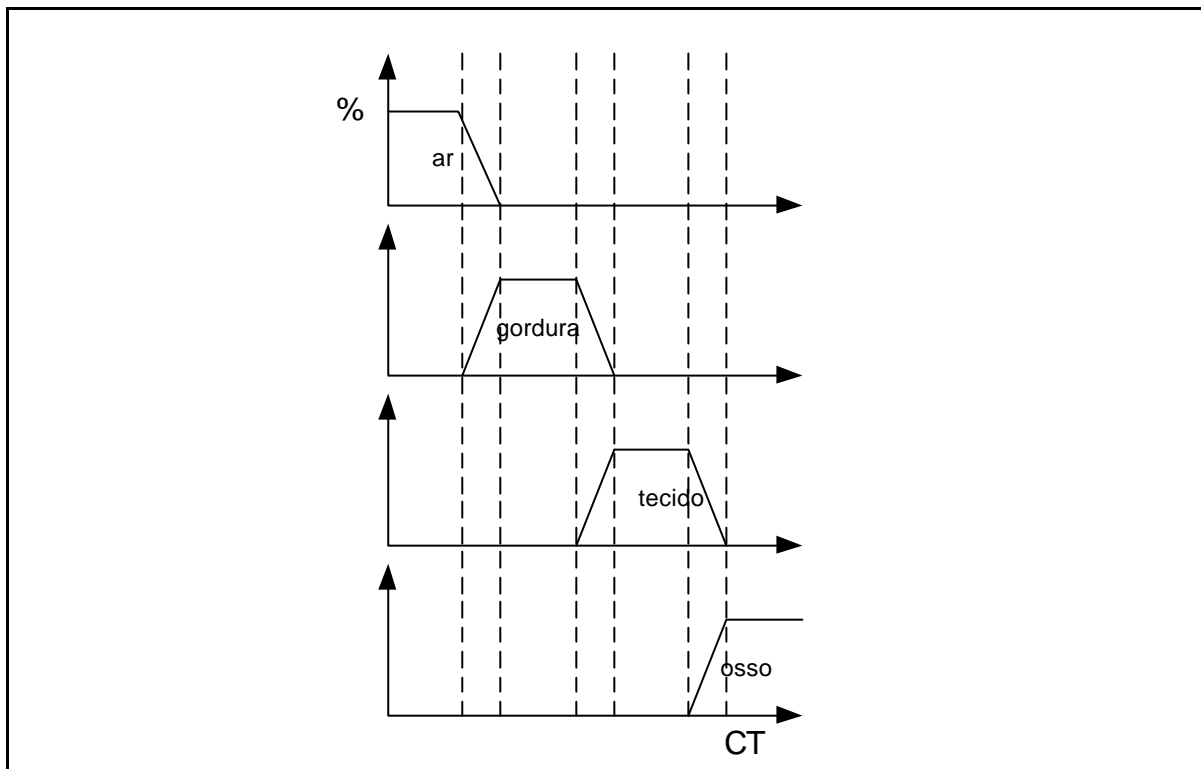


Figura 2.7: Função de classificação de valores CT. Extraído de: (GEUS, 1993).

O processo de classificação é de natureza probabilística, e estima a probabilidade de um material estar presente homogeneamente num *voxel*, ou seja, estima a quantidade daquele material no *voxel*. A Figura 2.7 apresenta o diagrama da função usada no processo de classificação (DREBIN, 1988) (GEUS, 1993).

### 2.4.1.2 DETECÇÃO DE SUPERFÍCIES

O processo de detecção de superfícies é baseado em um valor de densidade atribuído a cada material. Quando dois materiais de diferentes densidades se encontram, uma superfície ocorre. Se o mesmo valor de densidade for atribuído a dois materiais distintos, não aparecerá superfície quando os dois materiais se encontrarem. Isso dá bastante flexibilidade ao processo de geração de imagem, uma vez que se pode controlar que materiais devem ser visíveis por meio da manipulação dos parâmetros de densidade (DREBIN, 1988) (GEUS, 1993).

O vetor normal é calculado a partir do gradiente de densidade entre *voxels* adjacentes:

$$N_x = D(x + 1, y, z) - D(x - 1, y, z)$$

$$N_y = D(x, y + 1, z) - D(x, y - 1, z)$$

$$N_z = D(x, y, z + 1) - D(x, y, z - 1)$$

O vetor então é normalizado, resultando num vetor de comprimento unitário, para que possa ser usado no processo de sombreamento.

### 2.4.1.3 MODELO DE ILUMINAÇÃO

No processo de iluminação, o sombreamento é calculado quando cada amostra é processada, levando em consideração a luz refletida na direção do olho a partir das fontes de luz.

Um raio parte de um *pixel* da tela e viaja através do volume, entrando num *voxel* com intensidade  $I$  e saindo do *voxel* com intensidade  $I'$ . O *voxel* então contribui para o valor do *pixel* correspondente ao raio, de acordo com os valores de opacidade e cor de ambos (*voxel* e raio) (GEUS, 1993).

A cor refletida da superfície é função dos seguintes parâmetros:

- Vetor normal à superfície;
- Quantidade de superfície encontrada (variação de densidade de um material para o outro);
- Cor difusa da superfície;
- Direção e cor da fonte de luz;
- Posição do observador.

O modelo de iluminação normalmente usado é o de PHONG (1975).

#### 2.4.1.4 PROJEÇÃO

Uma vez que um *pixel* no vídeo pode representar centenas de valores que foram amostrados ao longo do raio, é necessário acumular estes valores em um único valor. A acumulação é obtida por meio da função de *compositing*. Existem duas maneiras básicas: *front-to-back* e *back-to-front*. A maneira usada depende da direção em que o raio atravessa o volume (LICHTENBELT, 1998).

#### 2.5 RENDERIZAÇÃO POR EXTRAÇÃO DE SUPERFÍCIES

A técnica de extração de superfícies pré-processa o volume de dados para identificar, ou extrair, superfícies que posteriormente são renderizadas por meio de técnicas tradicionais de computação gráfica. A técnica mais conhecida nesta categoria é a técnica *Marching Cubes* (LORENSEN, 1987).

### 2.5.1 MARCHING CUBES

Este algoritmo extrai as superfícies dos volumes de dados por meio de uma técnica que localiza a superfície em um cubo lógico criado com oito *voxels*. Neste algoritmo, uma decisão binária (*thresholding*<sup>3</sup>) deve ser feita para cada *voxel*, para observar se uma superfície atravessa este cubo específico ou não. Um polígono é criado para representar a porção de superfície presente dentro do cubo. As superfícies são então definidas pela conexão destes polígonos, e finalmente são aplicadas técnicas de renderização nestas superfícies.

Cada cubo é processado separadamente. Para determinar como a superfície atravessa o cubo, o valor binário 1 é associado ao vértice do cubo se o valor deste vértice exceder ou igualar o valor da superfície que está sendo construída, com o valor de densidade especificado pelo usuário. Nestes casos, quando o valor do dado excede ou iguala o valor da superfície, assume-se que o vértice está dentro ou sobre a superfície respectivamente. Quando o valor do dado é menor que o valor da superfície, o valor binário 0 (zero) é associado ao vértice, e assume-se que este está fora da superfície.

A superfície corta as arestas do cubo que têm um vértice dentro ou na superfície (um) e o outro fora (zero). Desta forma, é determinada a topologia da superfície dentro do cubo, e ela pode ser classificada em 256 casos diferentes. Contudo, levando em consideração os casos complementares e as propriedades simétricas, este número pode ser reduzido para 14, conforme ilustrado na Figura 2.8.

Um índice de um byte para cada caso é criado, um bit para cada vértice, baseado no estado do vértice, o qual é usado como ponteiro na tabela de arestas pré-calculadas que possui todos os cortes de arestas para uma configuração de

---

<sup>3</sup> Thresholding é uma técnica para seleção de dados que estão dentro de uma faixa de valores. Normalmente um thresholding escalar seleciona dados cujos valores escalares atendem a um critério escalar.

cubo. A interpolação linear é usada para determinar a localização onde a superfície cruza cada aresta. Desta forma, polígonos, normalmente triângulos, são construídos por meio de cortes.

Para aplicar as funções de renderização nas superfícies, deve-se encontrar o vetor normal em cada vértice dos polígonos. Eles são determinados pelos valores de densidade. Uma superfície de densidade constante tem um componente de gradiente zero ao longo da direção tangencial da superfície. Assim, a direção do vetor gradiente é normal á superfície. O vetor gradiente na superfície é calculado pela interpolação linear do vetor gradiente sobre os vértices do cubo.

Este algoritmo vem sendo usado em aplicações envolvendo imagens médicas, tais como CT, para extrair, por exemplo, ossos, tais como crânio ou outra parte do esqueleto que possa ser representada por uma malha de polígonos (GEUS, 1992).

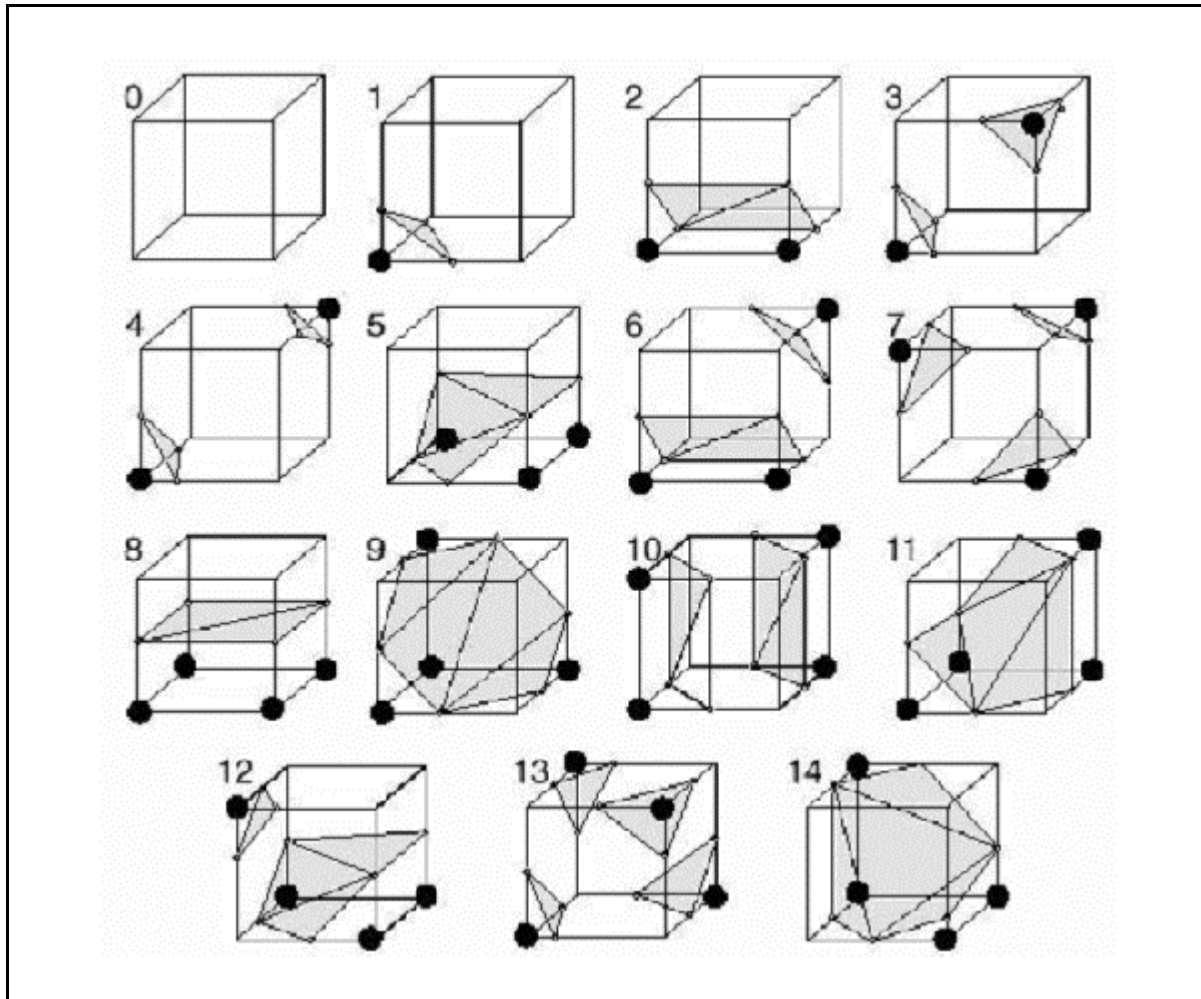


Figura 2.8: Possíveis casos do algoritmo de *Marching Cubes* (LORENSEN, 1987).

### 3 ESTADO DA ARTE EM VISUALIZAÇÃO CIENTÍFICA VIA INTERNET

Devido à possibilidade do compartilhamento de informações entre um grande número de usuários em ambientes heterogêneos, a Internet é vista como uma oportunidade para desenvolvimento de projetos na área de Visualização Científica (PURGATHOFER, 1997).

A *World Wide Web*, ou simplesmente *Web*, começou como meio de publicação, com *links* de hipertexto<sup>4</sup> usados para conectar documentos em várias localidades na Internet. Ela era usada pelos cientistas para publicação de resultados de suas pesquisas. Estes documentos de hipertexto normalmente incluíam imagens 2D (em arquivos GIF<sup>5</sup> e JPG<sup>6</sup>), capturadas como saída dos sistemas de visualização. O próximo passo foi a publicação de representações de modelos 3D, a qual permitia aos usuários a oportunidade de visualizar estes modelos em diferentes ângulos. Isso foi possível devido à padronização do *VRML (Virtual Reality Modelling Language)*<sup>7</sup> (BRODLIE, 1997).

Os Sistemas de Visualização Volumétrica passaram a ter módulos para geração de resultados em arquivos no formato VRML. Tais sistemas liberavam estes tipos de arquivos para consulta via *Web*. Desta forma, os cientistas podiam selecionar um volume disponível e fazer o seu *download* para executar a renderização localmente.

---

<sup>4</sup> Hipertexto é uma rede de ligações entre documentos, onde uma palavra ou uma imagem em um documento pode ser uma referência para direcionar o foco atual para outro documento.

<sup>5</sup> GIF - *Graphics Interchange Format*, é um formato que permite armazenar imagens com um máximo de 256 cores, definidas por meio de mapas de cor.

<sup>6</sup> JPG é a extensão convencional para o formato JFIF (*JPEG File Interchange Format*), permite armazenar imagens com  $2^{24}$  cores.

<sup>7</sup> VRML, linguagem para modelagem de Realidade Virtual é o Padrão Internacional (ISO/IEC 14772) de formato de arquivo para descrição de objetos 3D interativos na Internet.

Porém a Web evoluiu de um simples meio de comunicação para um grande ambiente para computação distribuída. Esta evolução contribuiu para que os Sistemas de Visualização Volumétrica passassem a atuar como ferramentas de análise, onde os cientistas podiam, por meio de um *Web browser*, especificar quais conjuntos de dados e quais ferramentas de renderização seriam utilizados no processo de visualização.

### 3.1 AS ARQUITETURAS CONVENCIONAIS

Arquitetura de Sistemas é um conjunto de definições e decisões que são tomadas durante a fase de projeto, que irão influenciar todo o ciclo de desenvolvimento e execução dos sistemas. Estas definições afetam diretamente a qualidade dos sistemas, tais como: Modificabilidade, Reaproveitamento, Portabilidade, Funcionalidade, Usabilidade, Desempenho, Segurança, Disponibilidade e Integração. Trabalhos recentes de pesquisa atuam mais fortemente em uma ou mais destas qualidades.

A maioria dos trabalhos de visualização científica via Internet são baseados no modelo proposto por UPSON et al (1989) e HABER e MCNABB (1990). Este modelo apresenta o processo de visualização como um *Pipeline*. Este *Pipeline* é iniciado com a entrada de dados que passam por processos de Filtro, Mapeamento e renderização até a geração de uma imagem final, conforme ilustrado na Figura 3.1.

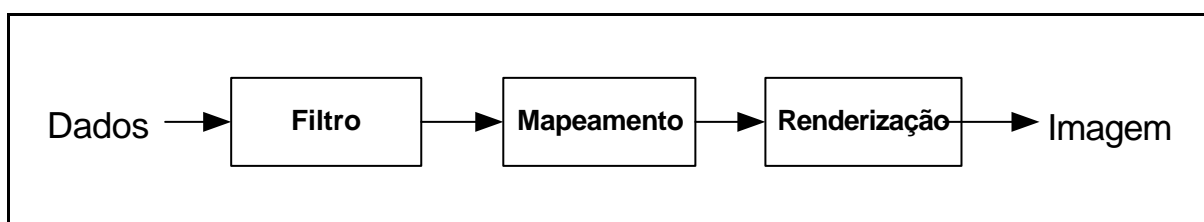


Figura 3.1: Pipeline de Visualização.

O processo de Filtro seleciona os dados de interesse no volume. O processo de Mapeamento atua nos dados normalmente até a etapa de detecção de superfícies. No processo de Renderização são aplicadas técnicas, tais como iluminação e projeção, para a geração da imagem.

Este *pipeline* é a base para criação dos sistemas de visualização por fluxo de dados, ou MVE's (*Modular Visualization Environments*). Nestes sistemas, cada módulo implementa um processo do *pipeline*. Os mais conhecidos são: *Application Visualization System* (AVS), IRIS Explorer, IBM Data Explorer e Khoros. Estes sistemas são usados como ferramenta para construção de outros sistemas de visualização específicos (GAO, 1998).

Para facilitar a compreensão das arquiteturas, os participantes principais do processo de visualização na Web são separados em Usuário, Provedor do Serviço de Visualização e Provedor dos Dados (BRODLIE, 1997) (WOOD, 1996).

O Usuário pode ser um cientista especialista ou mesmo qualquer pessoa que tenha a necessidade de acessar um sistema de visualização volumétrica via Internet. Assume-se que este usuário tenha disponível um *Web browser* ou uma aplicação que faça acesso à Internet. Em alguns casos, também é necessário ter um *plug-in VRML*<sup>8</sup> instalado em seu *Web browser*.

O Provedor de Dados é normalmente uma organização provedora de serviços de coleta e publicação de dados de interesse particular. Os Institutos de Radiologia podem ser considerados provedores de dados, pois eles fazem a coleta das informações dos pacientes e as disponibilizam para as clínicas e para os próprios pacientes.

---

<sup>8</sup> Plug-in é o software instalado no Web browser do cliente para aumentar suas funcionalidades. O plug-in VRML é o software responsável por interpretar o arquivo VRML e fazer a renderização dos objetos deste arquivo.

O Provedor do Serviço de Renderização é um serviço publicado na Internet que facilita o acesso do Usuário aos sistemas de renderização de volumes. Ele pode atuar como servidor de *download* de software para os usuários ou como canal entre o Usuário e o Provedor de Dados.

Nos Sistemas de Visualização Volumétrica é comum haver grandes quantidades de dados para compor um volume. Devido a essa quantidade de dados, o Provedor de Serviço de Visualização, em muitas arquiteturas, fica fortemente acoplado com o Provedor de Dados, ligados por um canal de alta velocidade.

No trabalho *Volume Graphics and the Internet*(BRODLIE , 2000) os sistemas de visualização de volumes via Internet são classificados em Arquiteturas com ênfase no Cliente ou no Servidor. A Figura 3.2 apresenta essa classificação.

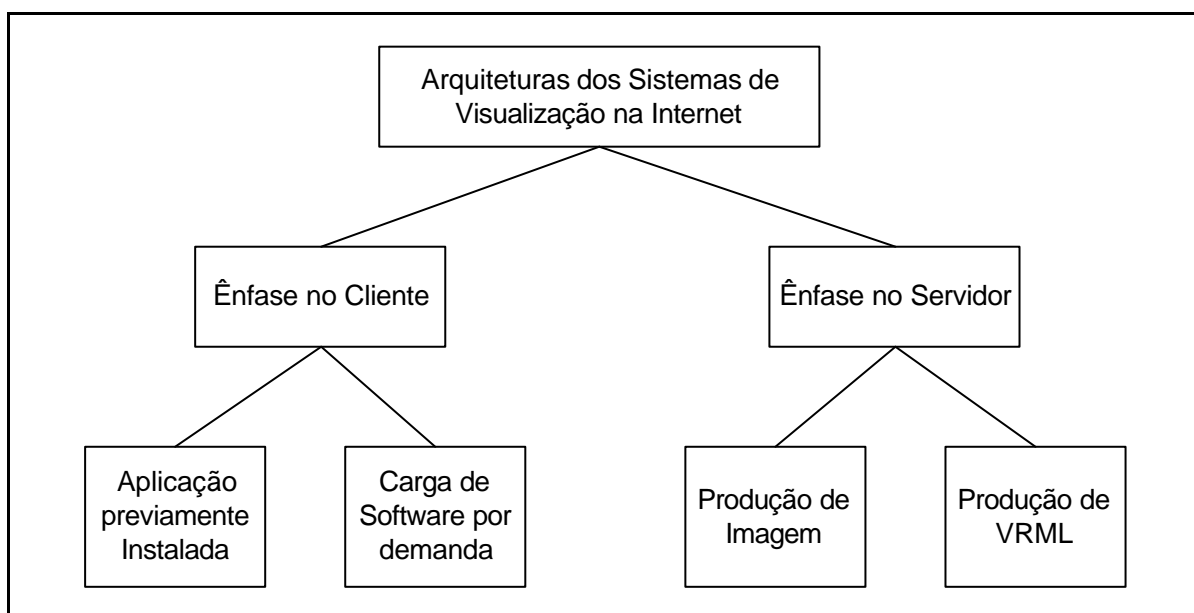


Figura 3.2: Classificação das Arquiteturas Cliente/Servidor Convencionais.

### 3.1.1 ARQUITETURAS COM ÊNFASE NO CLIENTE

Nesta categoria os dados que estão disponíveis no servidor são transportados para o cliente que faz a maior parte do processamento localmente. A Figura 3.3 exemplifica esta distribuição. Para isso, o cliente deve ter uma aplicação de visualização previamente instalada em seu equipamento, ou receber esta aplicação por demanda.

O maior benefício das aplicações previamente instaladas no cliente é que, após o carregamento dos dados, o cliente fica independente da rede para explorar o volume, com as funcionalidades específicas da sua aplicação local.

Em aplicações carregadas por demanda, o cliente solicita a visualização de um volume e recebe o software específico para visualizar este volume selecionado. Trabalhos com esta arquitetura utilizam tecnologias como *Java Applets*, que são pequenas aplicações que podem ser carregadas e executadas no *Web browser*.

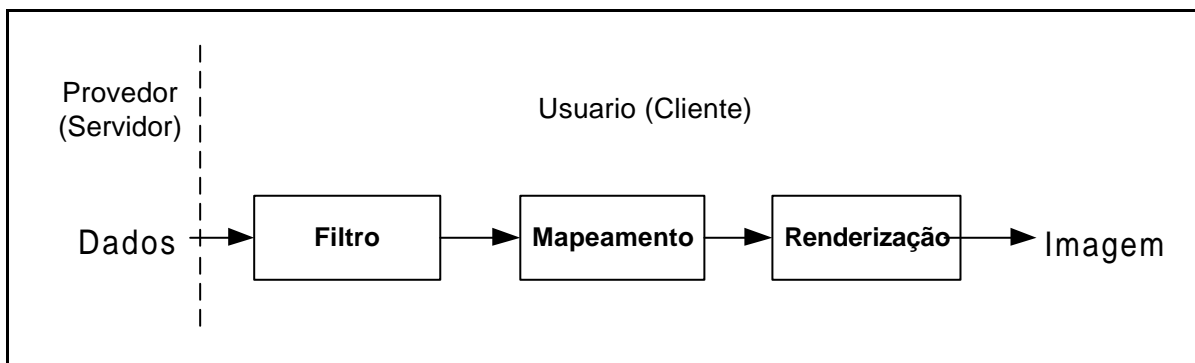


Figura 3.3: Arquitetura com ênfase no Cliente.

Esta solução aproveita os recursos computacionais do cliente, porém não fica acessível aos clientes que tenham equipamentos com poder de processamento limitado. Além disso, uma grande quantidade de dados deve ser transferida do Provedor para o Usuário antes de começar o processo de visualização.

### 3.1.2 ARQUITETURAS COM ÊNFASE NO SERVIDOR

Esta categoria é caracterizada por ter a maior parte do processamento executada pelos servidores, e por utilizar os *Web browsers* como interfaces nos clientes. Os Provedores de Serviços de Visualização desta categoria, podem retornar uma imagem ou um arquivo VRML para os clientes.

A Figura 3.4 apresenta um cenário onde o Provedor é responsável pelos dados, que são passados por processos de filtragem, mapeamento e renderização para compor uma imagem, a qual finalmente será enviada para apresentação no *Web browser* do cliente.

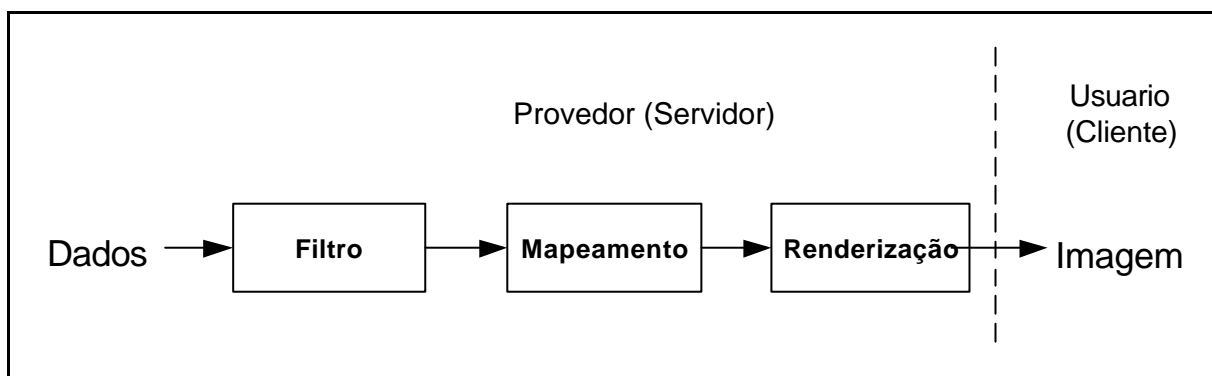


Figura 3.4: Arquitetura com ênfase no Servidor gerando imagem para o Cliente.

Esta solução é bastante usada para casos em que o usuário possui equipamentos com baixo poder de processamento, os quais não suportam aplicações mais complexas, porém, possuem *Web browser* e conexão com a Internet. A maior limitação desta abordagem é que o cliente depende totalmente do servidor para processar a visualização.

Um exemplo deste tipo de arquitetura foi desenvolvido pela *University of Califórnia, Santa Cruz (UCSC)* (WITTENBRINK, 1997). Este projeto permite acesso

a um renderizador de volumes via Web (PermWeb: *Remote Parallel and Distributed Volume Visualization*). Nele são utilizados vários tipos de algoritmos de renderização de volumes (*Ray-casting* e *Shear-Warp* com paralelismo) rodando em equipamentos tais como MasPar MP-2 com processador 4096 e várias *workstations*. O acesso ao sistema é feito por páginas HTML e o servidor utiliza scripts CGI para controlar a comunicação com os clientes.

A Figura 3.5 apresenta um cenário onde o Provedor é responsável pelos dados, os quais são passados por processos de filtragem e mapeamento para compor o arquivo VRML. Este arquivo VRML contém os objetos de representação geométrica que compõem o volume, o qual será enviado para o cliente que deve ter um *plug-in VRML* instalado em seu *Web browser*.

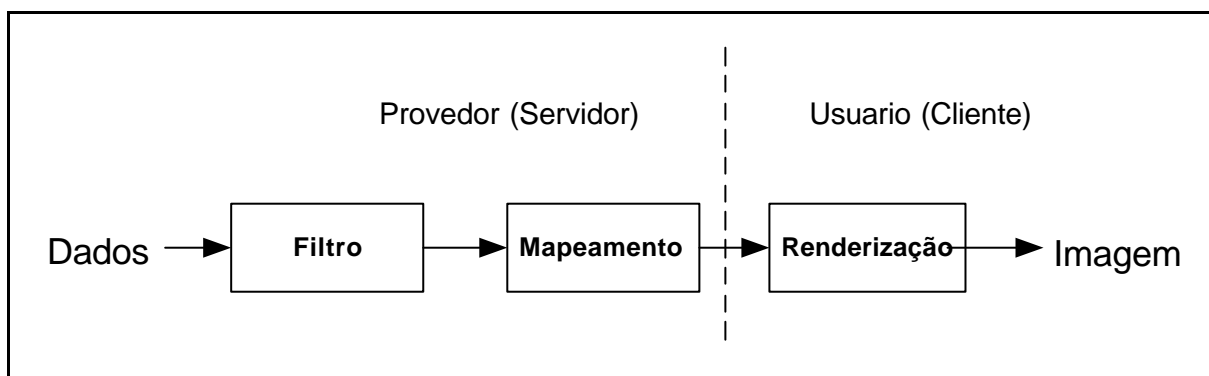


Figura 3.5: Arquitetura com ênfase no Servidor gerando arquivo VRML para o Cliente.

### 3.2 ARQUITETURAS BASEADAS EM AGENTES

Estas arquiteturas são caracterizadas por terem seu processamento realizado por agentes de visualização. Estes agentes são capazes de decidir em tempo de execução qual é o melhor local disponível (cliente ou servidor) para executarem os processos. Ao tomar esta decisão, o agente é capaz de se

movimentar entre as máquinas, levando os dados e o código necessário para processá-los.

No trabalho de Tsoi e Gröller (TSOI, 2002) é apresentado um modelo onde são criados agentes para cada processo do pipeline de visualização, além dos agentes de busca de dados e agente de visualização. Neste modelo, cada nó (equipamento por onde passam os agentes) tem um módulo que avalia constantemente o tráfego de rede e disponibilidade de CPU e Memória. Estas informações são passadas para os agentes móveis para que eles decidam o melhor nó para serem executados. A Figura 3.6 ilustra a interação destes agentes.

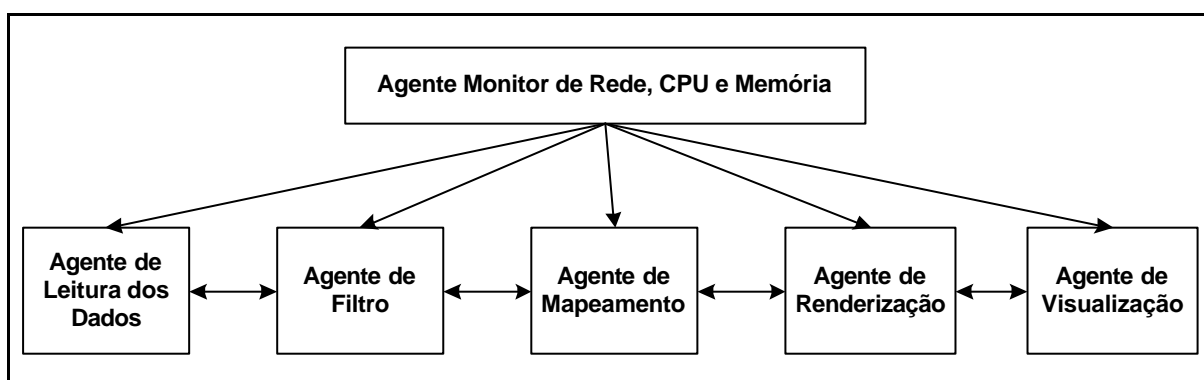


Figura 3.6: Arquitetura baseada em Agentes. Os Agentes são distribuídos entre os Clientes e Servidores disponíveis em tempo de execução.

Esta abordagem tem maior benefício quando utilizada em ambientes heterogêneos, porém exige que os equipamentos dos usuários suportem a instalação do software que gerencia os agentes móveis. Além disso, aumentam o tráfego na rede para fazerem o controle de comunicação entre os agentes.

### 3.3 VISUALIZAÇÃO COLABORATIVA VIA INTERNET

Diagnósticos médicos em doenças críticas e pesquisas científicas são raramente executadas por uma só pessoa. Estas características proporcionaram o aumento em uma nova classe de sistemas, os chamados sistemas de visualização colaborativa ou CSCV (*Computer Supported Collaborative Visualization*). Nesta categoria de sistemas os colaboradores em *sites* remotos podem simultaneamente analisar o mesmo conjunto de dados científicos (MANSSOUR, 2000).

Para suportar visualização colaborativa, o modelo de HABER e MCNABB (1990) foi expandido para ter entradas e pontos de saída intermediários para os dados e informações de controle. Para suportar participantes independentes, deve haver um *pipeline* de visualização para cada colaborador (WOOD, 1995). A Figura 3.7 ilustra o novo *pipeline*.

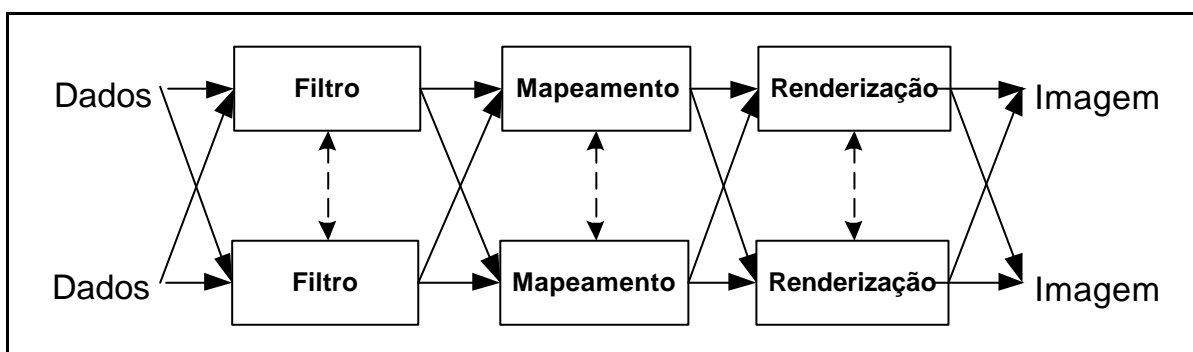


Figura 3.7: *Pipeline* de Visualização Expandido para Suportar CSCV (WOOD, 1995).

Um exemplo de sistema para visualização colaborativa pode ser encontrado no trabalho “*Volume Visualization in a Collaborative Computing Environment*” (ELVINS, 1996). Nele é apresentado o sistema experimental SDSC\_NetV, desenvolvido com técnicas avançadas de renderização em ambiente distribuído. O principal objetivo deste sistema é superar os problemas de desempenho com processamento de grande volume de dados em um ambiente compartilhado. Outros

exemplos desta categoria de sistemas para área médica podem ser encontrados no trabalho “*Collaborative Visualization In Medicine*” (MANSSOUR, 2000).

### 3.4 INTEROPERABILIDADE DOS SISTEMAS DE VISUALIZAÇÃO

Muitos trabalhos de visualização científica têm usado a Internet como base para suas arquiteturas de sistemas distribuídos. Eles geralmente atuam mais fortemente nos seguintes pontos:

- Superar as barreiras de desempenho para permitir melhor interatividade dos participantes com o sistema;
- Disponibilidade em ambientes heterogêneos;
- Balanceamento na distribuição dos processos para aproveitar os recursos disponíveis dos equipamentos envolvidos;
- Aprimoramento das funcionalidades de colaboração.

Os sistemas pesquisados alcançam um certo nível de integração, proporcionando a conexão entre os servidores e os clientes, e até conectando várias pessoas ao mesmo trabalho. Porém estes sistemas não têm interoperabilidade entre eles, pois possuem módulos fortemente acoplados, o que dificulta a integração com outros sistemas.

Arquiteturas de sistemas fortemente acoplados são baseadas em infra-estruturas, tais como *CORBA - Common Object Request Broker Architecture*, *RMI - Remote Method Invocation*, e *DCOM - Distributed Component Object Model*, as quais definem seus próprios protocolos de comunicação.

Além disso, a Internet é utilizada apenas como meio de distribuição destes sistemas. Por exemplo: Neste contexto, um especialista convidado a participar de

uma sessão de visualização colaborativa deverá contar com a seguinte infraestrutura:

- O software cliente do sistema utilizado pelo grupo;
- Uma plataforma compatível com o software;
- Treinamento adequado para poder utilizar o software.

Tais sistemas poderiam ser melhor aproveitados, se fossem projetados com base em padrões abertos para comunicação na Internet, com módulos fracamente acoplados, os quais serão vistos em detalhes no próximo capítulo.

## 4 ARQUITETURA PARA VISUALIZAÇÃO CIENTÍFICA BASEADA EM PADRÕES ABERTOS

Como mencionado anteriormente, modelos para comunicação de sistemas distribuídos, tais como RMI, DCOM e CORBA utilizados nas arquiteturas convencionais, são fortemente acoplados, e por isso, não aproveitam completamente os níveis de integração e acessibilidade oferecidos pela Internet.

Com o uso da Internet, vista como uma grande rede para sistemas distribuídos, novos modelos baseados em padrões abertos estão sendo criados para explorar melhor as características desta rede. Neste capítulo os modelos e padrões abertos para computação distribuída serão apresentados, bem como a arquitetura proposta para visualização científica via Internet.

### 4.1 PADRÕES ABERTOS PARA COMPUTAÇÃO DISTRIBUÍDA.

*Middleware* consiste de mecanismos que facilitam a comunicação entre os sistemas distribuídos. Uma arquitetura pode utilizar um ou mais destes mecanismos como solução para comunicação dos módulos do sistema (JURIC, 2001). Os mais conhecidos são:

- Tecnologias de Acesso a Banco de Dados;
- *Middleware* Orientado a Mensagens (MOM);
- Chamadas para Procedimentos Remotos (RPC);
- Agentes de Requisições de Objetos (ORBs).

As Tecnologias de Acesso a Banco de Dados utilizam uma camada abstrata de acesso aos bancos de dados, a qual permite que as aplicações utilizem um código comum para acessar diferentes bancos de dados. As mais conhecidas são: ODBC – *Open Database Connectivity* e ADO – *Active Datba Objects* da Plataforma

Microsoft e JDBC – *Java Database Connectivity* e JDO – *Java Data Objects* da Plataforma Java.

O *Middleware Orientado a Mensagens* (MOM) é uma infra-estrutura cliente/servidor que permite a comunicação entre aplicações em plataformas heterogêneas por meio de *API*<sup>9</sup> - *Application Programming Interface*, as quais encapsulam detalhes de comunicação e protocolos envolvidos.

O MOM trabalha com comunicação assíncrona, usando filas de mensagens distribuídas no cliente e no servidor. Estas mensagens suportam diversos tipos de dados. O maior benefício desta comunicação é a garantia de entrega da mensagem mesmo se o cliente ou o servidor estiverem temporariamente inativos.

As Chamadas para Procedimentos Remotos (RPC), assim como os MOM, são infra-estruturas cliente/servidor que permitem a comunicação entre aplicações em plataformas heterogêneas encapsulando detalhes de comunicação e protocolos. As RPCs trabalham com comunicação síncrona, a qual bloqueia o cliente até que a sua solicitação seja atendida pelo servidor. Neste modelo, os clientes e servidores envolvidos não podem estar temporariamente inativos no momento da transmissão dos dados.

Os Agentes de Requisições de Objetos (ORBs) são tecnologias que gerenciam a comunicação entre componentes ou objetos distribuídos. Eles encapsulam os detalhes de protocolos e comunicação entre os objetos, oferecem facilidades para localização e heterogeneidade de plataforma.

Os ORBs normalmente trabalham com comunicação síncrona. A independência de linguagem de implementação é alcançada com a utilização de uma linguagem comum para definição da interface entre o cliente e o servidor. Esta

---

<sup>9</sup> API - *Application Programming Interface* é um conjunto de funções de software usadas por um programa para facilitar o acesso ao software principal.

interface define o conjunto de métodos que serão implementados no servidor e que poderão ser acessados pelo cliente.

Os principais padrões de ORBs são o OMG CORBA ORB; Java RMI e RMI-IIOP e Microsoft COM/DCOM/COM+. Dentre eles, o CORBA é o que mais tem se destacado como solução para computação distribuída em sistemas de visualização científica, e será descrito a seguir.

#### 4.1.1 CORBA

*CORBA – Common Object Request Broker Architecture* é um padrão aberto para computação distribuída definido pelo *OMG – Object Management Group*. Baseado no protocolo IOP - *Internet Inter-ORB Protocol*, programas CORBA podem se comunicar com outros programas CORBA independentemente da linguagem de programação, sistema operacional, e localização na rede (OMG, 2001).

Os três principais componentes do CORBA são: O servidor, o cliente e uma camada intermediária que conecta os dois conforme ilustrado na Figura 4.1. Essa camada que conecta o cliente ao servidor é o *ORB - Object Request Broker*. E os clientes acessam os servidores pelas interfaces que estes servidores publicam através da *IDL – Interface Description Language*, é com ela que são especificados os comportamentos dos objetos.

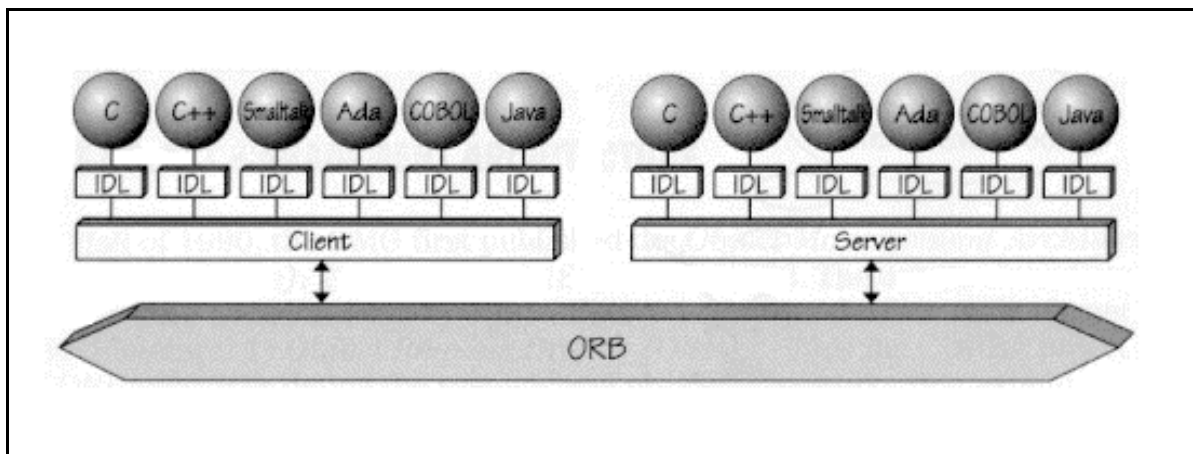


Figura 4.1: Comunicação Cliente/Servidor por meio do ORB. Extraído de: (ORFALI, 1997).

Embora seja baseada em padrões abertos, CORBA é uma arquitetura fortemente acoplada e requer a aquisição de uma implementação CORBA e código compilado com bibliotecas específicas para estabelecer comunicação com outras aplicações.

#### 4.1.2 WEB SERVICES

*Web Services* foram criados para facilitar a interação entre as aplicações na Web, e a integração das diversas plataformas e modelos de programação existentes na Web (CURBERA, 2001).

*Web Services* são componentes de software criados a partir de conjuntos de protocolos e padrões abertos para computação distribuída na Internet definidos pela W3C – *World Wide Web Consortium*. A comunicação baseada nestes padrões permite que as aplicações descrevam o que fazem, podendo então chamar ou utilizar os serviços de outras aplicações (SOWEK, 2002).

As características mais importantes nos *Web Services* são:

- Independência de implementação e plataforma: *Web Services* podem ser criados e acessados em qualquer plataforma e linguagem de programação que suporte leitura e escrita de arquivos XML;
- Permite comunicação Síncrona e Assíncrona;
- As mensagens podem ser enviadas sobre o protocolo HTTP permitindo a sua passagem pelos *Firewalls* sem necessidade de configurações de portas especiais;
- Arquitetura Fracamente Acoplada, ou seja, é Independente de protocolos e códigos específicos para controle da transferência das informações;
- Reutilização dos serviços. Os serviços existentes podem ser utilizados para compor um novo serviço;
- Modelo *SOA – Service-oriented Architecture*. Utiliza o modelo de Arquitetura Orientada a Serviço, no qual o software desenvolvido pode ser publicado, localizado e acessado como um serviço disponível na Web (GRAHAM, 2001).

Os principais participantes do modelo de Arquitetura Orientada a Serviço são: O Provedor de Serviço, o Cliente do Serviço e o Serviço de Registro. Este modelo é ilustrado na Figura 4.2.

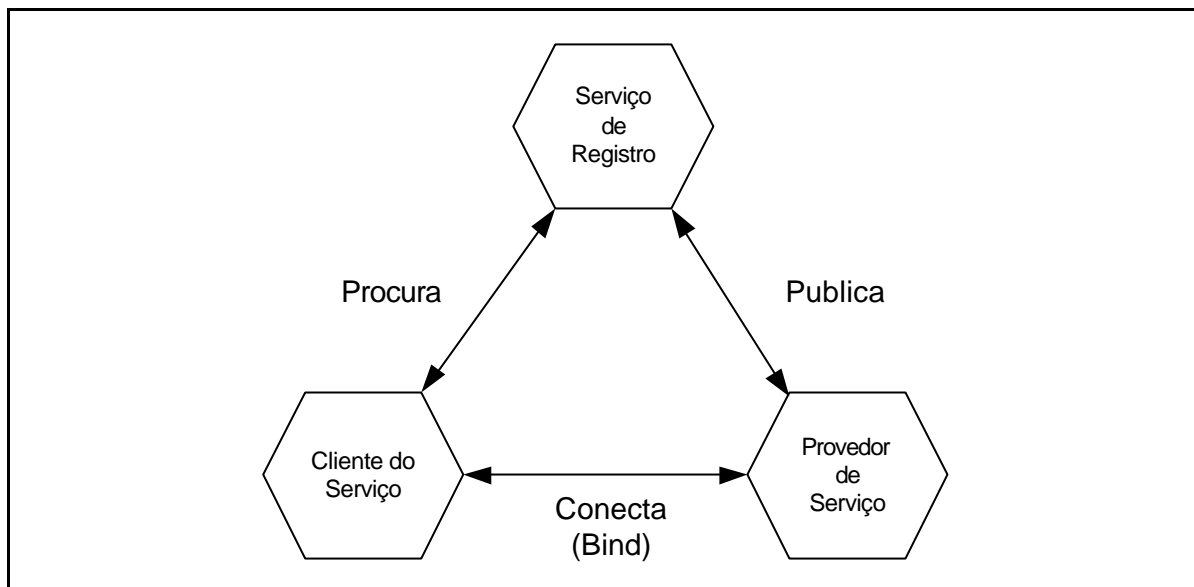


Figura 4.2: SOA – *Service-oriented Architecture* (GRAHAM, 2001).

O Provedor de Serviço é responsável por criar uma descrição do serviço, publicá-la em um ou mais servidores de registro, e receber requisições de mensagens vindas de um ou mais Clientes do Serviço.

O Cliente do Serviço é responsável por procurar uma descrição de serviço publicada em um ou mais Serviços de Registros, e é responsável por usar esta descrição do serviço para fazer conexão com um *Web Service* localizado no Provedor de Serviço.

O Serviço de Registro é responsável por disponibilizar as descrições de serviços que foram publicadas pelos Provedores de Serviços. Isto permite que os Clientes do Serviço possam pesquisar quais serviços estão registrados e acessá-los.

*Web Services* utilizam os padrões baseados em *eXtensible Markup Language* (XML), incluindo: *Simple Object Access Protocol* (SOAP), *Universal Description, Discovery, and Integration* (UDDI), *Web Services Description Language* (WSDL) e outros protocolos para permitir a comunicação entre as aplicações. Estes padrões são detalhados a seguir.

#### 4.1.2.1 XML

*XML – eXtensible Markup Language* foi aprovada pela W3C em 1998, e é uma simplificação da SGML – *Standard Generalized Markup Language*, projetada para facilitar a troca de documentos estruturados na Internet. XML é considerada uma metalinguagem, pois é capaz de definir outras linguagens.

Um dos principais objetivos de XML é ser capaz de descrever informações e metadados, ou seja, dados que descrevem outros dados. Além disso, se beneficia de um conjunto de mecanismos para tratar a Estrutura, a Apresentação, o Processamento e a Manipulação destes dados (COYLE, 2002).

Um documento XML é basicamente constituído de elementos textuais definidos por *tags*. Estes elementos são estruturados de forma hierárquica. Um elemento pode ainda, conter atributos, os quais são usados para fornecer informações adicionais sobre este elemento. A Figura 4.3 apresenta um exemplo de informações de um paciente, definidas em um documento XML.

```
<?xml version="1.0" ?>
<PACIENTE>
  <ID>P123</ID>
  <NOME TRATAMENTO="Sr.">Joao da Silva</NOME>
  <DATA_NASC>10/02/1972</DATA_NASC>
  <ENDERECO>
    <RUA>Central, 234</RUA>
    <CEP>80200010</CEP>
    <CIDADE>Curitiba</CIDADE>
  </ENDERECO>
</PACIENTE>
```

Figura 4.3: Exemplo de um documento no formato XML.

Neste exemplo, a primeira linha indica que este é um documento XML. O primeiro elemento do documento é o elemento-raiz: <PACIENTE>, o qual é

composto pelos elementos descendentes: <ID>, <NOME>, <DATA\_NASC>, e <ENDEREÇO>. O elemento <ENDEREÇO> é composto dos elementos <RUA>, <CEP> e <CIDADE>. O elemento <NOME> contém o atributo TRATAMENTO.

Para trocar informações entre sistemas, é necessário que estas informações estejam organizadas em uma especificação comum. Para isso existem os *DTDs* e *XML Schemas*. Eles são mecanismos utilizados para estruturar e especificar quais elementos são válidos, além de definir os tipos de dados que podem ser utilizados em um XML.

Para interpretar e validar um documento XML são utilizados mecanismos *XML Parsers*. Os *Parsers* são capazes de ler e extrair informações armazenadas na hierarquia de elementos de um documento XML. Os principais *Parsers* são: *DOM – Document Object Model* e o *SAX – Simple API for XML*.

#### 4.1.2.2 SOAP

*SOAP – Simple Object Access Protocol* é um protocolo baseado em XML para troca de informações em um ambiente distribuído e descentralizado (BOX, 2000). SOAP encapsula a mensagem XML em um envelope e transporta esta mensagem por meio de protocolos padrões, tais como: HTTP, FTP e SMTP.

Uma mensagem SOAP é um documento XML composto de três partes: Envelope, Header e Body. O <Envelope> é o primeiro elemento da mensagem SOAP e é composto por dois elementos: o <Header> e o <Body>. O elemento <Header> é opcional, e é usado para especificar informações adicionais sobre a mensagem.

O elemento <Body> contém a mensagem XML que pode ser uma requisição de um cliente ou a resposta de um servidor. A especificação SOAP descreve como uma chamada RPC pode ser mapeada em uma estrutura XML. A Figura 4.4

apresenta um exemplo da chamada do método `getNome` do objeto `Paciente`, passando o parâmetro `ID` com o valor "P123".

```
<SOAP-ENV:Envelope ...>
  <SOAP-ENV:Header>...</SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <getNome>
      <ID xsi:type="xsd:string">P123</ID>
    </getNome>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figura 4.4: Chamada de um método usando o protocolo SOAP.

Uma possível resposta da chamada anterior em SOAP é exemplificada pela

Figura 4.5.

```
<SOAP-ENV:Envelope ...>
  <SOAP-ENV:Header>...</SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <getNomeResponse>
      <getNomeResult xsi:type="xsd:string">Sr. Joao da
Silva</getNomeResult>
    </getNomeResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figura 4.5: Resposta de um método usando o protocolo SOAP.

#### 4.1.2.3 WSDL

*WSDL – Web Services Description Language* é uma especificação padrão para a descrição de *Web Services* com base em XML. WSDL descreve as operações que um *Web Service* pode executar, os formatos de mensagens que ele pode processar, os protocolos que ele suporta, e a localização para acesso da instância deste *Web Service* (SYSTINET, 2001).

Assim como IDL define a interface para um componente CORBA na forma de uma linguagem independente, WSDL usa XML para definir a interface para um *Web Service* (JURIC, 2001).

WSDL é um documento XML que usa os seguintes elementos para definição dos serviços na rede:

- <Types>, representa os tipos de dados usados: float, integer, string, etc. WSDL usa a especificação de *XML Schema* para codificar os tipos de dados.
- <Message>, é o elemento que define o formato de uma mensagem. As mensagens são usadas como estruturas de entrada e saída para operações.
- <Operation>, uma descrição abstrata de uma ação suportada pelo serviço.
- <PortType>, um conjunto abstrato de operações suportadas pelo serviço.
- <Binding>, realiza dois mapeamentos com as operações e mensagens definidas no PortType. O primeiro mapeamento define um protocolo concreto (HTTP, SMTP, ou outro protocolo específico). O segundo define a o estilo de requisição (RPC ou documento).
- <Port>, um ponto de acesso único, definido como uma combinação de um binding e um endereço de rede.
- <Service>, uma coleção de portas relacionadas.

#### 4.1.2.4 UDDI

*UDDI – Universal Description, Discovery, and Integration* é uma especificação que define uma maneira de publicar e descobrir informações sobre *Web Services* (UDDI, 2000).

Na arquitetura orientada a serviço (SOA), os Provedores de Serviço precisam de um mecanismo para registrar e publicar os *Web Services*, e os Clientes precisam dispor de um mecanismo para encontrar estes serviços, um UDDI oferece um repositório centralizado para possibilitar tais mecanismos.

O próprio UDDI é um *Web Service*. Os usuários acessam o UDDI usando mensagem SOAP. UDDI pode ser dividido em duas funcionalidades principais: A Publicação e o Descobrimento. A Publicação é a parte que permite um Provedor de Serviço registrar informações sobre seu negócio e seus serviços, por meio de uma estrutura hierárquica em XML. A figura 4.6 apresenta esta estrutura.

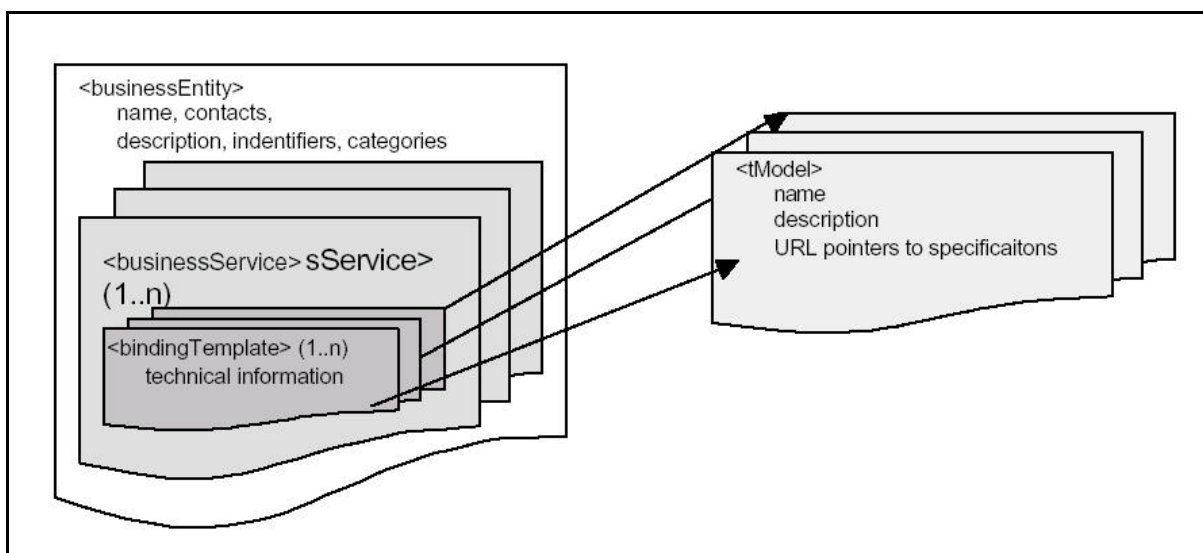


Figura 4.6: Hierarquia dos elementos XML em UDDI. Extraído de: (UDDI, 2000).

<businessEntity> é o elemento que descreve informações do negócio em geral incluindo nome e uma breve descrição da organização, endereço, informações de contato e classificação negocial.

<businessService> são elementos que descrevem os *Web Services* oferecidos pela organização. Um elemento <businessService> contém informações do serviço e uma lista de categorias que descrevem o serviço.

<bindingTemplate> são elementos que contêm uma ou mais especificações técnicas sobre os serviços. Normalmente estas especificações técnicas são detalhadas em um documento WSDL.

<tModel> são elementos que definem tipos de serviços abstratos, ou seja, vários negócios podem oferecer o mesmo tipo de serviço e com a mesma interface, o tModel é usado para descrever estes tipos de serviços.

Para descobrir um serviço, um cliente pode navegar nos registros de um UDDI pesquisando por categorias negociais e outros detalhes até encontrar o serviço desejado. Após encontrar o serviço, o cliente pode obter o documento WSDL com especificações técnicas e detalhes para fazer a conexão com este serviço.

#### 4.2 A INFLUÊNCIA DOS PADRÕES ABERTOS NA ARQUITETURA.

Em *XML, Web Services, and Data Revolution*, COYLE (2002) apresenta os impactos da utilização do XML sobre três aspectos: Dados, arquitetura e software. Como revolução dos dados, observa-se que os dados em XML são transportados na rede usando protocolos padrões Web, sem a necessidade de infra-estruturas específicas para tratar da formatação e transporte destes dados.

A revolução de software está relacionada à capacidade de reaproveitamento dos serviços distribuídos na rede. A criação de um novo serviço na web pode ser

feita pela composição de outros serviços já disponíveis. Como revolução de Arquitetura, observam-se os benefícios de integração e reaproveitamento dos serviços gerados pelas arquiteturas com módulos fracamente acoplados.

Diante das possibilidades apresentadas pelos padrões abertos para computação distribuída, e das necessidades de integração dos sistemas de visualização científica em medicina, o uso do conceito de *Web Services* é apresentado como uma solução para criação de um novo tipo de Arquitetura. Esta arquitetura deve considerar as seguintes premissas:

- Mudança do paradigma de desenvolvimento de sistemas para o desenvolvimento de serviços. Os sistemas fechados passam a ser serviços distribuídos na Internet;
- Acessibilidade: Para permitir que o serviço seja acessível ao maior número de usuários possíveis, são necessários mecanismos de publicação e busca destes serviços;
- Adaptabilidade: Atender aos diversos tipos de equipamentos disponíveis na rede, aproveitando os recursos disponíveis da melhor forma possível;
- Integração, os serviços desenvolvidos devem ser facilmente integrados a qualquer sistema que tenha a necessidade de utilizá-los;
- Reaproveitamento: Maior granularidade e reaproveitamento de código, ou seja, os serviços podem ser compostos por outros serviços já existentes;
- Portabilidade, as aplicações devem ser implementadas e executadas para a maioria das plataformas disponíveis na Internet;
- Baseada em padrões abertos para computação distribuída na Internet, especificados pela *W3C – World Wide Web Consortium*.

Existem outras características também importantes em sistemas na área de medicina, tais como segurança, autenticação e autorização e controle de sessão, as quais fogem do escopo deste trabalho, e devem ser tratados em futuros desenvolvimentos.

### 4.3 A ARQUITETURA PROPOSTA

Esta arquitetura pode ser melhor compreendida por meio do diagrama do tipo *deploy*<sup>10</sup> da notação UML<sup>11</sup>. A Figura 4.7 apresenta este diagrama.

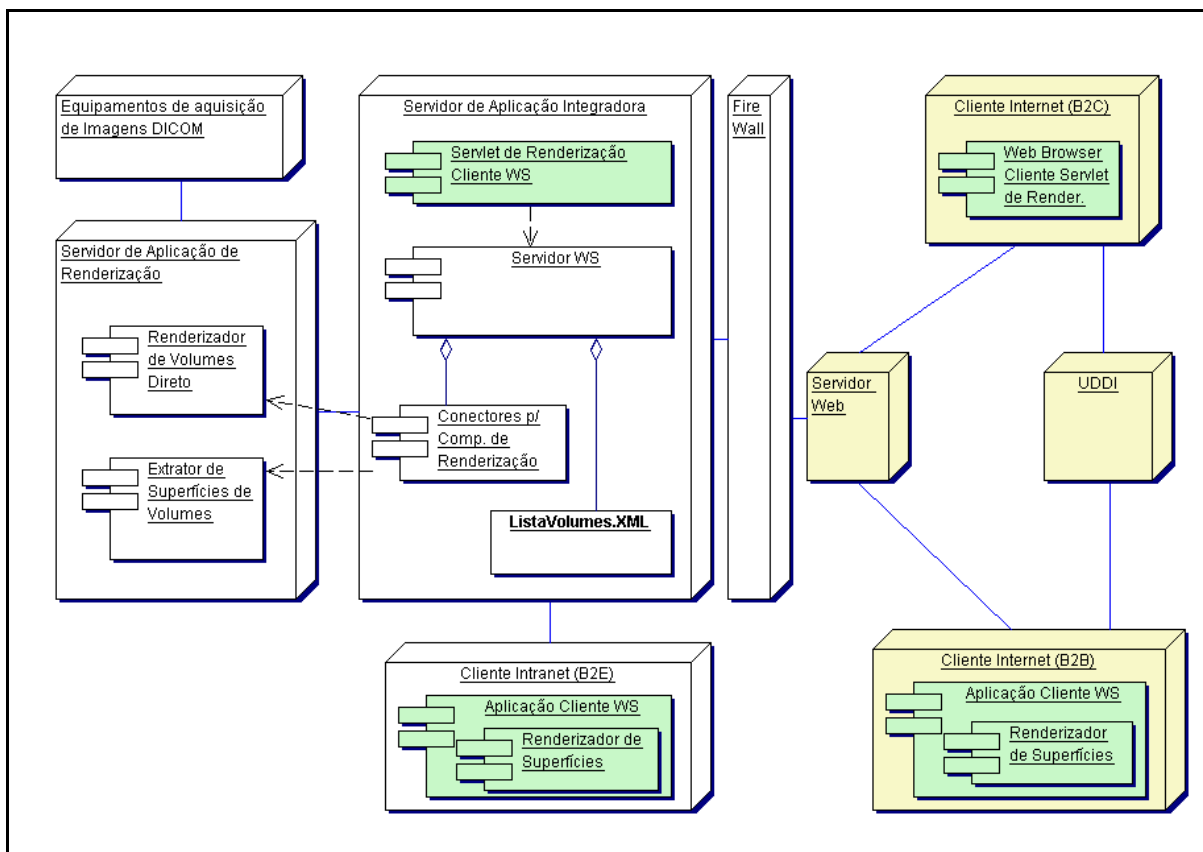


Figura 4.7: Diagrama dos componentes da Arquitetura.

Os módulos que estão apresentados em amarelo são aqueles que estão no domínio da Internet. Os que estão em verde representam os clientes.

<sup>10</sup> *Deploy* é o diagrama da UML – *Unified Modeling Language* usado para representar o relacionamento entre os componentes de um projeto.

<sup>11</sup> UML – *Unified Modeling Language* ou “Linguagem de Modelagem Unificada” é uma padronização da modelagem orientada a objetos.

No diagrama apresentado na Figura 4.7, os *equipamentos de aquisição de imagens DICOM* são representados por qualquer equipamento que possa extrair imagens médicas e exportá-las no formato DICOM. Exemplos destes equipamentos são os de MRI e de CT.

O Servidor de Aplicação de Renderização é o equipamento onde estão instalados os renderizadores de Volumes Direto e o Extrator de Superfícies de Volumes, os quais são sistemas já existentes na organização.

O *Renderizador de Volumes Direto* atende a solicitações de renderização de seus clientes, utilizando técnicas de renderização de volumes direto para compor uma imagem resultante, a qual representa o volume. Este componente tem como parâmetros de entrada um conjunto de imagens DICOM e especificações de visualização definidas pelo cliente.

O *Extrator de Superfícies de Volumes* é semelhante ao Renderizador de Volumes Direto, porém, em seu algoritmo, ao invés de gerar uma imagem do volume, ele gera um arquivo contendo a representação da superfície do volume por meio de uma malha de polígonos.

Partindo-se do princípio que a organização já tem estes módulos disponíveis, a solução de integração e acessibilidade proposta pela Arquitetura é alcançada com a inclusão do *servidor de aplicação integradora*. Este servidor é composto pelos módulos Servidor WS, Conectores para Componentes de Renderização, ListaVolumes.XML e Servlet de Renderização, que são descritos a seguir.

*Servidor WS (Web Services)*, é o componente responsável por receber as solicitações de seus clientes internos ou externos através do protocolo SOAP, encaminhar estas solicitações para o *Servidor de Aplicação de Renderização* e devolver os resultados para os clientes.

O módulo de *Conectores para Componentes de Renderização* está agregado ao Servidor WS e tem por objetivo fazer a ponte de acesso entre o Servidor WS e os componentes de renderização do Servidor de Aplicação de Renderização.

Existem várias soluções para implementação destes conectores. Elas dependem muito do ambiente e linguagem de programação utilizada nos dois servidores. As soluções mais comuns são a utilização de RMI, CORBA, COM e APIs.

A *ListaVolumes.XML* é um objeto que contém informações dos Volumes DICOM que estão disponíveis para renderização, e informações complementares sobre a calibragem destes volumes no formato XML.

Esta arquitetura permite dois tipos de clientes: Os *Thin Clients* e os *Fat Clients*. Os *Thin Clients* são aqueles com poder de processamento limitado e possuem apenas um *Web browser* para acesso à Internet, podendo ser até mesmo PDAs<sup>12</sup>.

Os *Fat Clients* são aqueles com maior poder de processamento, geralmente dotados de placas de aceleração gráfica. Eles são compostos por uma *aplicação Cliente WS* que possui um componente *Renderizador de Superfícies* que lhes permite receber o arquivo de polígonos gerado pelo *Extrator de Superfícies de Volumes* e realizar a renderização localmente.

O *Servlet*<sup>13</sup> de *Renderização – Cliente WS* atua como um dos clientes do Servidor WS e como servidor dos *Thin Clients*. Uma vez que estes clientes têm recursos limitados, o *Servlet* de *Renderização* tem o código necessário para

---

<sup>12</sup> PDA - Os computadores portáteis são chamados genericamente de PDAs - *Personal Digital Assistants*, ou Assistentes Pessoais Digitais. Eles geralmente são equipamentos de tamanho reduzido e com recursos limitados de hardware.

<sup>13</sup> Servlet é uma solução Java para implementação de CGIs. CGI (*Common Gateway Interface*) é uma aplicação servidora usada para atender solicitações de clientes Internet.

trabalhar com o servidor WS e retornar os resultados já convertidos em HTML para os clientes.

O *Cliente Internet B2C (Business to Consumer)* é um cliente externo que acessa o serviço de renderização de Volumes por meio da Internet. Exemplos destes clientes são as Clínicas Médicas conveniadas, que podem acessar o renderizador da organização utilizando funções simples como rotação para investigar melhor um determinado caso.

O *Cliente Internet (Business to Business)* é um cliente externo que acessa o Serviço de Renderização de Volumes por meio da Internet. Exemplos destes clientes são os Especialistas de outras organizações que podem atuar na colaboração para solução de um caso.

O *Cliente Intranet (Business to Employees)* é um cliente interno da organização que acessa o Serviço de Renderização por meio da Intranet. Estes clientes são os próprios funcionários da organização.

As requisições da Internet chegam pelo *Servidor Web*, são passadas pelo *FireWall* e encaminhadas para o *servidor de Aplicação Integradora*. Estas requisições, após o seu processamento, voltam pelo mesmo caminho até chegarem aos clientes.

O *UDDI (Universal Description, Discovery, and Integration)* representa o servidor na Internet que contém os registros dos serviços de renderização volumétrica disponíveis. Ele recebe as solicitações dos clientes e as repassa para o local onde se encontra o serviço.

### 4.3.1 COMPOSIÇÃO DO ARQUIVO XML COM INFORMAÇÕES DE VOLUMES

Os arquivos DICOM são estruturados em diretórios que definem a hierarquia das informações do paciente, as quais são subdivididas em estudo, série e imagens. A série é um agrupamento de imagens que representam as fatias para constituir um volume. Desta forma, um estudo pode conter mais de um volume.

Na arquitetura proposta, estas informações são necessárias para informar ao cliente quais são os volumes disponíveis para visualização. A Figura 4.8 apresenta uma ilustração dessa estrutura de diretórios, onde:


\DICOM – é o diretório-raiz.

\DICOM\PA1 – é o paciente “PA1”

\DICOM\PA1\ST1 – é o estudo 1 “ST1” do paciente.

\DICOM\PA1\ST1\SE1 – é a série “SE1”.

\DICOM\PA1\ST1\SE1\IM1 – é a imagem.



```
\DICOM\PA1\ST1\SE1\IM1
\dicom\pa1\st1\se1\im2
...
\dicom\pa1\st1\se1\imn
```

Figura 4.8: Hierarquia das informações DICOM estruturadas em diretórios.

No processo de renderização, além destas informações de localização das imagens (fatias do volume), são necessárias informações de pré-calibragem, as quais descrevem a melhor forma para visualização inicial do volume. Quando o cliente seleciona um volume para visualização, estas informações são usadas pelo renderizador para fazer o primeiro processamento. Nas próximas solicitações elas

são alteradas e transportadas entre o cliente e o renderizador. A Figura 4.9 exemplifica estas informações.

```
Materiais=  
Ar;0.0;22.0;0.0;0.8;0.8;0.8;0.0010  
Gordura;17.0;40.0;0.2;0.8000000558793557;0.8000000558793557;0.0;0.02  
Musculo;35.0;60.0;0.5;1.0;0.0;0.0;0.05  
Osso;55.0;1000.0;1.0;1.0;1.0;0.800000044703483;0.1  
rotz=0.0  
roty=0.0  
rotx=45.0  
ka=0.0  
observador.k=1.0  
observador.j=0.0  
observador.i=0.0  
ii=1.0  
n=5.0  
ia=0.0  
luz.k=0.5773503184318542  
luz.j=0.5773503184318542  
ks=0.4000000059604645  
luz.i=0.5773503184318542
```

Figura 4.9: Exemplo das informações de pré-calibragem definidas para um volume.

As informações de materiais (Ar, Gordura, Músculo e Osso) indicam ao renderizador como será feito o processo de classificação das imagens. As demais informações indicam rotação do volume, vetor da posição do observador, vetor da posição da luz, e outras grandezas usadas no processo de iluminação. É importante observar que estas informações podem variar de acordo com o renderizador utilizado.

Visto que as informações DICOM de hierarquia das imagens e as informações de renderização são trafegadas entre o cliente e o servidor, detectou-se a necessidade da criação de um arquivo no formato XML que agrupa todas estas informações em uma lista de volumes disponíveis. A Figura 4.10 apresenta uma ilustração deste arquivo.

```

<SERVIDOR>
  <ID>DAPIServer1</ID>
  <IP>168.144.25.198</IP>
  <PACIENTE>
    <NOME>Joao da Silva</NOME>
    <ESTUDO>
      <NOME>Estudo 1</NOME>
      <SERIE>
        <PARAMETROS>
          <MATERIAIS>
            <AR>0.0;22.0;0.0;0.8;0.8;0.8;0.0010</AR>
            ...
          </MATERIAIS>
          <ROTZ>0.0</ROTZ>
          <ROTX>0.0</ROTX>
          <ROTY>45.0</ROTY>
          ...
        </PARAMETROS>
        <NOME>Serie 1</NOME>
        <ARQUIVO>C:\PA1\ST1\SE1\Im1</ARQUIVO>
        <ARQUIVO>C:\PA1\ST1\SE1\Im2</ARQUIVO>
        ...
        <ARQUIVO>C:\PA1\ST1\SE1\ImN</ARQUIVO>
      </SERIE>
    </ESTUDO>
  </PACIENTE>
  ...
</SERVIDOR>

```

Figura 4.10: Exemplo do arquivo ListaVolumes.XML

Este arquivo pode estar disponível em um diretório no provedor de renderização, ou ainda, para aumentar o desempenho do processo de busca, pode estar armazenado em uma base de dados. Neste trabalho, optou-se por utilizar o arquivo XML, que permite maior portabilidade em ambientes heterogêneos e não exige a instalação de um banco de dados.

A construção e liberação deste arquivo XML pela instituição provedora de renderização, indicando quais volumes estão disponíveis, é vista como uma preparação inicial de todo o processo da visualização na arquitetura proposta. Este processo será detalhado a seguir.

#### 4.3.2 PUBLICAÇÃO DO SERVIÇO E O ACESSO REMOTO

Para que o serviço fique disponível aos *consumidores*, ele deve ser publicado em algum UDDI disponível na Internet. Como o serviço de registro UDDI é um Web Service, ele é acessado por meio de uma aplicação cliente WS, cuja tarefa é registrar as informações necessárias para publicação do serviço. São exemplos destas informações:

- O nome do serviço que está sendo publicado;
- A categoria deste serviço;
- O seu arquivo WSDL.

Depois de publicado, o serviço poderá ser consultado por qualquer consumidor que acesse um UDDI disponível. Quando o consumidor encontra o serviço desejado, ele recebe as informações WSDL que indicam exatamente o local onde o serviço está publicado. Ele poderá então fazer um *bind*<sup>14</sup> direto para o serviço e executar os métodos disponíveis. Os métodos disponíveis no serviço são:

- ListaVolumes getListaVolumes();
- Volume getVolume(Paciente, Estudo, Serie);
- ArqIMG getRenderImagem(Volume);
- ArqPOL getRenderPoligonos(Volume);
- ArqDICOM getArqDICOM(Volume, Imagem);
- ConjArqDICOM getVolumeDICOM(Volume).

---

<sup>14</sup> Bind (conexão) é o ato em que o cliente faz uma conexão com um servidor.

Um possível cenário na execução do serviço é detalhado da seguinte maneira: O consumidor, após ter feito o *bind* no serviço, invoca o método `getListaVolumes`, o qual processa uma consulta na base de volumes disponíveis do provedor do serviço e retorna uma Lista de Volumes simplificada. Um exemplo desta lista de volumes simplificada é apresentada na Figura 4.11.

```
<SERVIDOR>
  <ID>DAPIServer1</ID>
  <IP>168.144.25.198</IP>
  <PACIENTE>
    <NOME>Joao da Silva</NOME>
    <ESTUDO>
      <NOME>Estudo 1</NOME>
      <SERIE>
        <NOME>Serie 1</NOME>
      </SERIE>
    </ESTUDO>
  </PACIENTE>
  <PACIENTE>
    <NOME>Maria da Silva</NOME>
    <ESTUDO>
      <NOME>Estudo 1</NOME>
      <SERIE>
        <NOME>Serie 1</NOME>
      </SERIE>
    </ESTUDO>
  </PACIENTE>
</SERVIDOR>
```

Figura 4.11: Exemplo da Lista de Volumes Simplificada, no formato XML.

Esta lista contém as informações básicas do servidor de renderização, e a hierarquia das informações dos volumes, que são:

- nome do paciente;
- nome do estudo;
- nome do volume.

Recebendo esta lista, o usuário seleciona um dos volumes e requisita todas as informações referentes a este volume por meio do método `getVolume`, o qual retorna o objeto volume.

O objeto volume contém todas as informações da composição do volume, e também os parâmetros de renderização que podem ser alterados pelo cliente. O cliente, de posse deste objeto, pode então solicitar um destes métodos ao serviço:

- `getRenderImagem`, para solicitar uma imagem resultante da renderização direta do volume. Exemplo: Ray-casting.
- `getRenderPoligonos`, para solicitar um arquivo poligonal resultante da extração de superfícies do volume. Exemplo: Marching Cubes.
- `getArqDICOM`, para solicitar uma imagem original (sem processamento do renderizador) no formato DICOM.
- `getVolumeDICOM`, para solicitar todas as imagens originais (sem processamento do renderizador) de um Volume no formato DICOM.

A extensibilidade da arquitetura é sustentada com a utilização de tecnologias abertas e padronizadas. Por exemplo, detectou-se a necessidade de incluir um método chamado “`MostraPreVisualizacaoDoVolume`” que retorna uma pequena imagem do volume para auxiliar o cliente no processo de seleção dos volumes.

Para realizar esta implementação, basta que o desenvolvedor adicione o novo método à classe principal da arquitetura. As informações referentes à imagem de pré-visualização podem ser incluídas no `ListaVolumes.XML` sem que afetem as implementações já realizadas.

#### 4.4 ABRANGÊNCIA DA ARQUITETURA

Esta arquitetura pode atuar em diferentes etapas do *pipeline* de visualização. Quando o cliente acessa o serviço por meio de um equipamento limitado, ele atua como um *Thin Client*, recebendo apenas as imagens resultantes do processo que é executado no lado do Servidor.

Quando o Cliente dispõe de um equipamento com maior poder de processamento, ele pode atuar como um *Fat Client*, recebendo apenas os arquivos no formato VRML para processar a etapa de renderização localmente, ou receber todos os arquivos DICOM necessários para realização de todo o *pipeline* de visualização localmente.

Além disso, os componentes de renderização do Servidor de Aplicação de Renderização podem atender a qualquer etapa do pipeline, permitindo que o cliente fique responsável pelo restante do pipeline. Dessa forma a arquitetura fica mais flexível e reutilizável.

## 5 O DESENVOLVIMENTO DE UM PROTÓTIPO

Neste capítulo serão apresentados detalhes da criação de um protótipo simplificado baseado na arquitetura proposta. O objetivo principal da construção deste protótipo é demonstrar a viabilidade da utilização desta arquitetura para o desenvolvimento de serviços de visualização volumétrica via Internet.

Um possível cenário onde o protótipo poderia ser utilizado é apresentado da seguinte maneira: Um instituto de radiologia pretende disponibilizar serviços de renderização volumétrica para que os seus funcionários especialistas e as clínicas conveniadas possam acessar tais serviços remotamente por meio de um simples *Web browser* ou aplicações mais complexas. Estes clientes podem ter necessidades diferentes e ambientes computacionais heterogêneos para acessar este serviço. Este cenário é ilustrado na Figura 5.1.

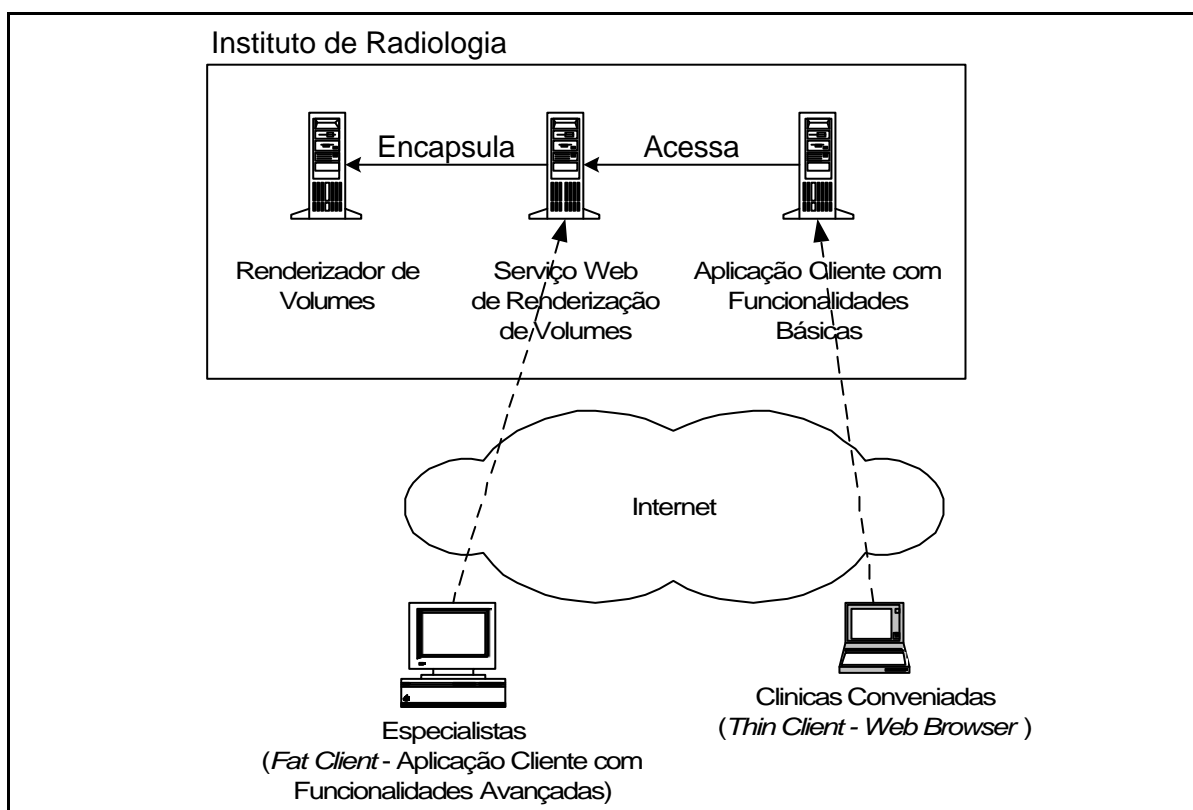


Figura 5.1: Cenário de atuação do protótipo.

O protótipo poderia atuar como uma possível solução às necessidades apresentadas no cenário, encapsulando o renderizador de volumes da instituição e transformando-o em um serviço disponível na Web. Este serviço é baseado em padrões abertos, os quais proporcionam acessibilidade e integração com qualquer aplicação desenvolvida sob estes padrões. Desta forma, pode-se ter aplicações clientes com funcionalidades diferentes acessando o mesmo serviço para atender às diferentes necessidades dos usuários.

## 5.1 PLATAFORMA

Uma vez que a arquitetura utiliza *Web Services* para criação do serviço de renderização, é importante avaliar quais são as possíveis alternativas para implementação desta tecnologia. As duas maiores alternativas são: Microsoft .NET<sup>15</sup> e *Java 2, Enterprise Edition (J2EE)*.

Microsoft .NET é a plataforma de *Web Services* da Microsoft. Essa plataforma é um conjunto de ferramentas de desenvolvimento de software usadas para criar, publicar e utilizar *Web Services*. Conforme publicado no site da MICROSOFT (2002), *“Na nova plataforma .NET a Internet deixará de ser um emaranhado de sites isolados. Ela será um grande conjunto de sistemas interagindo e trabalhando a favor do usuário. De telefones celulares a controles de automóveis, todos os dispositivos irão compartilhar uma plataforma única. A transição do Windows para .NET, embora suave, será tão significativa quanto foi a transição do DOS para a plataforma Windows.”*

Apesar da capacidade de integração e comunicação com outras plataformas por meio de *Web Services*, a .NET é centrada no ambiente Microsoft, ou seja, os

---

<sup>15</sup> .NET pronuncia-se: dot net.

serviços criados com .NET ficam limitados a serem instalados em Sistemas Operacionais Microsoft.

*Java 2, Enterprise Edition* (J2EE) é uma especificação centrada na linguagem Java e baseada em componentes e APIs que compõem um ambiente de execução consistente para desenvolvimento de aplicações corporativas robustas e *Web Services*.

J2EE foi criada pela empresa Sun Microsystems, porém as mudanças na especificação são realizadas por meio de um processo colaborativo da Comunidade Java, da qual fazem parte empresas, tais como Sun, IBM, BEA, Oracle, e HP (COYLE, 2002). Estas empresas adotam a especificação J2EE em seus produtos servidores de aplicação, a qual garante a portabilidade dos componentes e aplicações desenvolvidas nesta plataforma.

O protótipo foi implementado na plataforma J2EE devido à sua maior portabilidade. Isso garante uma abrangência maior de ambientes heterogêneos para realizar a implantação (*deployment*) do serviço. Além das APIs básicas da plataforma J2EE também foram utilizados componentes, tais como AXIS e o servidor de aplicações Jakarta Tomcat, descritos a seguir.

*Apache eXtensible Interaction System* (AXIS) é um *framework*<sup>16</sup> para construção de *Web Services* baseados em Java. O projeto AXIS é desenvolvido no conceito de código aberto e tem a missão principal de dar suporte total às especificações SOAP e WSDL da W3C (APACHE, 2002) (IRANI, 2002).

Jakarta Tomcat é um servidor de aplicações Web do Grupo Apache, o qual é centrado na linguagem Java e desenvolvido no conceito de código aberto (APACHE, 2002). Optou-se pela utilização de AXIS e Jakarta Tomcat no protótipo, por serem baseados em Java, e encaixarem-se perfeitamente com a plataforma J2EE.

---

<sup>16</sup> *Framework* é uma coleção de classes e interfaces inter-relacionadas, as quais são reutilizadas para o desenvolvimento de projetos com funcionalidades semelhantes.

Além destes componentes de software, o protótipo foi desenvolvido em um equipamento PC AMD K6 II 500Mhz, com 256Mb de RAM, e sistema operacional Microsoft Windows 2000 Professional.

## 5.2 IMPLEMENTAÇÃO

Para atender as necessidades abordadas no cenário, o protótipo foi implementado usando os seguintes módulos da arquitetura proposta:

- Servidor de Aplicação Integradora;
- Cliente Internet (B2C).

### 5.2.1 SERVIDOR DE APLICAÇÃO INTEGRADORA

O Servidor de Aplicação Integradora é o módulo principal da arquitetura, o qual faz a ponte de comunicação entre o Renderizador de Volumes e os seus clientes. Este módulo foi implementado conforme o diagrama de classes simplificado apresentado na figura 5.2.

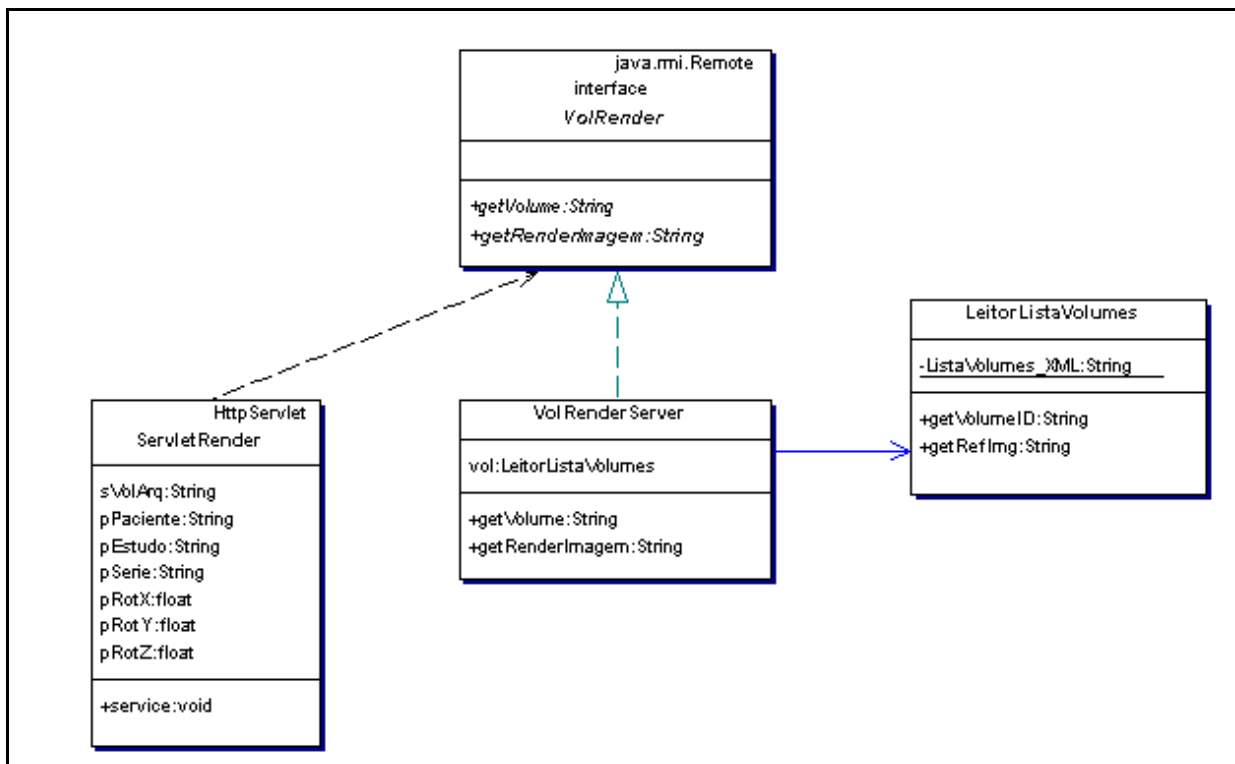


Figura 5.2: Diagrama de Classes do Protótipo.

A Interface `VolRender` define quais são as operações que o serviço disponibiliza e serão implementadas pela classe `VolRenderServer`. As duas operações implementadas foram:

- Volume `getVolume` (Paciente, Estudo, Serie)
  - Parâmetros de Entrada: Recebe o nome do Paciente, Estudo e Série selecionada pelo Cliente.
  - Processamento: Acessa a `ListaVolumes.XML` e realiza uma pesquisa para encontrar o identificador único do volume selecionado.
  - Retorno da operação: Retorna o identificador do volume, ou o valor nulo, caso não encontre o volume selecionado.
- RefImg `getRenderImagem` (Volume)

- Parâmetros de Entrada: Recebe o Identificador único do Volume.
- Processamento: Acessa a ListaVolumes.XML e encontra a referência para a imagem que representa o Volume. Então esta referência é devolvida como resposta para o Cliente.
- Retorno da operação: Retorna a referência da imagem .JPG que representa o volume selecionado.

A classe `LeitorListaVolumes` é responsável por acessar o componente `ListaVolumes.XML`, o qual é um arquivo que contém todas as informações dos volumes disponíveis e seus parâmetros iniciais para renderização. Este componente foi implementado conforme figura 4.10 apresentada no capítulo 4.

`ServletRender` é o servlet de renderização que aceita requisições e gera respostas em páginas HTML para os *Thin Clients*. Ao receber uma requisição de um *Thin Client* pelo método `service`, o `ServletRender` atua como um cliente do serviço de renderização `VolRender`, o qual processa as requisições e as devolve para o `ServletRender`.

## 5.2.2 CLIENTE INTERNET

O cliente implementado neste protótipo é o *Thin Client*, conforme foi apresentado no módulo Cliente Internet B2C (*Business to Consumer*) da arquitetura proposta no capítulo 4. Este cliente pode ser acessado por meio de qualquer *Web browser* disponível na Internet. A Figura 5.3 apresenta um acesso realizado por este cliente ao serviço de renderização implementado no protótipo.

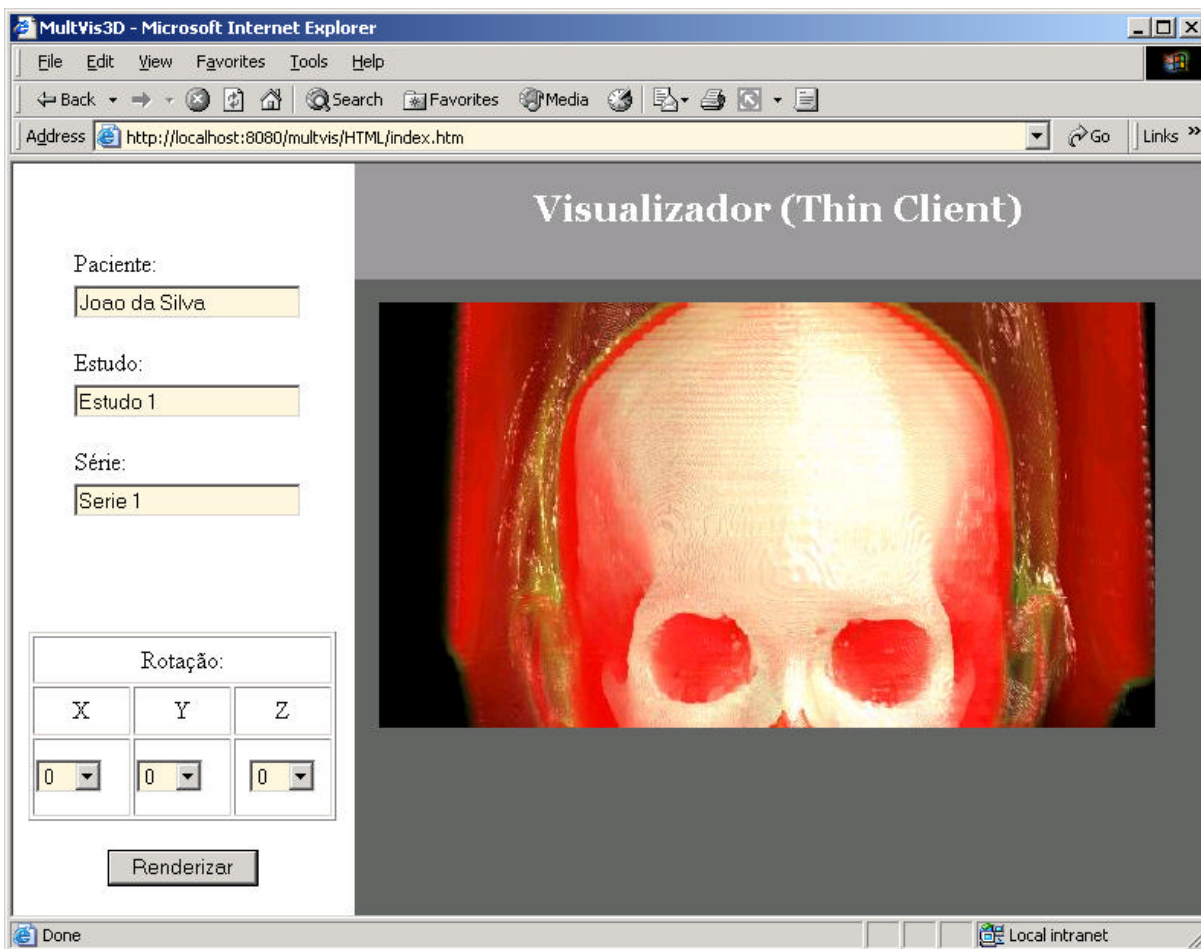


Figura 5.3: Acesso do *Thin Client* ao Serviço de Renderização por meio de um *Web browser*.

O *Thin Client* faz a conexão com o componente Servlet de Renderização e recebe uma interface HTML que permite acessar uma determinada imagem de volume de um paciente. Esta interface pode impor limites nas funcionalidades do renderizador, e neste caso, foi criada para permitir apenas operações básicas de rotação neste volume. A Figura 5.4 apresenta o mesmo volume sendo visualizado com rotação de 45 graus no eixo Y.

Para a geração destas imagens foi utilizada uma aplicação de renderização de Volumes baseada no algoritmo de *ray casting*, a qual foi desenvolvida em Java em outro projeto deste programa. Este renderizador permite outras funcionalidades, tais como alteração de níveis de opacidade e transparência, parâmetros de iluminação, as quais podem ser utilizadas ou não pelas aplicações clientes.

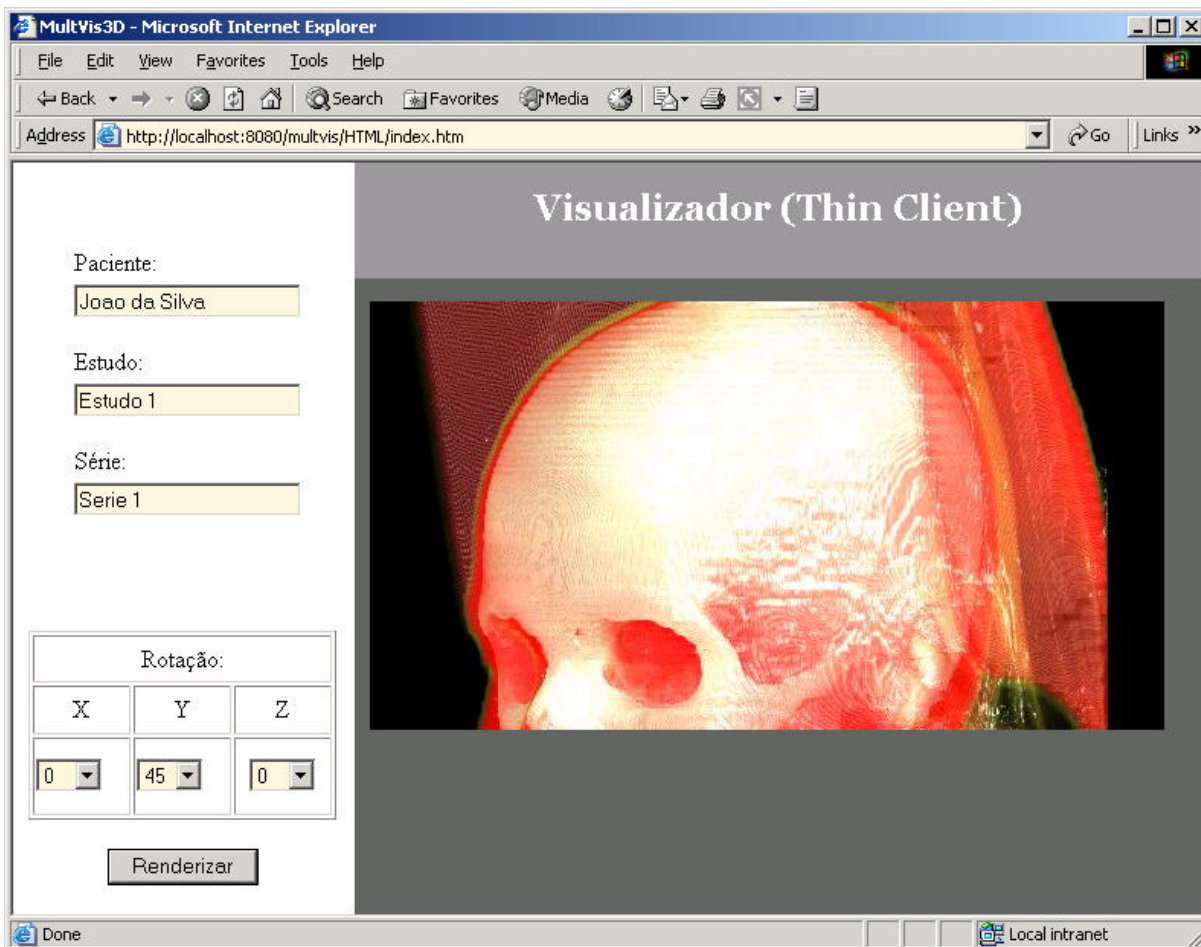


Figura 5.4: Acesso do *Thin Client* ao Serviço de Renderização por meio de um *Web browser* com rotação de 45 graus no eixo Y do volume.

### 5.3 PUBLICAÇÃO E LOCALIZAÇÃO DO SERVIÇO

O módulo UDDI da arquitetura é representado por um serviço UDDI para registro, publicação e localização de *Web Services*. Para registrar o serviço de renderização de volumes criado com o protótipo foi utilizado o *site* <http://uddi.microsoft.com>, o qual é oferecido pela Microsoft. A figura 5.5 apresenta a página de registro do serviço.

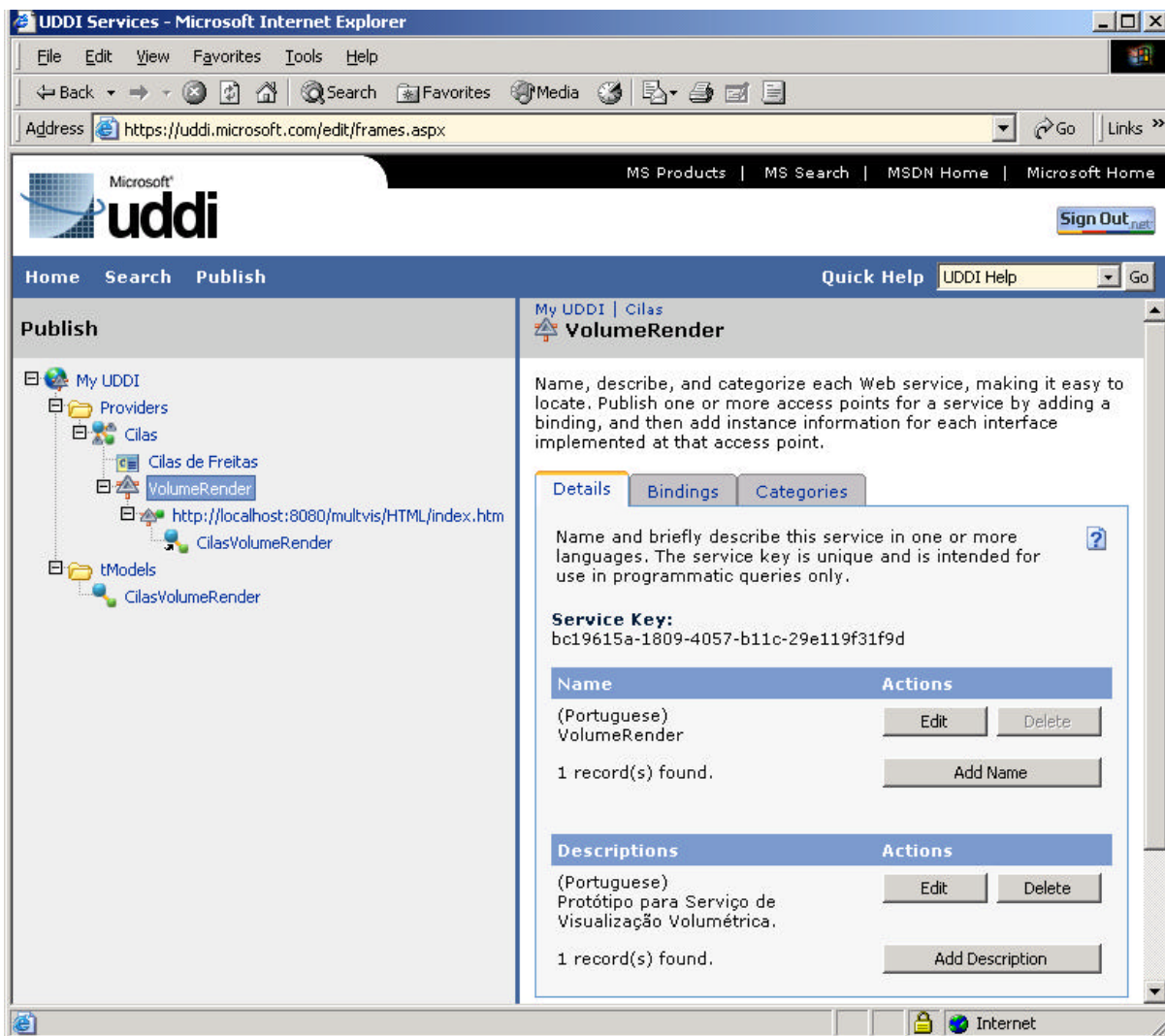


Figura 5.5: Acesso ao serviço UDDI da Microsoft. Disponível em <<http://uddi.microsoft.com>> Acesso em 10/07/2002.

Além de dados do provedor do serviço, tais como nome e formas de contato, também são registradas informações sobre os serviços oferecidos. As informações mais importantes do serviço são nome, descrição, endereço para acesso do serviço e endereço para acesso do WSDL do serviço.

O WSDL é gerado automaticamente pelo AXIS. Para acessá-lo basta utilizar um *Web Browser* e incluir a informação "?WSDL" no final da URL do serviço. Por exemplo, a URL: <http://localhost:8080/axis/services/VolRender?WSDL> gera o WSDL

referente ao protótipo. Este WSDL foi gravado em um arquivo chamado vr.wsdl e está listado nos apêndices.

Após o registro, este serviço fica disponível para ser localizado por qualquer pessoa ou software que acesse um domínio de registro UDDI. Apesar de ter sido registrado no domínio de registro da Microsoft, este registro é replicado para todos os domínios UDDI conveniados, tais como IBM e SAP.

## 6 CONCLUSÕES

Demonstrado o uso do conceito de uma Arquitetura baseada em padrões abertos para visualização científica via Internet, verificou-se os benefícios de integração e acessibilidade que esta arquitetura traz para os sistemas de visualização volumétrica remota aplicados à área médica.

Sistemas baseados na arquitetura proposta podem ser desenvolvidos em linguagens e plataformas heterogêneas utilizando mensagens no formato XML. Estas mensagens podem ser transferidas por meio de protocolos padrões para tráfego de informações na Web, tais como HTTP, FTP, SMTP.

Assim como o padrão DICOM oferece mecanismos para acesso e transferência de imagens médicas em rede, a arquitetura proposta oferece serviços de renderização volumétrica destas imagens via Internet. Estes serviços ficam disponíveis na Internet para serem usados diretamente, ou para serem usados como componentes reutilizáveis para desenvolvimento de novas aplicações.

Por meio da utilização de padrões abertos na arquitetura proposta, tornam-se possíveis soluções para problemas de integração entre instituições provedoras de serviços de renderização de volumes e seus clientes, tais como institutos de radiologia e clínicas médicas, ou mesmo entre os próprios institutos de radiologia, ou em universidades usando aplicações para treinamento à distância.

Além disso, essa arquitetura pode ser vista como um passo inicial para construção de sistemas baseados em padrões abertos para visualização colaborativa, uma vez que a integração e a acessibilidade são características essenciais para estes sistemas.

## 6.1 TRABALHOS FUTUROS

Em sistemas voltados para a medicina, a segurança de acesso às informações é um aspecto bastante crítico, porém o foco deste trabalho foi direcionado para integração e acessibilidade, não levando em conta a questão de segurança, que é também um forte candidato para pesquisas futuras.

Futuros desenvolvimentos devem abordar a questão do controle de acesso, ou seja, quais as funções que podem ser liberadas para diferentes tipos de cliente. Por exemplo: Uma Clínica Médica poderia utilizar as funções de rotação, iluminação e outras, porém, não poderia alterar os parâmetros dos materiais.

Este trabalho oferece a base para a construção de novos serviços distribuídos de visualização baseados em padrões abertos. Por meio da implementação de parte da arquitetura proposta, o protótipo demonstra a viabilidade da construção destes serviços. Porém não foram realizados testes de desempenho e comparação com resultados gerados pelas arquiteturas convencionais. Estes quesitos devem ser levados em consideração em trabalhos futuros nesta plataforma.

## 7 REFERÊNCIAS BIBLIOGRÁFICAS

APACHE GROUP. **The Apache XML Project**. Disponível em: <<http://xml.apache.org>> Acesso em: 20 abr. 2002.

BOX, D. et al. **Simple Object Access Protocol (SOAP) 1.1**. World Wide Web Consortium (W3C), 2000. Disponível em: <<http://www.w3.org/TR/SOAP>> Acesso em 10 mar. 2002.

BRODLIE, K. **Visualization Over The World Wide Web**. Information Visualization. IEEE Conference, 1997.

BRODLIE, K.; WOOD, J. **Volume Graphics and the Internet**. M Chen, A E Kaufman and R Yagel (editors), Volume Graphics, pp317-331. Springer Verlag, 2000.

COYLE, F. P. **XML, Web Services, and the Data Revolution**. Addison-Wesley Information Technology Series, 2002.

CURBERA, F.; NAGY, W. A.; WEERAWARANA, S. **Web Services: Why and How**. IBM T.J. Watson Research Center, 2001.

DAPI - Diagnóstico Avançado Por Imagem. Disponível em: <<http://www.dapi.com.br>> Acesso em: 17 jan. 2002.

DICOM. **DICOM Standard**. Disponível em: <<http://medical.nema.org/dicom/2001.html>>. Acesso em: 14 jan. 2002.

DREBIN, R. A.; CARPENTER, L.; HANRAHAN, P. **Volume Rendering**. Computer Graphics 22, p65-74. Proceedings of SIGGRAPH'88. 1988.

ELVINS, T. T. **Volume Visualization in a Collaborative Computing Environment**. Computers&Graphics, Oxford, Vol. 20, No.2, p. 219-222, 1996.

GAO, Z. **Extending Scientific Visualization into the World Wide Web**. Master Thesis of Computer Science. Dalhousie University - DALTECH, 1998.

GEUS, K. **Visualização 3D em Planejamento de Radioterapia Usando Rendering de Volumes por Ray Casting**. Anais do SIBGRAPI VI, 1993.

GEUS, K. **Visualization in Radiotherapy Planning**. PhD. Thesis, University of Sheffiedd, 1992.

GRAHAM, S. et al. **Building Web Services with Java: Making Sense of XML, SOAP, WSDL and UDDI**. SAMS Publishing, 2001.

HABER, R. B.; MCNABB, D. A. **Visualization idioms: A conceptual model for scientific visualization systems**. Visualization in Scientific Computing, pp74-93, IEEE, 1990.

IRANI, R.; BASHA, S. J. **AXIS: The next generation of Java SOAP**. Wrox Press Ltd., 2002.

ISO/IEC 14772-1. Information technology – Computer Graphics and Image Processing – **Virtual Reality Modelling Language (VRML)**, 1997. Disponível em: <<http://www.bsi.org.uk/sc24/sc24/standard.htm>>. Acesso em 15 mar. 2002.

JOHN, N. K. et al. **Bringing 3D to Teleradiology**. IEEE, 2000.

JURIC, B. M. et al. **Professional J2EE EAI**. Wrox Press Ltd., 2001.

LEVOY, M. **Display of Surfaces from Volume Data**. IEEE Computer Graphics & Applications, p29-37, 1988.

LICHTENBELT, B.; CRANE, R.; NAQVI, S. **Introduction to Volume Rendering**. Hawlett-Packard Professional Books, 1998.

LORENSEN, W. E.; CLINE, H. E. **Marching Cubes: A High Resolution 3D Surface Construction Algorithm**. Computer Graphics, 1987.

MANSSOUR, I. H.; FREITAS, C. M. D. S. **Collaborative Visualization In Medicine**. WSCG 2000 – The 8th International Conference in Central Europe on Computer Graphics, Plzen, Czech Republic, 2000.

MATHIESEN, F. K. **WEB technology — the future of teleradiology?** Computer Methods and Programs in Biomedicine 66, pp87–90, Elsevier Science Ireland Ltd., 2001.

MCCORMICK, B.; DEFANTI, T. & BROWN, M. **Visualization in Scientific Computing**. Computer Graphics, Vol.21, No.6, 1987.

MICROSOFT **.NET**. Disponível em: <http://www.microsoft.com/brasil/net/default.asp>. Acesso em: 25/06/2002.

OLIVEIRA JR., P. P. M. **Exames Virtuais Utilizando um Algoritmo de Ray Casting Acelerado**. Dissertação de Mestrado, TeCGraf, PUC-Rio, 1999.

OMG – OBJECT MANAGEMENT GROUP. **Common Object Request Broker Architecture (CORBA), v2.4.2 Specification**. OMG Document, 2001. Disponível em: <<http://www.omg.org>> Acesso em: 25 mar. 2002.

ORFALI, R.; HARKEY, D.; EDWARDS, J. **Instant CORBA**. John Wiley & Sons, 1997.

PAIVA, A. C.; SEIXAS, R. B.; GATTASS, M. **Introdução à Visualização Volumétrica**. PUC-Rio, 1999.

PHONG, B. T. **Illumination for computer generated pictures**. Communications of the ACM, 18, p311-317, 1975.

PURGATHOFER W.; LÖFFELMANN H. **Selected New Trends in Scientific Visualization**. Institute of Computer Graphics, Vienna University of Technology, 1997.

SCHROEDER, W.; MARTIN, K.; LORENSEN, W.; **The Visualization Toolkit**. 2nd. ed. Prentice Hall PTR, 1997.

SOWEK, C. A. **Web Services - a nova "onda"**. CELEPAR - Bate Byte 120, 2002.

SYSTINET Corp. **Introduction to Web Services**. SYSTINET White Paper, 2001. Disponível em: <<http://www.systinet.com>> Acesso em: 20 mar. 2002.

TSOI, K.; GRÖLLER E. **Adaptive visualization over the Internet**. Disponível em: <<http://citeseer.nj.nec.com/379785.html>>. Acesso em: 10 abr. 2002.

UDDI Project. **UDDI Technical White Paper**. 2000. Disponível em: <<http://www.uddi.org>> Acesso em: 15 abr. 2002.

UPSON, C., et al. **The Application Visualization System: A computational environment for scientific visualization**, IEEE Computer Graphics and Applications, Vol 9, número 4, pp30-42, 1989.

WATT, A. **3D Computer Graphics**. 2.ed. Addison-Wesley Publishing Company Inc., p.314-329, 1993.

WITTENBRINK, C. M. et al. **PermWeb: Remote parallel and distributed volume visualization**. Technical Report HPL-97-34, HP Labs, 1997.

WOOD, J.; BRODLIE, K.; WRIGHT, H. **CSCV- Computer Supported Collaborative Visualization**. Proceedings of BCS, Displays Group International Conference on Visualization and Modelling, 1995.

WOOD, J.; BRODLIE, K.; WRIGHT, H. **Visualization over the World Wide Web and its application to environmental data**. Visualization '96, 1996.

## 8 APÊNDICE

A seguir é listado o arquivo WSDL (vr.wSDL) do serviço de renderização de volumes criado para o protótipo e gerado automaticamente pelo AXIS, o qual pode ser acessado por meio do link: <http://localhost:8080/axis/services/VolRender?WSDL>.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
targetNamespace="http://localhost:8080/axis/services/VolRender"
xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:impl="http://localhost:8080/axis/services/VolRender-impl"
xmlns:intf="http://localhost:8080/axis/services/VolRender"
xmlns:tns1="http://description.axis.apache.org"
xmlns:tns2="http://lang.java" xmlns:tns3="http://encoding.axis.apache.org"
xmlns:tns4="http://reflect.lang.java"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"><types><schema
targetNamespace="http://schemas.xmlsoap.org/soap/encoding/"
xmlns="http://www.w3.org/2001/XMLSchema"><element name="Array"
nillable="true" type="SOAP-ENC:Array"/></schema><schema
targetNamespace="http://description.axis.apache.org"
xmlns="http://www.w3.org/2001/XMLSchema"><complexType
name="OperationDesc"><sequence><element name="parent" nillable="true"
type="tns1:ServiceDesc"/><element name="name" nillable="true"
type="xsd:string"/><element name="elementQName" nillable="true"
type="xsd:QName"/><element name="returnQName" nillable="true"
type="xsd:QName"/><element name="returnType" nillable="true"
type="xsd:QName"/><element name="returnClass" nillable="true"
type="tns2:Class"/><element name="method" nillable="true"
type="tns4:Method"/><element name="soapAction" nillable="true"
type="xsd:string"/><element name="Style"
type="xsd:int"/></sequence></complexType><complexType
name="ServiceDesc"><sequence><element name="STYLE_RPC"
type="xsd:int"/><element name="STYLE_DOCUMENT" type="xsd:int"/><element
name="STYLE_WRAPPED" type="xsd:int"/><element name="STYLE_MESSAGE"
type="xsd:int"/><element name="name" nillable="true"
type="xsd:string"/><element name="allowedMethods" nillable="true"
type="SOAP-ENC:Array"/><element name="disallowedMethods" nillable="true"
type="SOAP-ENC:Array"/><element name="style" type="xsd:int"/><element
name="implClass" nillable="true" type="tns2:Class"/><element
name="stopClasses" nillable="true" type="SOAP-ENC:Array"/><element
name="TypeMapping" nillable="true" type="tns3:TypeMapping"/><element
name="WSDLFile" nillable="true"
type="xsd:string"/></sequence></complexType><element name="OperationDesc"
nillable="true" type="tns1:OperationDesc"/></schema><schema
targetNamespace="http://lang.java"
xmlns="http://www.w3.org/2001/XMLSchema"><complexType
name="Class"><sequence/></complexType></schema><schema
targetNamespace="http://encoding.axis.apache.org"
xmlns="http://www.w3.org/2001/XMLSchema"><complexType abstract="true"
```

```

name="TypeMapping"><sequence><element name="Delegate" nillable="true"
type="tns3:TypeMapping" /></sequence></complexType></schema><schema
targetNamespace="http://reflect.lang.java"
xmlns="http://www.w3.org/2001/XMLSchema"><complexType
name="Method"><complexContent><extension
base="tns4:AccessibleObject"><sequence/></extension></complexContent></comp
lexType><complexType name="AccessibleObject"><sequence><element
name="Accessible"
type="xsd:boolean" /></sequence></complexType></schema></types>
  <wsdl:message name="getVolumeRequest">
    <wsdl:part name="in0" type="xsd:string" />
    <wsdl:part name="in1" type="xsd:string" />
    <wsdl:part name="in2" type="xsd:string" />
  </wsdl:message>
  <wsdl:message name="getOperationDescByNameRequest">
    <wsdl:part name="methodName" type="xsd:string" />
  </wsdl:message>
  <wsdl:message name="getOperationDescsRequest">
  </wsdl:message>
  <wsdl:message name="getRenderImagemResponse">
    <wsdl:part name="return" type="xsd:string" />
  </wsdl:message>
  <wsdl:message name="getVolumeResponse">
    <wsdl:part name="return" type="xsd:string" />
  </wsdl:message>
  <wsdl:message name="getRenderImagemRequest">
    <wsdl:part name="in0" type="xsd:string" />
    <wsdl:part name="in1" type="xsd:float" />
    <wsdl:part name="in2" type="xsd:float" />
    <wsdl:part name="in3" type="xsd:float" />
  </wsdl:message>
  <wsdl:message name="getOperationDescByNameResponse">
    <wsdl:part name="return" type="tns1:OperationDesc" />
  </wsdl:message>
  <wsdl:message name="getOperationDescsResponse">
    <wsdl:part name="return" type="SOAP-ENC:Array" />
  </wsdl:message>
  <wsdl:portType name="VolRender">
    <wsdl:operation name="getRenderImagem" parameterOrder="in0 in1 in2
in3">
      <wsdl:input message="intf:getRenderImagemRequest" />
      <wsdl:output message="intf:getRenderImagemResponse" />
    </wsdl:operation>
    <wsdl:operation name="getVolume" parameterOrder="in0 in1 in2">
      <wsdl:input message="intf:getVolumeRequest" />
      <wsdl:output message="intf:getVolumeResponse" />
    </wsdl:operation>
    <wsdl:operation name="getOperationDescs">
      <wsdl:input message="intf:getOperationDescsRequest" />
      <wsdl:output message="intf:getOperationDescsResponse" />
    </wsdl:operation>
    <wsdl:operation name="getOperationDescByName"
parameterOrder="methodName">
      <wsdl:input message="intf:getOperationDescByNameRequest" />
      <wsdl:output message="intf:getOperationDescByNameResponse" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="VolRenderSoapBinding" type="intf:VolRender">
    <wsdlsoap:binding style="rpc"
transport="http://schemas.xmlsoap.org/soap/http" />

```

```

    <wsdl:operation name="getRenderImagem">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input>
        <wsdlsoap:body
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="getRenderImagem" use="encoded"/>
        </wsdl:input>
      <wsdl:output>
        <wsdlsoap:body
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="return" use="encoded"/>
        </wsdl:output>
      </wsdl:operation>
    <wsdl:operation name="getVolume">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input>
        <wsdlsoap:body
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="getVolume" use="encoded"/>
        </wsdl:input>
      <wsdl:output>
        <wsdlsoap:body
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="return" use="encoded"/>
        </wsdl:output>
      </wsdl:operation>
    <wsdl:operation name="getOperationDescs">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input>
        <wsdlsoap:body
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="getOperationDescs" use="encoded"/>
        </wsdl:input>
      <wsdl:output>
        <wsdlsoap:body
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://localhost:8080/axis/services/VolRender" use="encoded"/>
        </wsdl:output>
      </wsdl:operation>
    <wsdl:operation name="getOperationDescByName">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input>
        <wsdlsoap:body
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="getOperationDescByName" use="encoded"/>
        </wsdl:input>
      <wsdl:output>
        <wsdlsoap:body
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://localhost:8080/axis/services/VolRender" use="encoded"/>
        </wsdl:output>
      </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="VolRenderService">
    <wsdl:port binding="intf:VolRenderSoapBinding" name="VolRender">
      <wsdlsoap:address
location="http://localhost:8080/axis/services/VolRender"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>

```