

FUNDAÇÃO DE ENSINO EURÍPIDES SOARES DA ROCHA
CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA – UNIVEM
PROGRAMA DE MESTRADO EM CIÊNCIA DA COMPUTAÇÃO

ANA CLÁUDIA MELO TIESSI GOMES DE OLIVEIRA

***ViMeT* – PROJETO E IMPLEMENTAÇÃO DE UM *FRAMEWORK* PARA
APLICAÇÕES DE TREINAMENTO MÉDICO USANDO REALIDADE
VIRTUAL**

MARÍLIA
2007

ANA CLÁUDIA MELO TIESSI GOMES DE OLIVEIRA

***ViMeT* – PROJETO E IMPLEMENTAÇÃO DE UM *FRAMEWORK* PARA
APLICAÇÕES DE TREINAMENTO MÉDICO USANDO REALIDADE
VIRTUAL**

Dissertação apresentada ao Programa de Mestrado do Centro Universitário Eurípides de Marília, mantido pela Fundação de Ensino Eurípides Soares da Rocha, para obtenção do Título de Mestre em Ciência da Computação (Área de Concentração: Realidade Virtual).

Orientadora:
Prof.^a Dr.^a Fátima L.S. Nunes Marques

MARÍLIA
2007

OLIVEIRA, Ana Cláudia M. T. G. de

ViMeT – Projeto E Implementação de um *Framework* Para Aplicações De Treinamento Médico Usando Realidade Virtual / Ana Cláudia Melo Tiessi Gomes de Oliveira; Orientadora: Fátima L. dos Santos Nunes Marques.

Marília, SP: [s.n.], 2007.

137 f.

Dissertação (Mestrado em Ciência da Computação) - Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha.

1. Biópsia 2. *Framework* 3. Java 4. Java3D 5. Realidade Virtual
6. Treinamento Médico

CDD: 006 6784

Ana Cláudia Melo Tiessi Gomes de Oliveira

ViMeT - Projeto e Implementação de um Framework para Aplicações de Treinamento Médico Usando Realidade Virtual

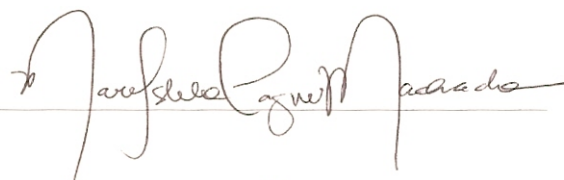
Banca examinadora da dissertação apresentada ao Programa de Mestrado da UNIVEM / FEESR, para obtenção do Título de Mestre em Ciência da Computação.

A Comissão Julgadora:

Profa. Dra Fátima L. S. Nunes Marques



Profa. Dra Maria Istela Cagnin Machado



Prof. Dr. Romero Tori



Marília, 03 de abril de 2007.

Dedico este trabalho às pessoas mais importantes da minha vida:

meus amados pais, pelo amor que sempre me deram e por entenderem a minha ausência e meu marido, pelo incentivo, companheirismo e amor, sem o apoio de vocês tudo teria sido muito mais difícil.

AGRADECIMENTOS

Quando iniciei o mestrado pedi a Deus paciência, sabedoria, inteligência, determinação, dedicação, disciplina, força de vontade e entusiasmo. Ele não me deu tudo isso de uma vez. Porém, na Sua infinita sabedoria colocou na minha vida pessoas que me ensinaram e me mostraram como adquirir tudo o que havia pedido e muito mais. E é a essas pessoas que presto os mais profundos e sinceros agradecimentos, pois todos citados aqui contribuíram de alguma forma na elaboração deste trabalho.

Aos meus pais, Pico e Dulce, agradeço à dedicação, orações e amor que sempre me dedicaram. Amo vocês.

Ao meu marido, José Carlos, que me motivou e entendeu minha ausência. Você é o maior exemplo de dedicação, determinação e disciplina. Obrigada pelo companheirismo e amor.

A Prof.^a Dr.^a Fátima L. S. Nunes Marques que me deu exemplo de sabedoria, determinação e entusiasmo. Obrigada pelas horas de orientação e conselhos. Agradeço-lhe por te confiado em mim e aceitado me orientar nesse projeto e por ser um exemplo de honestidade e entusiasmo.

Ao Prof. Dr. José Remo Brega pela iniciação a Realidade Virtual. Obrigada pelo incentivo.

A Prof.^a Dr.^a Maria Istela Cagnin, pela ajuda com a conceituação de framework, cookbook e tudo mais. Agradeço pelo carinho, ajuda e paciência.

Ao corpo docente do Programa de Mestrado em Ciência de Computação -UNIVEM

A Marcinha, secretária do mestrado e minha amiga, obrigada pela doçura e paciência que sempre me atendeu e tirou todas as minhas dúvidas.

A querida Ana Paula Medeiros Lima, obrigada por vibrar comigo em cada avanço da implementação e aceitação de artigos. E, principalmente, por e me motivar e levantar o meu astral quando tudo dava errado.

A Bárbara Monteiro, exemplo de maturidade, obrigada por me ajudar, pelo carinho e amizade.

A Larissa Pavarini pela amizade e carinho. Obrigada por sempre me ajudar.

A Cláudia Mendonça pelas palavras de carinhos, incentivo e serenidade.

Aos meus amigos da turma de 2005 Richard, Fabrício, Fernando, Lima, Marliane, Franciene e Vânia.

Ao meu amigo Sérgio. Obrigada por sempre me ajudar e por dividir todas as angústias dos prazos e dos artigos não aceitos.

Aos integrantes do LApIS (Leonardo, Bárbara, Ana Paula, Sérgio, Montanha, Danilo, Adriano e Kera).

Aos meus amigos Rodolfo Chiaramonte e Paulo Nardi. Vocês são exemplos de sabedoria e de humildade. Aprendi muito com os dois.

Ao CNPq pelo apoio financeiro

OLIVEIRA, Ana Cláudia, M. T. G. de. *ViMeT – Projeto E Implementação de um Framework Para Aplicações De Treinamento Médico Usando Realidade Virtual*. 2007. 137 f. Dissertação (Mestrado em Ciência da Computação) – Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2007.

RESUMO

O conceito de *framework* tem sido aplicado em computação com o objetivo de construir aplicações para as mais diversas áreas de conhecimento, pois garante um desenvolvimento de software de qualidade e sem grandes esforços de implementação. Este trabalho apresenta o processo de desenvolvimento e a implementação de um *framework* orientado a objetos para treinamento médico que utiliza técnicas de Realidade Virtual, denominado *ViMeT (Virtual Medical Training)*. O objetivo é construir um *framework*, a partir da reutilização de códigos de aplicações anteriormente desenvolvidas, que implementam funcionalidades de detecção de colisão, deformação e estereoscopia. A fim de propiciar, dentro do contexto de RV aplicada à Medicina, uma forma otimizada de construir ferramentas para simulação de exames de punção, o *ViMeT* foi desenvolvido de forma que garanta a possibilidade de inserção de novas técnicas para cada funcionalidade e a combinação entre as várias técnicas utilizadas para construir novas aplicações.

Palavras-chave: Biópsia, *Framework*, Java, Java3D, Realidade Virtual, Treinamento Médico.

OLIVEIRA, Ana Cláudia, M. T. G. de. *ViMeT – Projeto E Implementação de um Framework Para Aplicações De Treinamento Médico Usando Realidade Virtual*. 2007. 137 f. Dissertação (Mestrado em Ciência da Computação) – Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2007.

ABSTRACT

In computing studies the concept of *framework* has been applied in order to build applications meant for a myriad of knowledge fields, since it guarantees a quality software development with less implementation effort. Thus, the present work is intended for presenting the process of development and the implementation of *ViMeT* (Virtual Medical Training), an object-oriented *framework* for medical training using Virtual Reality techniques. The purpose is to build a *framework* from the adjustment of codes of applications previously developed which implements functionalities of collision detection, deformation and stereoscopy. In order to provide, in the context of VR applied to Medicine, an optimized manner of building tools for the biopsy exam simulation, *ViMeT* was developed to guarantee the possibility of introducing new techniques for each application. It is also part of *ViMeT* an instantiation tool (*Wizard*) which helps the developer to create new applications instantaneously.

Keywords: Biopsy, *Framework*, Java, Java3D, Medical Training, Virtual Reality.

LISTA DE ILUSTRAÇÕES

Figura 1 - Modelos de Dispositivos: (a) luva 5DT Data Glove, (b) dispositivos com seis graus de liberdade, (c) HMD - 5DT HMD 800.....	20
Figura 2 - (a) Plataforma de cirurgia laparoscópica, (b) Interface para laparoscopia virtual..	21
Figura 3 - (a) Equipamento para simulação de cirurgia endovascular	21
Figura 4 - Equipamento PHANTON	22
Figura 5 - Possíveis formas de visualização do simulador de anestesia.....	24
Figura 6 - Interface de um treinamento de sutura sendo executado no simulador	24
Figura 7 - Simulador de Histeroscopia.....	25
Figura 8 - Ambiente Virtual do Simulador de Histeroscopia.....	26
Figura 9 - (a) Simulador EyeSi , (b) Cenário do EyeSI.....	26
Figura 10 - (a) Sistema <i>EYESIY</i> TM , (b) Modelo Mass-Springs usado para romper a membrana	27
Figura 11 - (a) Modelo tridimensional da região pélvica, (b) Utilização do PHANTON	28
Figura 12 - Estudo do coração no <i>VirtWall</i>	29
Figura 13 - Protótipo para punção de mama	30
Figura 14 - Tela para passagem de parâmetros e caixa de diálogo	31
Figura 15 - Imagens geradas pelo sistema.....	31
Figura 16 - Interface do Atlas.....	32
Figura 17 - Imagens geradas a partir das opções selecionadas pelo usuário.....	32
Figura 18 - Principais diferenças de uma Biblioteca e um <i>framework</i>	36
Figura 19 - Desenvolvimento tradicional de uma aplicação orientada a objetos	39
Figura 20 - Desenvolvimento baseado em <i>frameworks</i>	39
Figura 21 - Processo de desenvolvimento dirigido por Hot spots	42
Figura 22 - Atividades para a construção de um <i>framework</i>	43
Figura 23 - Esquema dos componentes do IVORY.	52
Figura 24 – O IVORY <i>frontend</i> sendo executado em uma máquina com Windows NT4.0 com browser Netscape.....	53
Figura 25 – (a) Exemplo de aplicação utilizando dispositivos de <i>force-feedback</i> , (b) Aplicação sem o dispositivo de <i>force-feedback</i>	54
Figura 26- <i>Kernel</i> do <i>bash</i> e seus <i>plugins</i>	54

Figura 27- Diagrama de instalação.....	55
Figura 28 - Uma árvore do ViRAL com os sistemas de padrões.....	56
Figura 29 - Diagrama de classes aplicação de Laparoscopia Virtual.....	58
Figura 30 - Interface de Laparoscopia Virtual.....	58
Figura 31 - Exemplo de um grafo de cena.....	63
Figura 32 - Fluxograma das atividades do <i>ViMeT</i>	64
Figura 33 - Arquitetura do <i>ViMeT</i>	67
Figura 34 – Diagrama de classes do projeto detecção de colisão.....	69
Figura 35 - Indicação de colisão da agulha e da mama.....	69
Figura 36- Diagrama de classes do projeto deformação.....	70
Figura 37 – Teste com variação de força.....	71
Figura 38 - Teste variando a quantidade de vértices dos objetos modelados.....	71
Figura 39 - Diagrama de classes projeto estereoscopia.....	72
Figura 40 - (a) Representação da “Visão do Olho Esquerdo” com a camada vermelha, (b) Representação da “Visão do Olho Direito” com a camada azul.....	73
Figura 41 – Geração do Anaglifo.....	73
Figura 42 - Primeiro diagrama de classes proposto para o <i>ViMeT</i>	75
Figura 43 - Diagrama de Classes completo.....	76
Figura 44 - Diagrama de classes com os métodos implementados do <i>ViMeT</i>	77
Figura 45 - Fases da implementação.....	78
Figura 46 -Trecho do código com o método super da classe Environment recebendo o parâmetro da estereoscopia.....	81
Figura 47 - Grafo de Cena gerado pelo <i>ViMeT</i> (aplicação sem estereoscopia).....	82
Figura 48 - Grafo de Cena gerado pelo <i>ViMeT</i> (aplicação com Estereoscopia).....	82
Figura 49 - Esquema geral do BD.....	88
Figura 50 - Modelo relacional do BD da <i>Wizard</i>	89
Figura 51 - Diagrama de Classes do BD.....	90
Figura 52 – Protótipo da interface do <i>Wizard</i>	91
Figura 53 - (a) Interface de definição dos parâmetros para o órgão, (b) Interface para a definição dos parâmetros da funcionalidade de deformação.....	92
Figura 54 – Nova Interface da <i>Wizard</i>	92
Figura 55 - Trecho de código-fonte gerado pela <i>Wizard</i>	94
Figura 56 – Aplicação gerada por meio da <i>Wizard</i>	94
Figura 57 - Objetos modelados que simulam instrumentos médicos.....	96

Figura 58 - Objetos modelados que simulam órgãos humanos	97
Figura 59 - Trecho do código do estudo de caso 1	98
Figura 60 - Aplicação do estudo de caso 1	98
Figura 61 - Aplicação do estudo de caso 2	99
Figura 62 - Aplicação do estudo de caso 3	100
Figura 63 - Aplicação do estudo de caso 4	101
Figura 64 - Primeiro resultado do Estudo de Caso 5	102
Figura 65 - Aplicação do estudo de caso 5	102
Figura 66- Tela de manutenção com o estudo de caso 5	103
Figura 67 – Resultado do Estado de caso 5 – Recorte.....	103
Figura 68 - Aplicação do estudo de caso 6	104

LISTAS DE QUADROS

Quadro 1 - Comparativo entre as atividades de Mattsson (1996) e os métodos de Bosch <i>et al.</i> (1999)	47
Quadro 2 - Comparativos dos <i>frameworks</i> estudados	60
Quadro 3 – Legenda do Grafo de Cena	63
Quadro 4 - Classes reutilizadas no <i>ViMeT</i>	74
Quadro 5 - Descrição dos componentes do grafo de cena do <i>ViMeT</i>	83
Quadro 6 - Mapeamento entre os nós do Grafo de Cena com as classes e métodos do <i>ViMeT</i> 84	
Quadro 7 - Descrição das finalidades das classes da <i>Wizard</i>	90
Quadro 8 - Configuração dos estudos de casos	97

LISTA DE ABREVIATURAS

2D: Bidimensional
3D: Tridimensional
6DOF: *Six Degrees Of Freedom*
ACTP: Angioplastia Coronária Transluminal Percutânea
API: *Application Programming Interface*
AV: Ambiente Virtual
BD: Banco de Dados
CAD: *Computer Aided Design*
CORBA: *Common Object Request Broker Architecture*
GUI: *Graphical User Interface*
HMD : *Head Mounted Display*
IVML: *Information Visualization Modeling Language*
LApIS: Laboratório de Aplicações de Informática em Saúde
MIDI : *Musical Instrument Digital Interface*
MVC: *Model-View-Controller*
OMG: *Object Management Group*
ORB: *Object Request Broker*
PHANTOM: *Personal Haptic Interface Mechanism*
RV: Realidade Virtual
SGBD: Sistema Gerenciador de Banco de Dados
UML: *Unified Modeling Language*
VLI: *Virtual Laparoscopic Interface*
VRML: *Virtual Reality Modeling Language*
VRPN: *Virtual Reality Peripheral Network*
WIMP: *Windows, Icons, Menus and Pointing Device*

SUMÁRIO

1 INTRODUÇÃO	15
1.1 OBJETIVO DO TRABALHO	16
1.2 JUSTIFICATIVA DO TRABALHO	16
1.3 DISPOSIÇÃO DO TRABALHO	17
2 REALIDADE VIRTUAL NA MEDICINA	18
2.1 CONCEITOS GERAIS DE REALIDADE VIRTUAL	18
2.1.1 DISPOSITIVOS MAIS UTILIZADOS	19
2.2 APLICAÇÕES DE REALIDADE VIRTUAL NA MEDICINA	22
2.2.1 APLICAÇÕES DESENVOLVIDAS EM OUTROS PAÍSES	23
2.2.2 Pesquisas no Brasil.....	27
2.2.3 Pesquisa no LApIS.....	30
2.3 CONSIDERAÇÕES FINAIS.....	33
3 FRAMEWORKS ORIENTADOS A OBJETOS.....	34
3.1 CONCEITUAÇÃO	34
3.2 CLASSIFICAÇÕES DE FRAMEWORKS	36
3.3 DESENVOLVIMENTO DE UM FRAMEWORK	38
3.3.1 Metodologia proposta por Johnson (1993).....	39
3.3.2 Metodologia proposta por Roberts e Johnson (1996).....	40
3.3.3 Metodologia proposta por Bosch et al. (1999)	41
3.3.4 Metodologia proposta por Pree (1999).....	42
3.3.5 Metodologia proposta por Schmid (1997, 1999)	43
3.4 UTILIZAÇÃO DE UM FRAMEWORK	44
3.5 VANTAGENS E DESVANTAGENS DE UM FRAMEWORK	47
3.6 CONSIDERAÇÕES FINAIS.....	49
4 FRAMEWORKS DE REALIDADE VIRTUAL.....	50
4.1 AVANGO.....	50
4.2 SSVE – SHARED SIMPLE VIRTUAL ENVIRONMENT	51
4.3 IVORY	52
4.4 BASHO - VIRTUAL ENVIRONMENT FRAMEWORK	54
4.5 ViRAL (VIRTUAL REALITY ABSTRACTION LAYER).....	55
4.6 VPAT (VIRTUAL PATIENTS).....	57
4.7 COMPARAÇÃO ENTRE OS FRAMEWORKS DE RV	59
4.8 CONSIDERAÇÕES FINAIS.....	60
5 DESENVOLVIMENTO DO VIMET	61
5.1 CONTEXTUALIZAÇÃO	61
5.2 TECNOLOGIAS UTILIZADAS	61
5.3 METODOLOGIA EMPREGADA	64
5.3.1 Análise de Domínio.....	65
5.3.2 Projeto arquitetural	66
5.3.3 Projeto do framework	67
5.3.3.1 Detecção de Colisão com precisão	68
5.3.3.2 Deformação	69
5.3.3.3 Estereoscopia.....	72
5.3.3.4 Construção do Diagrama de Classes do ViMeT	74
5.3.4 Implementação do framework.....	76
5.3.5 Teste do framework.....	86
5.3.6 Documentação	86
5.4 CONSIDERAÇÕES FINAIS.....	87
6 WIZARD E BANCO DE DADOS.....	88

6.2 PROJETO DO BANCO DE DADOS.....	88
6.2 DESENVOLVIMENTO DA WIZARD	91
7 RESULTADOS E DISCUSSÕES	96
7.1 ESTUDOS DE CASOS.....	96
7.2 REUTILIZAÇÃO DE SOFTWARE.....	104
7.3 MODELAGEM 3D E INTERAÇÃO.....	105
7.4 PROCESSO DE CONSTRUÇÃO DO <i>ViMET</i>	106
7.5 VANTAGENS E LIMITAÇÕES.....	106
7.6 CONSIDERAÇÕES FINAIS.....	107
8 CONCLUSÕES.....	108
8.2 TRABALHOS FUTUROS.....	109
REFERÊNCIAS	111
GLOSSÁRIO - TERMOS MÉDICOS.....	121
APÊNDICES	122
APÊNDICE A – COOKBOOK	122
APÊNDICE B – JAVADOC	137

1 INTRODUÇÃO

Realidade Virtual (RV) é uma tecnologia de interface avançada que possibilita ao usuário não somente usar o sistema de *software*, como também ter a sensação de estar dentro de um ambiente tridimensional gerado por computador. De acordo com Vince (2004) e Kirner *et al.* (1999), o usuário pode explorar e até mesmo modificar o Ambiente Virtual (AV), o que lhe é possibilitado através de técnicas de navegação, interação e imersão.

As aplicações de RV são propícias para o desenvolvimento de ferramentas para o treinamento médico porque proporcionam ao estudante a possibilidade de repetir o treinamento inúmeras vezes, antes de executar o procedimento em um paciente real. Liu *et al.* (2003) afirmam que a RV proporciona ao cirurgião o aprendizado de detalhes da cirurgia em um AV ou paciente virtual, além de ter um *feedback* de sua atuação.

Segundo Monteiro *et al.* (2006), a Medicina vem se beneficiando com os avanços da RV por meio de sistemas que permitem maior nível de detalhes em exames, simulação de procedimentos e no ensino e treinamento de profissionais.

Sobre AVs aplicados à Medicina, Alberio e Oliveira (2006) afirmam que estes são utilizados, principalmente, para fins de diagnóstico, simulação de procedimentos invasivos, educação do médico e do paciente.

As aplicações de RV para a Medicina devem conter um alto grau de realismo. Para que isso seja garantido, a modelagem dos objetos 3D (Tridimensionais), as técnicas de deformação, detecção de colisão e as formas de interação devem ser os principais focos de atenção, entre os aspectos a serem considerados.

Como pode ser verificado em Blezek (2000), Montgomery (2001), Wagner *et al.* (2002), Webster (2005) todas as aplicações apresentadas por esses pesquisadores empregaram deformação, detecção de colisão e dispositivos hápticos específicos para garantirem um maior realismo e, assim, conquistar um envolvimento do usuário no processo. Porém estas aplicações possuem apenas uma técnica de deformação e detecção de colisão, não garantindo flexibilidade e nem possibilidade de comparação entre as inúmeras técnicas.

O LApIS (Laboratório de Aplicações de Informática em Saúde), pertencente ao Centro Universitário Eurípides de Marília, é constituído por alunos de mestrado e iniciação científica que desenvolvem projetos voltados para a área médica, com escopo principal direcionado ao treinamento virtual. A partir de trabalhos já desenvolvidos, percebeu-se a necessidade de integração de aplicativos, levando ao desenvolvimento do presente trabalho.

1.1 Objetivo do trabalho

Este trabalho possui como objetivo principal a construção de um *framework* orientado a objetos para aplicações de RV para treinamento médico e uma ferramenta de instanciação automática. O *framework* possui funcionalidades de detecção de colisão com precisão, estereoscopia e deformação.

Desta forma, como objetivos específicos para atingir o objetivo principal, tem-se:

- Estudar aplicações previamente desenvolvidas que executam deformação, detecção de colisão e estereoscopia para serem reutilizadas na implementação do *ViMeT*;
- Integrar estas aplicações;
- Implementar uma ferramenta de instanciação automática, utilizando um SGBD (Sistema Gerenciador de Banco de Dados);
- Desenvolver aplicações a partir do *ViMeT* para testar a flexibilidade e adequação do *framework*.

1.2 Justificativa do trabalho

O *framework* consiste em uma técnica de construção de *software* que possibilita a reutilização da análise, do projeto, do código e dos testes, constituindo, assim, uma técnica ideal para o presente projeto, visto que algumas funcionalidades (deformação, detecção de colisão e estereoscopia) já foram implementadas em projetos anteriores e seus códigos estão disponíveis para serem reutilizados na construção do *framework*.

Várias são as possibilidades de treinamento médico que podem ser realizadas utilizando RV e as mesmas constituem um domínio amplo para a construção de um *framework* para treinamento médico. O enfoque deste projeto são aplicações de exames de punção, para as quais é necessário um objeto que representa o órgão humano e outro que simula o equipamento necessário para a coleta de material. Com a instanciação do *framework* construído pretende-se tornar possível a construção de novas aplicações para treinamento

médico, com maior produtividade. Pretende-se, ainda, garantir a possibilidade de inserir novas funcionalidades ao *framework*, sem a necessidade de grandes alterações no código.

1.3 Disposição do trabalho

Esta dissertação possui, além desta introdução, sete capítulos, a saber:

Capítulo 2 – Apresenta os conceitos básicos de RV, assim como o estado da arte da RV na Medicina.

Capítulo 3 – Conceitua *framework* e apresenta suas classificações. Disponibiliza também as principais metodologias para a construção de *frameworks* e formas de utilização.

Capítulo 4 – Destaca os principais *frameworks* de RV citados na literatura e suas características relevantes. Também fornece um estudo comparativo entres os *frameworks* pesquisados e o *framework* proposto neste trabalho.

Capítulo 5 – Apresenta os detalhes da metodologia utilizada para o desenvolvimento, assim como os detalhes da implementação do *ViMeT*.

Capítulo 6 – Apresenta o desenvolvimento do Banco de Dados e da *Wizard*.

Capítulo 7 – Os principais resultados e testes são apresentados neste capítulo.

Capítulo 8 – Apresenta as conclusões e possíveis trabalhos futuros.

Em seguida estão as referências que serviram como embasamento teórico para a elaboração desta dissertação e os apêndices A e B que, respectivamente, apresentam o *cookbook* e o *javadoc*, que documentam o *framework* construído.

2 REALIDADE VIRTUAL NA MEDICINA

A RV auxilia na construção de ambientes sintéticos com características muito próximas das características do mundo real, o que pode tornar as aplicações propícias para treinamento e simulação. Em aplicações de RV há a possibilidade de interação e manipulação dos objetos no AV, além da possibilidade de navegação nesse ambiente. Estas características impulsionam os pesquisadores a desenvolverem aplicações de RV para treinamento médico.

Sobre isso, Machado (2003) afirmou que o uso de ferramentas baseadas em RV para treinamento pode oferecer uma nova forma de aprendizado, no quais imagens tridimensionais, exploração interativa e informações táteis podem ser combinadas para oferecer simulação mais realista.

Este capítulo mostra um panorama atual das aplicações de RV na Medicina e apresenta os dispositivos disponíveis para utilização em AV, que auxiliam na construção do *framework* proposto neste trabalho.

2.1 Conceitos Gerais de Realidade Virtual

De maneira simplificada, Kirner (1996) conceitua RV como a forma mais avançada de interação entre o usuário e o computador. Para Jacobson (1994), o principal objetivo desta nova tecnologia é fazer com que o usuário desfrute de uma sensação de presença no mundo virtual. Para Greenleaf (2004), a RV envolve geração de ambientes 3D, gerados por computador e um conjunto de ferramentas de interface que permitem aos usuários imergir, navegar e interagir com objetos no AV. Machado (2003, p. 24) afirma que

RV consiste em uma ciência que engloba conhecimento de diversas áreas, como a computação, eletrônica, robótica e cognição, dentre outras, visando oferecer sistemas computacionais que integram características de imersão e interatividade para simular ambientes reais onde os usuários têm estimulado simultaneamente os seus vários sentidos pelo uso de dispositivos específicos.

Na concepção adotada neste trabalho, RV é um conjunto de técnicas utilizadas para criar um AV com objetos modelados e dotados de características do mundo real, permitindo que o usuário possa navegar pelo AV e manipular estes objetos.

2.1.1 Dispositivos mais utilizados

Na década de 80, os dispositivos de interação para aplicações de RV eram muito caros e praticamente inacessíveis à maioria dos pesquisadores, porém, houve uma evolução muito rápida, tanto do *hardware* quanto do *software*, gerando um barateamento destes dispositivos. Sobre isto, Netto *et al.* (2002) afirmaram que o avanço das pesquisas na área tecnológica vem melhorando a qualidade dos dispositivos de *hardware*, gerando melhores capacetes de visualização, luvas e óculos mais leves e com mais recursos. Assim, tal avanço contribui para despertar maior interesse dos vários segmentos industriais e aumentar o número de usuários e de aplicações de RV no mundo todo.

Da mesma forma, existe uma grande quantidade de *software* disponível, com diferentes ferramentas de programação, voltadas para diferentes plataformas, o que torna possível a utilização de um computador pessoal para construir e explorar ambientes de RV.

Nas aplicações de RV não-imersivas, segundo Tori e Kirner (2006), o usuário é transportado parcialmente ao mundo virtual e são utilizados dispositivos convencionais, como teclado, mouse e monitor de vídeo. Para aplicações de RV imersivas, o usuário tem a sensação de presença no mundo virtual e para que isto ocorra, são necessários dispositivos não convencionais. Netto *et al.* (2002) dividiram estes dispositivos em dois tipos: dispositivos de entrada (dispositivos de interação e trajetória) e dispositivos de saída (dispositivos visuais, auditivos e hápticos).

Os dispositivos de entrada permitem a movimentação do usuário e sua interação com o mundo virtual. Sem um dispositivo de entrada de dados adequado o usuário participa da experiência de RV de forma passiva. Estes permitem ao usuário a movimentação e manipulação de objetos no mundo virtual, sendo representados pelas *datagloves* (luva de dados), pelos dispositivos com grau de liberdade¹ 6DOF (*Six degrees of freedom*) e sensores de entrada biológicos.

A segunda categoria de equipamentos, representada pelos dispositivos de saída, engloba equipamentos classificados como visuais, auditivos e hápticos. Os dispositivos visuais ajudam a proporcionar ao usuário a sensação de imersão no AV. Quanto melhor for a qualidade da imagem gerada, maior é a sensação de imersão. A imagem gerada pode ser

¹ A expressão graus de liberdade é utilizada para identificar o número de possibilidades de flexibilidade e/ou movimentação. Em um sistema de RV 6DOF o usuário pode navegar no AV nas direções X, Y e Z e rotacionar ao redor destes eixos (MACHADO, 2003).

monoscópica, ou seja, a mesma imagem é renderizada para os dois olhos, ou estereoscópica, quando são renderizadas duas imagens, uma para cada olho, com ângulos de rotação e translação diferentes. São mostrados alguns exemplos de dispositivos não convencionais na Figura 1 .

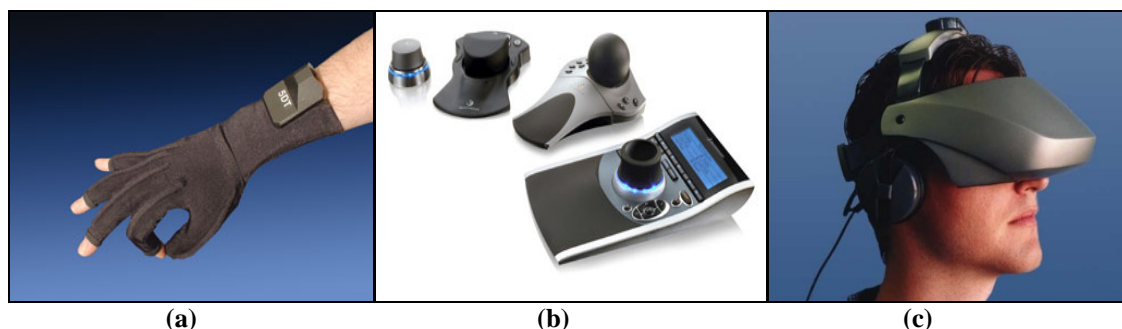


Figura 1 - Modelos de Dispositivos: (a) luva 5DT Data Glove (VIRTUAL REALTIES, 2005a), (b) dispositivos com seis graus de liberdade (LOGITECH, 2005), (c) HMD - 5DT HMD 800 (VIRTUAL REALTIES, 2005b)

Os dispositivos auditivos também são importantes para auxiliar a sensação de imersão no AV. Existem diversas placas de som projetadas para trabalhar com conjuntos de ferramentas que constroem mundos virtuais, sendo que a MIDI (*Musical Instrument Digital Interface*) é a interface mais conhecida para criar e controlar sons, nesses dispositivos.

Machado (2003) define dispositivos hápticos como sendo aqueles que incorporam sensores e atuadores, permitindo o monitoramento das ações do usuário e fornecendo-lhe sensação tátil e/ou de força. Os dispositivos hápticos são os mais relevantes para aplicações de treinamento médico, sendo utilizados para simulação de laparoscopia, sutura e outros procedimentos minimamente invasivos.

Segundo Freitas *et al.* (2003), o equipamento para laparoscopia virtual pode ser visto como um instrumento cirúrgico que se comunica com um computador por meio de uma porta serial. Existem dois tipos de equipamentos que foram projetados especificamente para este tipo de cirurgia: *Laparoscopic Surgical Workstation*, que é uma interface de *hardware* que simula a cirurgia laparoscópica e a interface para laparoscopia virtual VLI (*Virtual Laparoscopic Interface*), como mostrado, respectivamente, na Figura 2 (a) e (b).

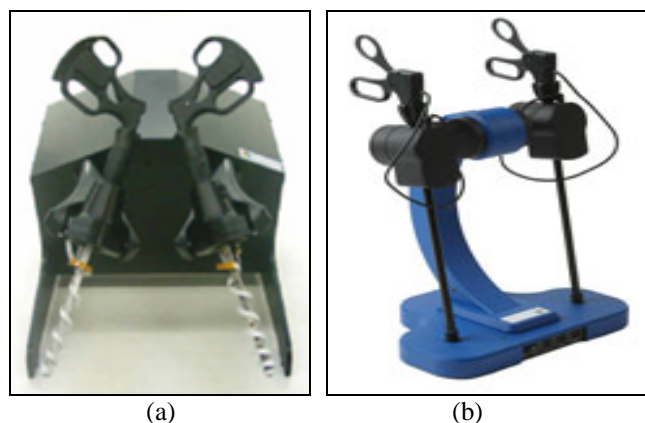


Figura 2 - (a) Plataforma de cirurgia laparoscópica, (b) Interface para laparoscopia virtual (IMMERSSION, 2005c)

O equipamento denominado *Immersion Medical's Endovascular AccuTouch System* consiste em um dispositivo para simulação de angioplastia coronária transluminal percutânea (ACTP) e propicia ao profissional de saúde um ambiente realístico. É composto por dispositivos que simulam o cateter e as guias, objetos que simulam as veias e artérias e monitores para visualização do cateter no AV. O *Endoscopy AccuTouch System* é um equipamento que simula procedimentos endoscópicos como gastroscopia, sigmoidoscopia e colonoscopia. Estes equipamentos podem ser observados, respectivamente na Figura 3 (a) e (b).



Figura 3 - (a) Equipamento para simulação de cirurgia endovascular (IMMERSSION, 2005a), (b) Simulador de Endoscopia (IMMERSSION, 2005b)

Na Figura 4 é mostrado o PHANTOM Omni (*Personal Haptic Interface Mechanism*), que é um braço mecânico compacto com seis graus de liberdade e que oferece reação tátil e de força com três graus de liberdade tanto em translação como em rotação.



Figura 4 - Equipamento PHANTON (SENSABLE TECHNOLOGIES, 2007)

2.2 Aplicações de Realidade Virtual na Medicina

Liu (2003) lembra que simuladores de cirurgia fornecem um ambiente para o médico treinar os procedimentos várias vezes, antes de fazer a intervenção em um paciente real. Porém, a RV não auxilia no treinamento médico somente por meio de simuladores de cirurgia, ela pode auxiliar na aprendizagem e na memorização de conceitos ou procedimentos como, por exemplo, a utilização de Atlas Virtuais. Outra forma de utilização de aplicações de RV na Medicina são os AV específicos para tratamento de fobias.

Machado *et al.* (2004) classificaram as aplicações de RV para a área médica em três grupos: planejamento, assistência e treinamento.

Os sistemas de planejamento permitem o estudo de um caso específico e geralmente utilizam imagens de ressonância magnética ou tomografia computadorizada do paciente para gerar uma réplica virtual da situação real. Os sistemas para assistência, por sua vez, são utilizados para dar suporte a um procedimento real, adicionando e sobrepondo elementos virtuais a uma situação real. Por fim, os sistemas de treinamento objetivam a incorporação de habilidades específicas e utilizam ambientes virtuais visando preparar o usuário para realizar um determinado procedimento, podendo simular situações genéricas (desassociadas das peculiaridades de um paciente específico) com alto grau de realismo. (MACHADO *et al.*, 2004)

Ainda sobre isso Netto *et al.* (2002) definem o grupo de treinamento como sistemas que objetivam a incorporação de habilidades específicas, e utilizam AV visando preparar o

usuário para realizar um determinado procedimento, podendo simular situações genéricas com alto grau de realismo. Para Machado *et al.* (2004), um dos grandes desafios dos sistemas de RV para treinamento médico é oferecer condições que reproduzam exatamente aquilo que o médico vê e sente na realidade.

Na Seção 2.2.1 é apresentado o estado da arte dentro do escopo proposto no trabalho que são as ferramentas para treinamento médico. As ferramentas de treinamento médico utilizam AV para simular procedimentos de forma genérica, motivo pelo qual ocorreu a escolha de tal escopo. Com a finalidade de mostrar a diferença do estágio de desenvolvimento no Brasil em relação a centros de pesquisa estrangeiros, primeiramente são apresentadas as aplicações estrangeiras, depois as ferramentas desenvolvidas no Brasil e, por último, as aplicações desenvolvidas até o momento no LApIS, do Centro Universitário Eurípides de Marília – UNIVEM.

2.2.1 Aplicações desenvolvidas em outros países

Com a finalidade de aumentar a qualidade no treinamento do procedimento de anestesia regional foi desenvolvida uma aplicação de RV destinada aos médicos residentes. A aplicação enfoca o treinamento do procedimento em pacientes com câncer abdominal, que não podem fazer cirurgia ou não toleram o uso de drogas, estas já não surtem mais efeito.

O simulador de anestesia regional tem como objetivo criar um AV realístico para que médicos residentes possam treinar o procedimento em um paciente virtual. Segundo Blezek *et al.* (2000), o sistema possui duas funções primárias: visualização e *feedback* de força. Para a imersão do usuário no AV é usado um HMD (*Head Mounted Display*) e o dispositivo de *force feedback* PHANTON. Para que a exibição seja rápida e a manipulação de estruturas anatômicas realística é utilizado o VHM (*Visible Human Male*), que consiste em um banco de dados de imagens tridimensionais, construídas a partir de imagens médicas provenientes de cadáveres (NLM, 1990). Na Figura 5 mostram-se as possíveis formas de visualização do AV.

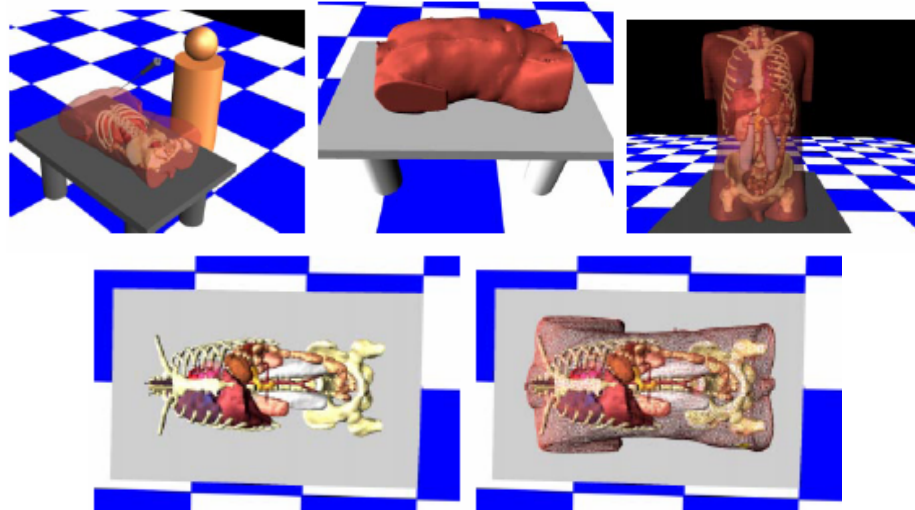


Figura 5 - Possíveis formas de visualização do simulador de anestesia (BLEZEK *et al.*, 2000)

Uma outra aplicação de RV voltada para o treinamento médico é o simulador háptico de sutura. Segundo Webster *et al.* (2001), a intenção era desenvolver um simulador realístico, simples de operar, econômico e com a possibilidade de difundir o seu uso, necessitando apenas de um computador pessoal para utilizá-lo. O simulador possui componentes que modelam e deformam a pele, o tecido e o material de sutura em tempo real, além de registrar todos os estados de atividade durante a tarefa.

O *software* de simulação calcula forças de contato e gera deslocamentos de tecido. Os cálculos da força de resistência variam de acordo com a profundidade e o ângulo de inserção da agulha. A força também muda quando a agulha perfura a pele virtual, quando sai da pele e quando é necessário apertar o material da sutura. Na Figura 6 mostra-se a interface do simulador.

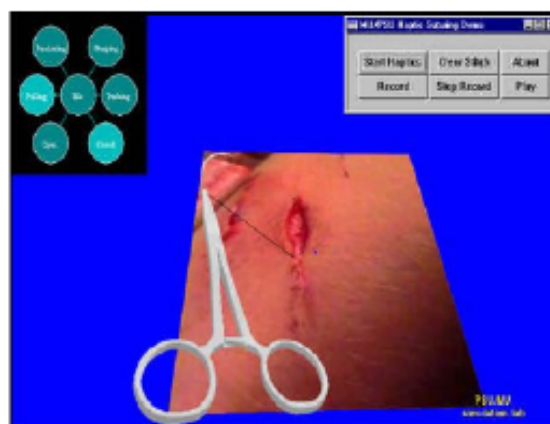


Figura 6 - Interface de um treinamento de sutura sendo executado no simulador (WEBSTER *et al.*, 2001).

Montgomery *et al.* (2001) apresentam um simulador cirúrgico baseado em RV para histeroscopia diagnóstica e cirúrgica. O simulador pode ser usado para análise e planejamento cirúrgico, treinamento de microcirurgia e simulação de suturas. Segundo Afonso (2000), histeroscopia consiste em um exame que possui três fases: passagem do canal cervical, inventário da cavidade uterina (vista panorâmica e com detalhes no aumento de mais ou menos 20 vezes) e revisão cervical.

O sistema descrito permite ao usuário visualizar dados estereoscópicos ou monoscópicos, como objetos *wireframe* (visualização dos pontos e linhas que descrevem o modelo). Possui também uma integração entre o modelo anatômico 3D e os dispositivos hápticos especializados (Figura 7). O simulador ainda possui uma interface que permite ao usuário a seleção de vários procedimentos, a apresentação de dados relativos ao exame, além de cronometrar a duração do procedimento e informar a força utilizada durante as ações executadas.



Figura 7 - Simulador de Histeroscopia (Adaptado de MONTGOMERY *et al.*, 2001)

O simulador oferece ao usuário uma simulação realística de manobras cirúrgicas usadas durante o alargamento cervical e os procedimentos de histeroscopia, como a remoção de lesões intra-uterinas e a separação e ressecção endometrial. Esta simulação permite a transferência das habilidades virtuais para a melhoria dos procedimentos reais. Na Figura 8 mostra-se o AV do simulador em três momentos: o momento de alargamento cervical, da seleção do endométrio e o momento da ressecção.

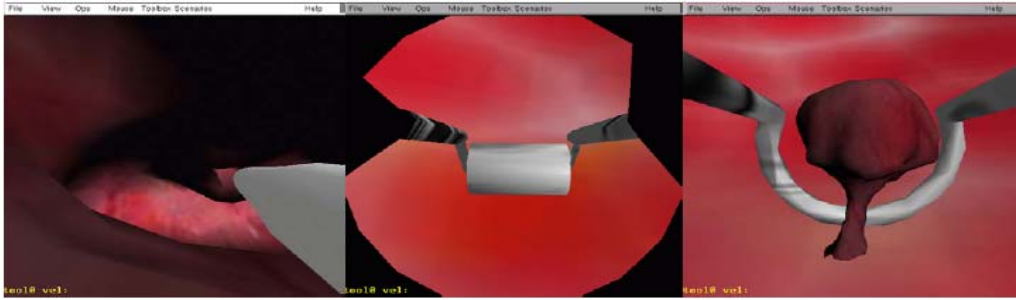


Figura 8 - Ambiente Virtual do Simulador de Histeroscopia (MONTGOMERY *et al.* 2001)

Outro exemplo de simulador baseado em RV é o *EyeSi*, um sistema que tem como objetivo simular cirurgias intra-oculares. Segundo Wagner *et al.* (2002), o projeto é baseado em uma instalação mecânica completamente modelada para fornecer ao cirurgião em treinamento todas as percepções sensoriais do olho, como pode ser observado na Figura 9 (a).

O olho mecânico tem o mesmo grau de liberdade de rotação do olho humano, sendo que o efeito dos músculos é modelado por um conjunto de molas sobre cada eixo de rotação. Para a deformação foi utilizado um método híbrido que inclui o método de elementos finitos e o método *Mass Spring* (Massa-Mola). Para a detecção de colisão foi utilizada parte do pacote de detecção de colisão *I-Collide* (CHUNG e WANG, 1996 *apud* WAGNER, 2002), junto com o método *boundingbox*. A visualização realista do cenário de operação inclui efeitos de iluminação e sombras como pode ser visto na Figura 9 (b).

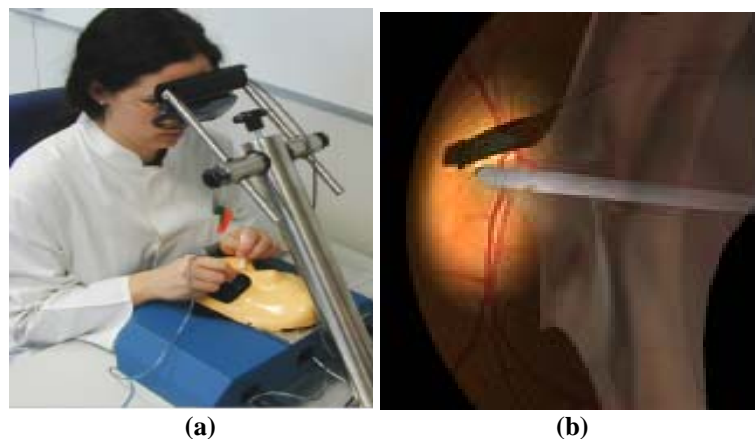


Figura 9 - (a) Simulador EyeSi, (b) Cenário do EyeSI (WAGNER *et al.*, 2002)

Webster *et al.* (2005) desenvolveram uma aplicação específica para simular o procedimento de Extração Extra-Capsular do Cristalino (EECC). Esta aplicação utiliza o sistema *EYESIY™*, conforme mostrado na Figura 10 (a). Na Figura 10 (b) mostra-se o modelo

utilizado para deformação. Esta aplicação possui praticamente as mesmas características da aplicação construída por Wagner *et al.* (2002).

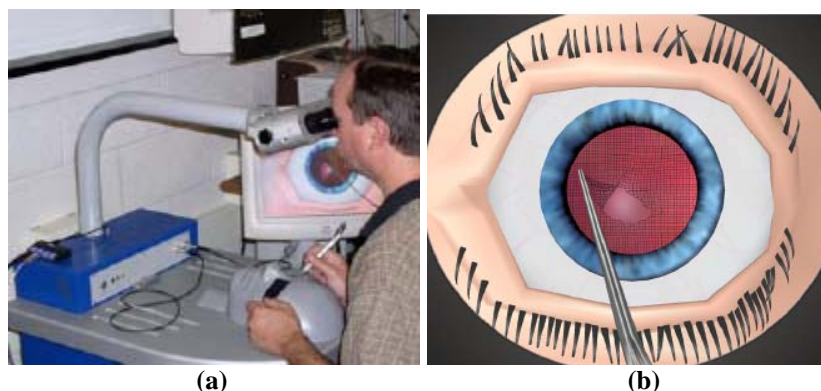


Figura 10 - (a) Sistema EYESIY™, (b) Modelo Mass-Springs usado para romper a membrana (WEBSTER *et al.*, 2005)

Um exemplo utilização de RV na Medicina são as aplicações para reabilitação. Niniss e Inoue (2006) desenvolveram um sistema de RV para auxiliar deficientes físicos que utilizam cadeiras de rodas. Nesse sistema é criado um AV que simula um local que o paciente já conhece. Por meio de dispositivos específicos o paciente pode explorar o AV para reconhecimento, familiarização com a cadeira de rodas e, principalmente, a reabilitação motora.

2.2.2 Pesquisas no Brasil

Machado (2003) apresenta um simulador para treinamento de coleta de medula óssea implementado primeiramente para procedimentos pediátricos, sendo considerado o primeiro sistema de RV para treinamento médico desenvolvido no Brasil. Esse modelo 3D é composto pela camada externa (pele) e camadas internas de tecido da região. Cada uma destas camadas oferece uma resistência diferente devido às suas propriedades físicas.

A simulação do treinamento da coleta é realizada em um modelo tridimensional da região pélvica e a diferença de resistência das camadas é sentida pelo médico no momento da penetração da agulha, através do dispositivo háptico PHANTON, como mostrado na mostrado na Figura 11 (a) e (b), respectivamente.

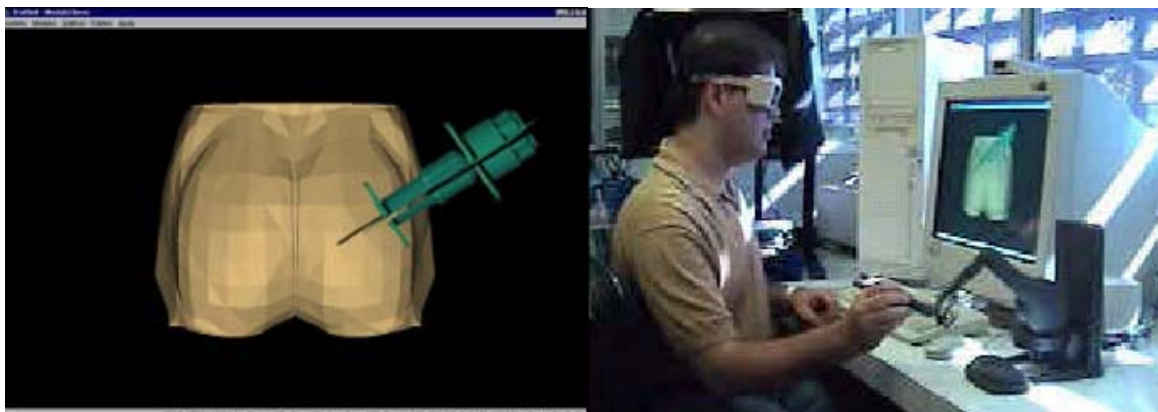


Figura 11 - (a) Modelo tridimensional da região pélvica, (b) Utilização do PHANTON (MACHADO, 2003)

Outra aplicação desenvolvida no Brasil é o Sistema de Simulação Tridimensional para Treinamento e Planejamento de Hepatectomia. Segundo Benes *et al.* (2004), o *software* desenvolvido proporciona ao usuário a possibilidade de interagir com o AV e, também, com o fígado virtual modelado. A interação é realizada através de rastreadores de movimento e óculos especiais para estereoscopia.

O sistema possibilita o treinamento e o planejamento de uma ressecção anatômica do fígado e a simulação de cortes a serem realizados sobre o órgão durante o procedimento cirúrgico. O AV do sistema é composto por uma mesa e um quadro de projeção dos cortes realizados. Com os planos de corte é possível testar várias possibilidades de seccionamento do fígado e obter estimativas de volume real do órgão em questão. O perfil do corte realizado no fígado é mostrado em tempo real.

O *CyberMed* consiste em um sistema baseado em RV para apoiar o treinamento médico através de explorações interativas do corpo humano e da simulação realista de procedimentos médicos em um AV imersivo. Segundo Machado *et al.* (2004), o sistema foi desenvolvido sobre a plataforma *VirtWall*, que é um sistema de projeção estereoscópica de ambientes simulados confeccionado com equipamentos de informática encontrados no mercado brasileiro, com alto desempenho e capacidade de processamento gráfico (MORAES *et al.*, 2003). Na Figura 12 mostra-se a visualização de um coração no *VirtWall*.

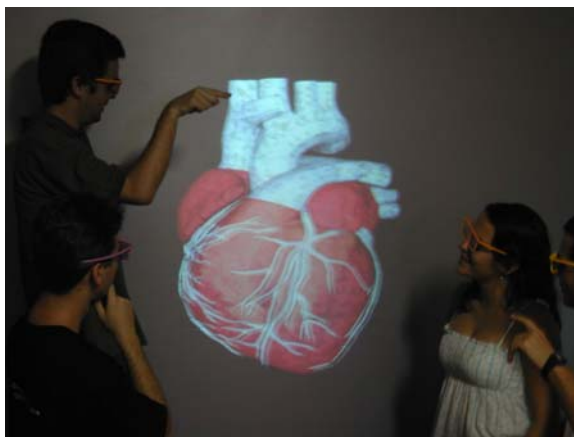


Figura 12 - Estudo do coração no *VirtWall* (MONTEIRO, 2006a)

As principais características do *CyberMed* são: visualização tridimensional, uso de modelos realistas, interação espacial com sensação de toque, deformação interativa das estruturas tocadas, compartilhamento visual e a supervisão e avaliação das ações do usuário. De acordo com Machado *et al.* (2004), o sistema tem sido utilizado na visualização interativa de estruturas do corpo humano. Os objetos tridimensionais foram modelados semelhantes à bacia, cabeça e coração humanos. Cada um deles é composto por camadas externas e internas detalhadas através de malhas triangulares.

O sistema oferece também um conjunto de menus por meio dos quais o usuário pode variar o nível de semitransparência, escolher as camadas visíveis ou não do modelo e requisitar uma descrição sobre a estrutura observada. Para a interação, o *CyberMed* utiliza dispositivos convencionais, tais como *mouse*, *joystick* e teclado.

A RV pode também auxiliar no planejamento e previsão de resultados de cirurgias. Dentre as cirurgias de natureza cosmética, as destinadas a alterar o tamanho e a forma dos seios são as mais frequentes, sendo que a paciente tem muitas opções para escolher o procedimento. Balaniuk *et al.* (2006) desenvolveram um sistema que possibilita visualizar os possíveis resultados da cirurgia plástica e tomar decisões com relação à técnica, tamanho e forma. Esta escolha é muito importante tanto para o médico quanto para a paciente.

Um sistema de RV para treinamento e planejamento de tratamentos em ortodontia foi desenvolvido por Rodrigues *et al.* (2006), permitindo, a partir de configurações iniciais fornecidas pelo usuário, a geração de modelos customizados dos pacientes e dos aparelhos ortodônticos, além da simulação do comportamento dentário para diferentes cenários de tratamento.

O tratamento de fobia também é contemplado em RV, Paiva *et al.* (2006) desenvolveram um sistema de RV para o tratamento de pessoas que possuem medo de dirigir

automóveis. O sistema simula o carro em um cenário que contém ruas de uma cidade, onde o usuário vivencia situações parecidas com as encontradas na realidade. As características do AV podem ser escolhidas em função de diversos fatores, tais como: o período do dia, presença de chuva ou neblina e estilos de solo (depressão, asfalto, terra, entre outros).

2.2.3 Pesquisa no LApIS

O LApIS é constituído por alunos de graduação, com projetos de iniciação científica, e por alunos do Programa de Mestrado em Ciência da Computação do UNIVEM, tendo como foco as pesquisas em RV e o desenvolvimento de ferramentas de baixo custo para o treinamento médico. O grupo também desenvolve projetos que auxiliam no diagnóstico médico por meio de processamento de imagens. Algumas das pesquisas que já foram concluídas e publicadas são descritas a seguir.

Lima (2004) desenvolveu um protótipo de ferramenta de simulação para a realização do exame de punção da mama, conforme pode ser observado na Figura 13, com características de Realidade Virtual não imersiva em uma plataforma convencional, tendo as interações disponibilizadas pelo mouse e a visualização em monitor de vídeo.

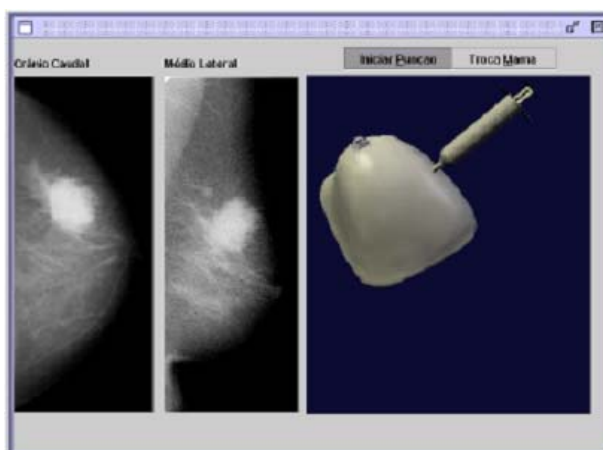


Figura 13 - Protótipo para punção de mama. (LIMA, 2004)

A interface disponibiliza duas imagens obtidas através de exame mamográfico e ainda os objetos sintéticos, que são representados por uma mama e uma seringa. O protótipo informa ao usuário quando houve a colisão, ou seja, quando a agulha atingiu o nódulo. As

imagens são armazenadas em um banco de dados, cujo conteúdo foi fornecido por especialistas em mastologia.

Outro projeto do LApIS é a implementação de uma ferramenta para construção dinâmica de estruturas de feto, que constitui um sistema composto por procedimentos responsáveis por modelar dinamicamente estruturas tridimensionais. Segundo Hermosilla (2004) foram feitos estudos das características das estruturas fetais, a fim de definir uma interface para obter os dados necessários (medidas de estruturas fetais) e proporcionar a geração dinâmica dos objetos. Na Figura 14 mostram-se a interface e uma caixa de diálogo com o código em VRML (*Virtual Reality Modeling Language*) gerado a partir dos parâmetros fornecidos. Na Figura 15 as imagens geradas pelo sistema são mostradas.

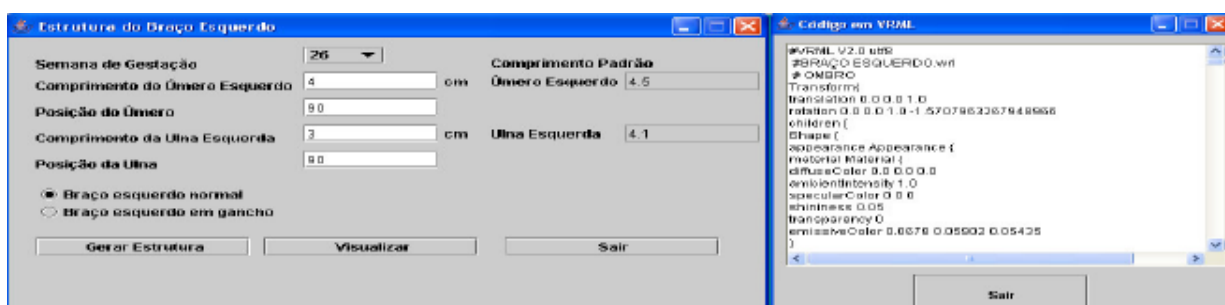


Figura 14 - Tela para passagem de parâmetros e caixa de diálogo (HERMOSILLA, 2004)



Figura 15 - Imagens geradas pelo sistema (HERMOSILLA *et al.*, 2005)

Foi construído também um Atlas Virtual Tridimensional da Mama com o objetivo, segundo Ramos (2005), de possibilitar aos estudantes de Medicina uma forma interativa de explorar esta região do corpo humano e verificar as fases de desenvolvimento do câncer neste órgão. O Atlas consiste de três módulos básicos: Anatomia Mamária, Fisiopatologia do Câncer de Mama e Estereoscopia. O módulo de Anatomia da Mama é o responsável por disponibilizar as estruturas tridimensionais (3D) modeladas, referentes às estruturas presentes no órgão humano e às informações inerentes a cada estrutura. No módulo de fisiopatologia o usuário pode observar animações, com o objetivo de conhecer como um tumor se desenvolve,

sendo conhecidas as estruturas nas quais o câncer de mama se desenvolve com maior frequência e os estágios de avanço da doença.

O usuário pode escolher, entre as opções de visualização do câncer, o quadrante da mama em que deseja observar o tumor se desenvolvendo, a estrutura mamária, o estágio de desenvolvimento da doença e o tipo de câncer. Na Figura 16 mostra-se a interface em que o usuário pode selecionar as características desejadas e na Figura 17 mostra-se a imagem gerada a partir desta seleção.

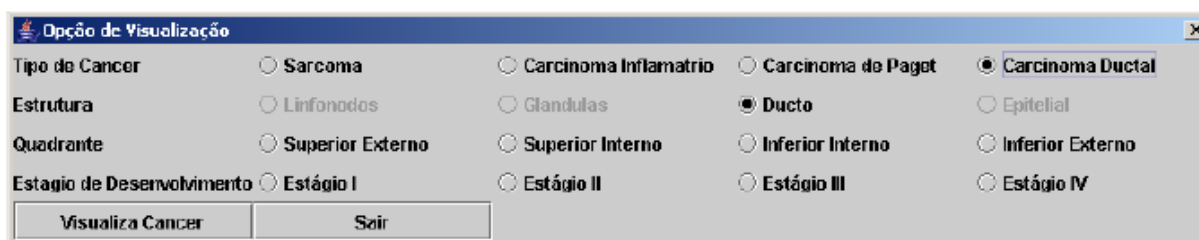


Figura 16 - Interface do Atlas (RAMOS, 2005)



Figura 17 - Imagens geradas a partir das opções selecionadas pelo usuário (RAMOS, 2005)

O módulo de Estereoscopia gera anaglifos a partir dos modelos tridimensionais fornecidos pelo Módulo de Anatomia da Mama. A função deste módulo é proporcionar o realismo e a imersão do usuário na cena virtual.

Berti (2005) construiu um sistema de visualização de grandes quantidades de dados, usando técnicas de Realidade Virtual e modelagem 3D. Para o estudo de caso foi utilizada uma base, com grande quantidade de dados e imagens médicas. No sistema, o usuário seleciona os parâmetros da consulta em uma tela convencional, obtém os registros filtrados e opta por dois modelos de representação gráfica. A cena 3D é gerada e o usuário pode navegar pelos registros filtrados pela consulta, possibilitando uma visão genérica dos dados e, simultaneamente, uma observação mais detalhada. A visualização é construída em ambiente de Realidade Virtual não imersiva, utilizando como dispositivos físicos de interação o mouse, responsável pela manipulação dos objetos da cena e seleção dos parâmetros da consulta e os monitores convencionais, responsáveis pela visualização.

Riquelme (2005) realizou um estudo comparativo dos métodos oferecidos pelas bibliotecas Java3D e *WorldToolkit* (SENSE8, 2005) para detecção de colisão, direcionando o estudo para uma ferramenta desenvolvida para treinamento médico. Nos testes realizados, observou-se que os métodos fornecidos por essa tecnologia não eram suficientes para uma detecção de colisão com precisão e rapidez e, por isso, propôs uma nova técnica que utiliza a equação do plano para refinar os métodos fornecidos pelas bibliotecas analisadas.

Luz *et al.* (2004) implementaram um sistema que segmenta estruturas fetais de imagens 2D (bidimensionais) de Ultra-som, a fim de representá-las através de objetos sintéticos tridimensionais e permitir o emprego de técnicas de Realidade Virtual para possibilitar a interação do usuário com as imagens, criando uma alternativa de baixo custo para a identificação e visualização tridimensionais das estruturas fetais.

Além dos trabalhos citados estão ainda sendo desenvolvidas pesquisas para desenvolvimento de detecção de colisão com precisão, estereoscopia e deformação que, por fazerem parte da proposta deste projeto, serão apresentadas adiante.

2.3 Considerações Finais

Neste capítulo foram apresentados os conceitos básicos de RV, os principais dispositivos utilizados para aplicações de RV, bem como o estado da arte das aplicações de RV para Medicina. Foram descritas aplicações estrangeiras, aplicações realizadas por diversos grupos de pesquisa do Brasil e bem como os resultados de pesquisas obtidos pelos integrantes do LApIS.

No próximo capítulo são apresentados os conceitos de *frameworks* orientados a objetos, suas principais classificações, as metodologias utilizadas para sua construção e possíveis formas de utilização.

3 FRAMEWORKS ORIENTADOS A OBJETOS

O desenvolvimento de *frameworks* orientados a objetos² e de aplicações a partir de um dado *framework* têm sido motivo de pesquisa desde a década de 80, com o surgimento do paradigma de orientação a objetos. Para Silva (2000), a abordagem desses *frameworks* utiliza o paradigma de orientação a objetos para produzir uma descrição de um domínio que permita a reutilização. A reusabilidade por meio de um *framework* se dá através da herança e ou composição e, como todas as formas de reuso de *software*, possui limitações. Há a possibilidade de utilizar simultaneamente algumas técnicas de reuso, tais como, *frameworks* com Padrões de Projetos (*design patterns*) criados por Gamma *et al.* (1995) ou *frameworks* com componentes³.

Neste capítulo é apresentado o conceito de *framework*, suas principais características, bem como suas classificações. São descritas as metodologias mais utilizadas no desenvolvimento de um *framework* e as formas de utilização. As vantagens e desvantagens de um *framework* também são apresentadas.

3.1 Conceituação

Um *framework* é definido como um conjunto de classes abstratas e concretas usadas para o desenvolvimento de uma aplicação com domínio específico. Vários pesquisadores conceituaram *framework* como uma forma de reuso de *software* com características do paradigma da orientação a objetos. A seguir, são apresentados alguns destes conceitos.

Para Mattsson (1996), *framework* é uma arquitetura desenvolvida com o objetivo de se obter a máxima reutilização, representada como um conjunto de classes abstratas e concretas, com grande potencial de especialização. Para Johnson (1997), um *framework* pode ser definido como sendo o esqueleto de uma aplicação, que pode ser instanciado por um desenvolvedor de aplicações.

² Neste trabalho o termo “*framework* orientado a objetos” será substituído por “*framework*” para efeito de simplificação.

³ Segundo Bastos *et al.* (2004), um componente é um artefato de software, com interfaces internas e externas bem definidas, que é independente de outros artefatos de software, sendo independentes entre si e capazes de se comunicar e trabalhar em conjunto sem conhecer um ou outro.

Johnson e Foote (1988) afirmaram que um *framework* trata-se de uma aplicação semicompleta reutilizável que, quando especializada, produz aplicações personalizadas e, também que um *framework* é um conjunto de objetos que colaboram com o objetivo de cumprir um conjunto de responsabilidades para uma aplicação ou um domínio de um subsistema.

Silva (2000) define *framework* como uma estrutura de classes que se relacionam dando origem a uma aplicação inacabada que podem gerar um conjunto de aplicações de um determinado domínio. De acordo com Braga (2002), a base do *framework* é fornecida pelas classes abstratas, a partir das quais classes concretas ou outras classes abstratas podem ser implementadas. Para um melhor entendimento, a seguir definem-se ambas as classes:

- **Classes Abstratas** - são as classes que não possuem objetos instanciados a partir dela. Segundo Freiberg (2002), classe abstrata é um projeto que especifica uma classe e a árvore de subclasses que poderá ser produzida a partir dela.
- **Classes Concretas** - são classes que permitem a instanciação de seus objetos. Freiberg (2002) afirma que as classes concretas definem uma estrutura de dados e métodos construídos para satisfazer uma necessidade específica.

Framework é considerado uma forma de reúso de *software*, pois possibilita o reúso do projeto, do código e da análise quando esta está disponível. Sobre isso, Johnson e Foote (1988) afirmaram que um *framework* orientado a objetos representa uma categoria de artefatos de *software* potencialmente capazes de promover reúso de alta granularidade. Os autores salientam que a característica mais importante dos *frameworks* é a inversão de controle, pois usualmente um desenvolvedor utiliza um programa principal para chamar as bibliotecas que deseja reutilizar.

De acordo com Johnson (1997), o desenvolvedor decide quando chamar os componentes, sendo responsável pela estrutura global e fluxo de controle do programa. No caso do *framework*, o programa principal é reusado e o desenvolvedor pode construir novos componentes e ligá-los ao programa principal, sendo o *framework* quem determina a estrutura global e o fluxo de controle do programa.

Em Taligent (1994) são apresentadas as principais diferenças de reutilização de bibliotecas e *frameworks*, reproduzidas na Figura 18.

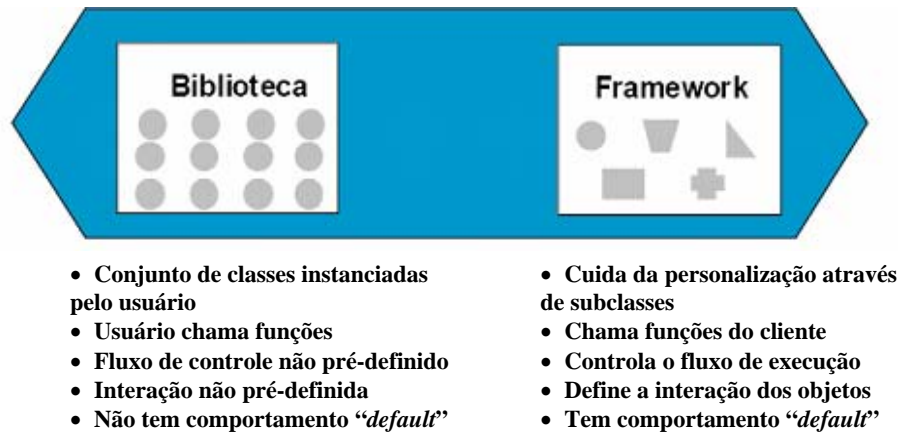


Figura 18 - Principais diferenças de uma Biblioteca e um *framework*. (Adaptado de TALIGENT, 1994)

3.2 Classificações de *frameworks*

Os *frameworks* podem ser classificados de acordo com o seu escopo, ou seja, a partir do domínio da aplicação e de acordo com a sua extensão.

Fayad *et al.* (1997) propuseram uma classificação quanto ao escopo, estabelecendo as seguintes categorias:

- *Framework* de infra-estrutura (*System Infrastructure framework*), utilizados na construção de sistemas operacionais, sistemas de comunicação, interface com o usuário e ferramentas de processamento de imagem. Este tipo de *framework* não é voltado para usuários finais.
- *Framework* de integração *Middleware* (*Middleware Integration framework*), usado para integrar aplicações e componentes distribuídos, este tipo de *framework* facilita a tarefa do desenvolvedor, pois aumenta a possibilidade de modular, reusar e estender a infra-estrutura do *software*. O exemplo mais conhecido é o *framework* ORB (*Object Request Broker*) criado pelo OMG (*Object Management Group*TM)⁴ e os bancos de dados transacionais.
- *Framework* de Aplicação Empresarial (*Enterprise Application framework*) são voltados para as aplicações comerciais e a área dos negócios. Este tipo de *framework* requer um investimento financeiro maior para ser desenvolvido. No entanto, garante

⁴ O OMG é uma organização sem fins lucrativos que promove a padronização de tecnologias orientadas a objetos publicando diretrizes e especificações (DEITEL e DEITEL, 2005).

o retorno do investimento, uma vez que permite o desenvolvimento de aplicações e produtos para usuários finais.

A forma como se efetiva o reuso por meio de um *framework* gera uma classificação de acordo com sua extensão. Inicialmente foram propostas duas categorias: caixa preta (*black box*) e caixa branca (*White box*), porém ambas possuíam algumas deficiências. Na tentativa de saná-las foi gerada uma nova categoria chamada caixa cinza (*gray box*) que possui classes concretas e abstratas.

Nos *frameworks* caixa branca o reuso acontece por herança e, de acordo com Freiburger (2002), são orientados à arquitetura. Sobre isso, Silva (2000) afirmou que em um *framework* orientado à arquitetura, a aplicação deve ser gerada a partir da criação de subclasses das classes do *framework*. Essas características o tornam mais flexível e fácil de projetar. Segundo Fayad *et al.* (1999), *frameworks* caixa branca tendem a produzir sistemas que são muito ligados aos detalhes específicos das hierarquias de herança do *framework*. Segundo Roberts e Johnson (1996), O *framework* caixa branca pode evoluir e se tornar um *framework* caixa preta e isso deve ser feito de forma gradativa, implementando-se várias alternativas que possam ser aproveitadas na instanciamento do *framework*.

Em um *framework* caixa preta o reuso acontece por composição, ou seja, sua extensibilidade é garantida definindo interfaces para os componentes, por composição de objetos. Fayad *et al.* (1999) afirmaram que *frameworks* caixa preta são estruturados usando composição de objeto e delegação no lugar de herança. O desenvolvedor poderá construir sua aplicação combinando as classes concretas do *framework* e, para isso, terá que se preocupar apenas com sua interface. Segundo Freiburger (2002), o *framework* caixa preta é orientado a dados. Silva (2000) afirmou que em um *framework* orientado a dados, diferentes aplicações são produzidas a partir de diferentes combinações de objetos, instâncias das classes presentes no *framework*. Esta categoria de *framework* é mais abstrata e menos flexível, porém a manutenção das aplicações derivadas é complicada e demanda muito tempo. Segundo Fayad *et al.* (1997), os *frameworks* caixa preta são mais difíceis de desenvolver, pois o desenvolvedor deve prever o maior número de possíveis aplicações.

Os *frameworks* caixa cinza são constituído por classes abstratas e concretas, ou seja, é uma mistura dos *frameworks* caixa branca e caixa preta, mantendo a flexibilidade do caixa branca e a facilidade de extensão do caixa preta. Sua forma de reuso é obtida por meio de herança, por ligação dinâmica e por interface de definição. Segundo Fayad *et al.* (1999), um bom *framework* caixa cinza tem muita flexibilidade e extensibilidade, além da habilidade de ocultar informações desnecessárias ao desenvolvedor da aplicação.

A extensibilidade de um *framework* é definida pelas partes fixas e variáveis. Pree *et al.* (1995) denominam as partes flexíveis (*hot spots*) de pontos de especialização. As partes fixas (*frozen spots*) são aquelas que não mudam, ou seja, são constantes em todas as instâncias do *framework*, pois possuem aspectos do domínio da aplicação. Esses pontos são representados pelas classes que podem ser adaptadas em novas aplicações determinando, assim, o grau de extensibilidade do *framework*.

Froehlich *et al.* (1997) nomeiam as partes variáveis de gancho (*hook*), afirmando que são pontos do *framework* passíveis de mudança e as formas de mudança utilizadas são preenchimento de parâmetros ou criação de subclasses. Segundo Braga (2002), para entender como funciona o reúso de *frameworks* é necessário conhecer os conceitos de Método-Gancho (*Hook Method*) e Método-Gabarito (*Template Method*).

O Método-Gancho é definido em uma classe abstrata do *framework* e sua implementação é fornecida pelas subclasses concretas. O Método-Gabarito é um dos Padrões de Projetos criados por Gamma *et al.* (1995) e, segundo Welfer *et al.* (2005), possui a finalidade de prover uma lógica comum para todas as subclasses de uma aplicação. Para isso, a classe mãe implementa um método (que, por sua vez, encapsula um algoritmo), utilizado por todas as subclasses evitando, assim, sua replicação em cada uma delas.

3.3 Desenvolvimento de um *framework*

O desenvolvimento de *frameworks* orientados a objetos tem sido pesquisado nos últimos anos devido a sua capacidade de alterabilidade e extensibilidade. A construção de um *framework* promete em longo prazo, uma economia de tempo ao desenvolver novas aplicações e uma diminuição significativa de erros e, por consequência, um aumento de qualidade do produto final que, nesse caso, é um *framework* ou um conjunto de aplicações derivadas dele. Segundo Silva (2000), a principal característica buscada ao desenvolver um *framework* é a generalidade em relação a conceitos e funcionalidades do domínio tratado. Na

Figura 19 apresenta-se o método tradicional de desenvolver uma aplicação orientada a objetos e na Figura 20 mostra-se o desenvolvimento por meio do *framework*, podendo notar também a extensibilidade do *framework*.



Figura 19 - Desenvolvimento tradicional de uma aplicação orientada a objetos (JOHNSON, 1993)

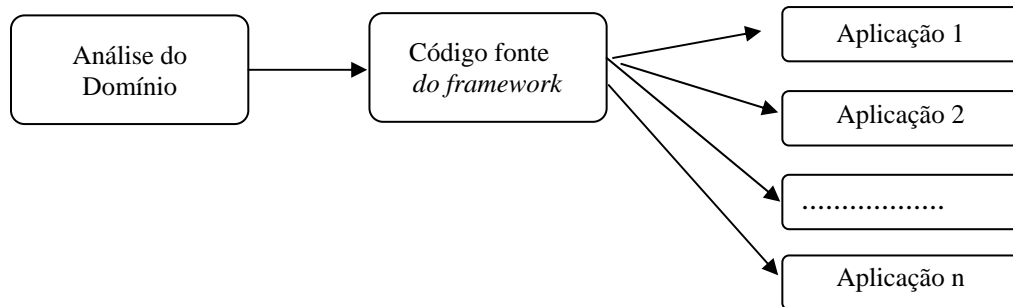


Figura 20 - Desenvolvimento baseado em *frameworks* (Adaptado de MATTSSON, 1996)

A necessidade de utilizar uma metodologia no desenvolvimento de um *framework* foi apontada por Silva (2000) e Braga (2002).

Silva (2000) utilizou em seu trabalho três metodologias: Projeto Dirigido por Exemplo, proposto por Johnson (1993); Projeto Dirigido por *Hot Spot* e de Pree (1994 *apud* BRAGA, 2002).

Braga (2002) citou as metodologias de Pree *et al.* (1999), Schmid (1997, 1999), Roberts e Johnson (1996) e Bosch *et al.* (1999). No presente trabalho foram pesquisadas as principais características destas metodologias, que são mostradas nas seções seguintes.

3.3.1 Metodologia proposta por Johnson (1993)

Johnson (1993) sintetiza o conceito de *framework* como o resultado da análise do domínio, a decomposição do problema e a representação deste problema por meio de um programa, com a possibilidade de reusar a análise, o projeto e o código de uma aplicação já existente, sendo a abstração do domínio o próprio *framework*.

O processo de desenvolvimento baseado no Projeto Dirigido por Exemplos consiste em três fases: análise, projeto e teste. Para o autor, essas fases se constituem na maneira ideal de construir um *framework*:

- **Fase da análise do domínio:** nesta fase os procedimentos envolvidos são o entendimento das abstrações já conhecidas, coleta de exemplos de possíveis programas a serem desenvolvidos a partir do *framework* e adequação de cada exemplo.
- **Fase do projeto:** a hierarquia das classes é projetada de forma que possa ser especializada para abranger todos os exemplos.
- **Fase de testes:** os testes são feitos usando o *framework* para desenvolver os exemplos e cada exemplo deve ser uma aplicação separada.

3.3.2 Metodologia proposta por Roberts e Johnson (1996)

Essa metodologia é também chamada “Evolução de *Frameworks*” e se preocupa tanto com a construção quanto com a instanciação. Nesta abordagem, os autores utilizam uma Linguagem de Padrão⁵ que possui nove padrões. Estes, ao serem aplicados, geram um *framework* caixa preta com todas as facilidades para sua instanciação. O processo de desenvolvimento dessa metodologia é baseado na aplicação dos padrões. O primeiro padrão sugere que três aplicações concretas sejam construídas e depois generalizadas. A regra geral é: construir a primeira aplicação, construir a segunda aplicação ligeiramente diferente da primeira, e por último construir a terceira aplicação diferente das duas anteriores, porém, todas dentro de um mesmo domínio. Com isso, as abstrações comuns ficarão evidentes.

Os padrões seguintes levam, através de iterações, um *framework* caixa branca a se transformar de forma gradual em um *framework* caixa preta. Para isso utilizam bibliotecas de componentes, *hot spots*, objetos plugáveis e objetos de granularidade menor. Quando os componentes são acrescentados à biblioteca percebe-se que freqüentemente partes fixas dos códigos são requisitadas e que há a necessidade de procurar os *frozen spots*, pois estas partes mudam de aplicação para aplicação. A criação de objetos plugáveis é uma forma de encapsular os *hot spots* do *framework*. E, por último, os padrões sugerem o desenvolvimento de um construtor visual e ferramentas de linguagem para facilitar o uso do *framework*.

⁵ Segundo Coplien (1998) Linguagens de Padrões pode ser considerada uma coleção estruturada de padrões que se apoiam uns nos outros para transformar requisitos e restrições numa arquitetura.

3.3.3 Metodologia proposta por Bosch et al. (1999)

Bosch *et al.* (1999) afirmaram que o desenvolvimento de um *framework* é diferente de uma aplicação padrão, pois o projeto de um *framework* necessita cobrir todas as características pertinentes em um domínio. Baseando-se nos problemas experimentados e identificados durante o desenvolvimento de *frameworks* Bosch *et al.* (1999) criaram seis atividades para auxiliar no desenvolvimento de um *framework* simples:

- **Análise de domínio:** esta atividade tem como objetivo descrever o domínio que será coberto pelo *framework*, capturar seus requisitos e identificar os conceitos. Seu resultado consiste em um modelo de análise de domínio. Muitos desenvolvedores recorrem a aplicações que já foram desenvolvidas no domínio, ou a padrões existentes para este domínio;
- **Projeto arquitetural:** baseado no modelo de análise de domínio, cria-se o projeto arquitetural do *framework*;
- **Projeto do *framework*:** nessas atividades as classes são refinadas. O resultado desta atividade consiste na extensão da funcionalidade dada pelo projeto do *framework*;
- **Implementação do *framework*:** as classes abstratas e concretas são implementadas utilizando a linguagem de programação escolhida;
- **Teste do *framework*:** essa atividade tem como objetivo testar o *framework* a fim de verificar se possuem as funcionalidades planejadas e avaliar a sua usabilidade. Dependendo do tipo de aplicação, diferentes aspectos do *framework* podem ser avaliados. Baseando os testes nas aplicações desenvolvidas a partir do *framework*, pode-se verificar a necessidade de remodelar o *framework*;
- **Documentação:** esta atividade é uma das mais importantes no processo de desenvolvimento de um *framework*, pois se torna possível a sua utilização plena somente por meio da documentação completa, onde se descreve como usar o *framework*, um manual do usuário e um documento do projeto do *framework*.

3.3.4 Metodologia proposta por Pree (1999)

Pree (1999) propõe uma metodologia na qual a construção inicia-se com a definição do modelo de objetos específicos de uma aplicação e as outras atividades são repetidas sucessivamente até que o *framework* se torne satisfatório. Na Figura 21 são mostradas as etapas e a forma cíclica de refinamento do projeto.

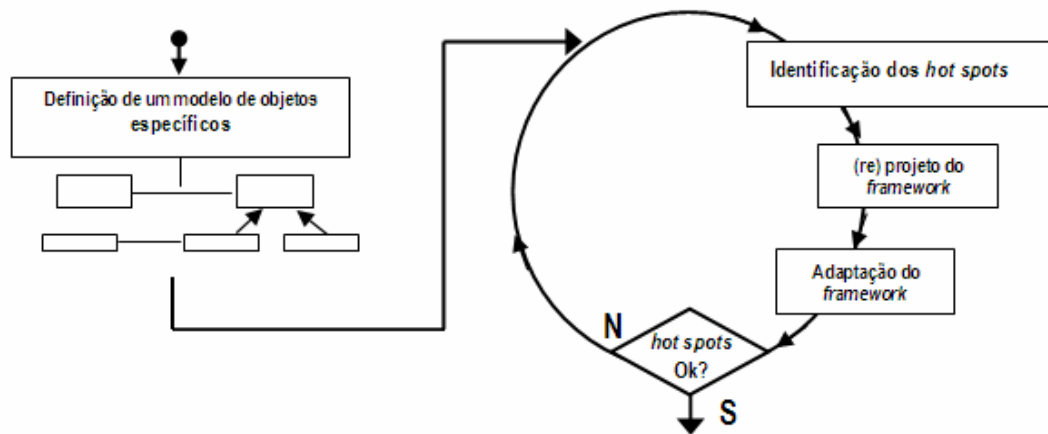


Figura 21 - Processo de desenvolvimento dirigido por *Hot spots* (Adaptada de PREE *et al.*, 1999)

Nessa metodologia há a necessidade de um desenvolvedor e de um especialista no domínio da aplicação e ambos serão co-responsáveis em todo o processo. As etapas são descritas a seguir:

- **Primeira etapa:** o especialista no domínio fornece todas as informações sobre o domínio para o desenvolvedor do *framework*, para que este defina a estrutura de classes.
- **Segunda etapa:** os especialistas no domínio ajudam os desenvolvedores de *framework* a identificarem os *hot spots*, que são documentados por meio de cartões de pontos variáveis (*hot spots cards*). Esses cartões têm como principal objetivo facilitar a comunicação entre especialistas de domínio e desenvolvedores, ou seja, padronizar a informação.
- **Terceira etapa:** após os especialistas no domínio identificarem e documentarem os *hot spots*, o desenvolvedor pode ter a necessidade de modificar o modelo de objeto para garantir a flexibilidade que os *hot spots* indicam. Nessa etapa o desenvolvedor utiliza os Padrões de Projetos. Os *hot*

spots são projetados e implementados e o *framework* é testado para verificar se os requisitos do domínio são satisfeitos.

- **Quarta etapa:** consiste em um refinamento da estrutura do *framework* a partir de novas sugestões dos especialistas no domínio. Alguns pontos variáveis podem se descobertos nesta etapa, levando à repetição da etapa de identificação de *hot spots*. Caso o *framework* foi avaliado como satisfatório, estará concluída uma versão do *framework*.

3.3.5 Metodologia proposta por Schmid (1997, 1999)

Schmid (1997, 1999) afirma que o mais importante é não iniciar o projeto de um *framework* tentando modelar sua variabilidade e flexibilidade. Ao invés disso, deve-se projetar uma aplicação fixa dentro do domínio do *framework* e, após o seu total entendimento, pode-se iniciar a generalização. Schmid (1999) criou quatro atividades para a construção de um *framework*. Na Figura 22 mostra-se o esquema dessas atividades.

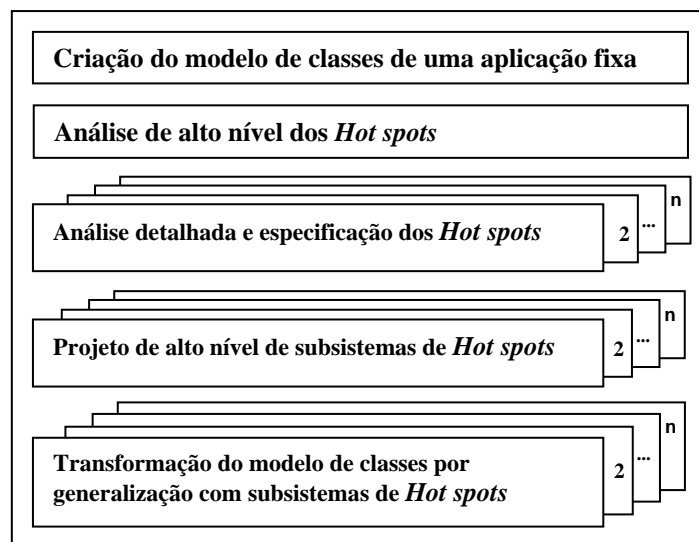


Figura 22 - Atividades para a construção de um *framework* (Adaptada de SCHMID *et al.*, 1999)

- **Análise de alto nível dos *hot spots*:** esta atividade identifica todos *hot spots* do domínio, descrevendo-os brevemente e colecionando-os em forma de cartões e com eles é realizada uma avaliação quanto a variabilidade e flexibilidade.

- **Análise detalhada e especificação dos *hot spots*:** as atividades de análise detalhada e de especificação são feitas separadamente para cada *hot spot*. Analisa-se um *hot spot* e descrevem-se suas características, com base no conhecimento do domínio e nos protótipos de aplicação. Se o resultado da análise de *hot spot* for um número muito grande de alternativas, deve-se concentrar nos *hot spots* elementares.
- **Projeto de alto nível de subsistemas de *hot spots*:** com as informações da atividade anterior são gerados vários subsistemas de *hot spots*. Esta atividade deriva um projeto baseando-se na especificação destes *hot spots*. Isto acontece separando este subsistema do resto da estrutura de classe do *framework*. Nesta atividade são utilizados Padrões de Projetos, pois estes provêm algumas vantagens, tais como: um nível de descrição mais abstrato, menos esforço no desenvolvimento, melhor comunicação de documentação.
- **Transformação do modelo de classes por generalização com subsistemas de *hot spots*:** essa atividade consiste em transformar a estrutura de classes do *framework* e generalizá-la, tendo como base uma única característica. Antes da generalização, a estrutura de classes contém normalmente esta característica como um *frozen spot* na forma de uma classe especializada ou de uma responsabilidade especializada como parte de uma classe, a qual possui relações diretas e fixas a outras classes. A transformação de generalização introduz a variabilidade e flexibilidade de um *hot spot* dentro da estrutura de classes.

3.4 Utilização de um *framework*

Utilizar um *framework* significa desenvolver aplicações a partir dele. Segundo Johnson (1997) e Fayad *et al.* (1999), há várias maneiras para utilizar um *framework*, sendo que para eles a maneira mais fácil é conectar componentes existentes. Isto não muda o *framework* ou cria qualquer nova subclasse concreta. Reusam-se as interfaces do *framework* e suas regras para conectar componentes. O desenvolvedor da aplicação só tem que saber que o objeto do tipo “A” é conectado ao objeto de tipo “B”, porém não precisa saber a especificação exata de “A” e “B”.

O usuário do *framework* é o desenvolvedor da aplicação a partir do *framework* e deve conhecer sua estrutura interna. O conhecimento profundo do *framework* é necessário para que toda a potencialidade de reuso seja aproveitada e esse nível de conhecimento varia de acordo com o tipo de *framework*.

No caso dos *frameworks* dos tipos caixa branca e caixa cinza, as classes que precisam ser conhecidas são as classes que devem ser geradas e as classes que podem ser geradas, pois estas herdaram algumas de suas características e as tornam adaptadas às exigências da nova aplicação. Sendo necessário diferenciar essas classes quais são as classes que são do *framework* e quais são as classes geradas a partir dele. Já nos *frameworks* do tipo caixa preta, além das classes concretas disponíveis é preciso também conhecer suas interfaces, relacionamentos e responsabilidades.

Para todos os tipos de *framework*, Silva (2000) levantou três questões-chave que devem ser respondidas pelo usuário do *framework*. São elas: Quais Classes do *framework* podem ser reutilizadas? Quais métodos podem ser reutilizados? O que os métodos fazem?

Primeira resposta: as classes concretas de uma aplicação podem ser criadas ou reutilizadas do *framework*. Então, que classes devem ser desenvolvidas pelo usuário e que classes concretas do *framework* podem ser reutilizadas?

Para a segunda resposta, Silva (2000) utiliza a classificação dos métodos de uma classe abstrata proposta por Johnson e Russo (1991). São eles: Método-Abstrato, que consiste em um método que tem apenas a sua assinatura definida na classe abstrata; Método-Gabarito que é definido em termos de outros métodos; Método-Base, que é completamente definido.

Ao produzir uma aplicação, os Métodos-Abstratos devem ser definidos; os Métodos-Gabaritos fornecem uma estrutura de algoritmo definida, mas permitem flexibilidade através dos Métodos-Ganchos. Os Métodos-Base, de acordo com o projeto do *framework*, não precisam ser alterados e ainda possuem a possibilidade de sobreposição.

Para a terceira resposta, Silva (2000) salienta que os métodos cujas estruturas devem ser definidas pelo usuário do *framework* produzem o comportamento específico da aplicação sob desenvolvimento. Assim, definidas as classes e os métodos a desenvolver, quais as responsabilidades destes métodos, em que situações de processamento eles atuam e como eles implementam a cooperação entre diferentes objetos. O autor afirma que aprender a usar um *framework* corresponde a buscar as respostas das questões-chave na estrutura do *framework*, usando como fontes de informação, a documentação, o código-fonte, ou ambos.

De acordo com Johnson (1993), o usuário deve compreender quais as aplicações que podem ser desenvolvidas a partir do *framework* e ser capaz de desenvolver aplicações básicas,

além de conhecer detalhadamente o projeto do *framework*. Sobre a utilização de um *framework*, Silva (2000, p. 75) afirma ainda que:

a motivação para produzir aplicações a partir de *frameworks* é a perspectiva de aumento de produtividade e qualidade, em função da reutilização promovida. Para produzir uma aplicação sob um *framework* deve-se estender sua estrutura, de modo a cumprir os requisitos da aplicação.

A reusabilidade pode ir além do reuso de linhas de código. Hoje o conceito de reuso é mais abrangente, pois é viável reutilizar projetos, análise de requisitos, código, documentação e até mesmo as decisões que foram tomadas diante de situações que aparecem com frequência no desenvolvimento de sistemas. O reuso de *software* tem como finalidade produzir sistema com qualidade e que possam ser reutilizados em parte ou modificados para atender as exigências de novos sistemas.

Os artefatos de *software* desenvolvidos com finalidade de reutilização, segundo Freiburger (2002) são: classe concreta, classe abstrata, Padrões de Projetos, *frameworks* e componentes. Segundo Fayad *et al.* (1997), o reuso de componentes de *framework* pode render melhorias significativas na produtividade de programador, aumentando qualidade, desempenho, confiança e interoperabilidade do *software*.

Como o *framework* é uma das formas de reuso de *software*, este deve ser bem documentado, para facilitar a sua utilização por outros desenvolvedores. A forma mais utilizada de documentação no caso de *frameworks* são os chamados *cookbooks*. Segundo Johnson (1997), esses documentos não são detalhados, porém possuem um roteiro bem definido que ajudam o usuário do *framework* na construção do projeto da aplicação.

Para auxiliar no desenvolvimento de aplicações baseadas em *framework*, Mattsson (1996) criou um conjunto de atividades que servem como guia. Bosch *et al.* (1999) desenvolveu Métodos-Baseados nas atividades de Mattsson. Essas atividades e métodos foram distribuídos no Quadro 1.

Descrição das atividades de Mattsson	Métodos de Bosch
Atividade 1: definição de um <i>framework</i> conceitual para a aplicação e isso se dá através da decomposição funcional e da análise preliminar das características da aplicação. Levantamento das associações entre os componentes e suas funcionalidades, relacionamentos e colaborações.	Análise dos requisitos: visa coletar e analisar as exigências da aplicação que deve ser construída usando um ou mais <i>frameworks</i> .
Atividade 2: seleção do <i>framework</i> para a aplicação. Baseando-se no levantamento da atividade anterior será selecionado um <i>framework</i> que atenda aos requisitos da aplicação.	Arquitetura Conceitual: Baseado nos requisitos coletados, uma arquitetura conceitual da aplicação é definida. Essa arquitetura possui as associações funcionais entre os componentes, relacionamentos e colaborações.
Atividade 3: mapeamento do <i>framework</i> conceitual para o <i>framework</i> selecionado.	Seleção do <i>framework</i>: com base na arquitetura da fase anterior, um ou mais <i>frameworks</i> são selecionados, levando em consideração as funcionalidades que devem ser oferecidas à aplicação.
Atividade 4: especificação ou revisão do relacionamento e colaboração entre os subsistemas do <i>framework</i> . Essas relações são pré-definidas pelo <i>framework</i> podendo ser revisadas ou manter-se sem alterações.	Seleção de reuso ou extensão: após decidir qual <i>framework</i> será usado, é decidido quais partes do <i>framework</i> serão usadas e quais serão definidas pela aplicação.
Atividade 5: estruturação dos <i>subframeworks</i> . Após o mapeamento das funcionalidades do <i>framework</i> conceitual, cada componente deve ser associado a um <i>subframework</i> ou a um componente ou ainda a um padrão e projeto inserido no <i>framework</i> .	Desenvolvimento das extensões: decididas às partes que serão desenvolvidas, essas serão especificadas e implementadas. Nessa fase as regras estabelecidas na documentação do <i>framework</i> devem ser obedecidas.
Atividade 6: estruturação dos componentes dos <i>subframeworks</i> . Nesse caso a atenção é voltada para os <i>hot spots</i> e <i>frozen spots</i> .	Teste das extensões: teste das partes desenvolvidas individualmente.
Atividade 7: implementação da aplicação.	Desenvolvimento da aplicação: elaboração do código-fonte de acordo com a tecnologia escolhida.

Quadro 1 - Comparativo entre as atividades de Mattsson (1996) e os métodos de Bosch et al. (1999)

3.5 Vantagens e Desvantagens de um *framework*

Como já foi dito nas seções anteriores, um *framework* possui características das linguagens de programação orientadas a objetos: herança, polimorfismo e abstração de dados. Essas características facilitam a reusabilidade e, com certeza, esta é uma das vantagens que chama mais a atenção dos pesquisadores. Se o *framework* já estiver pronto, há a maximização do reuso, ou seja, reutiliza-se, em geral, análise, projeto, código e testes e, com isso, acontece uma redução significativa do tempo de desenvolvimento de novas aplicações. As outras vantagens apontadas são:

- a diminuição das linhas de código;
- possibilidade de utilizar outras técnicas de reuso em conjunto, por exemplo, Padrões de Projetos (*design patterns*) e componentes;
- aumento da qualidade de *software*, facilidade na manutenção, pois quando um erro é corrigido no *framework*, automaticamente, ele é corrigido nas aplicações desenvolvidas a partir deste *framework*;
- diminuição dos erros no código já que ele é usado em várias aplicações.
- empacotamento do conhecimento dos especialistas sobre o domínio do problema, evitando a dependência de um único especialista. Os desenvolvedores ficam garantidos caso algum especialista ou pesquisador saia do projeto.

Algumas desvantagens apontadas por Mattsson (1996) e outros pesquisadores são:

- dificuldade em desenvolver um *framework*;
- se o *framework* não possuir uma documentação apropriada, certamente ele não será bem utilizado;
- o processo de depuração pode ser complicado porque é difícil distinguir quando o erro é do *framework* ou da aplicação. Caso o erro esteja no *framework* pode ser impossível o usuário conseguir corrigi-lo;
- a generalidade e flexibilidade do *framework* podem atrapalhar sua eficiência em uma aplicação particular.

Para Fayad *et al.* (1997), as desvantagens para utilizar um *framework* são: a dificuldade para entendê-lo; o esforço demandado para desenvolver e também utilizar um *framework*; dependência do desenvolvedor do *framework*, pois têm características que somente o desenvolvedor conhece e a inversão de controle e a falta de fluxo de controle explícito podem dificultar a depuração de alguns erros.

Fayad *et al.* (1997), assim como Mattsson (1996), também salientaram a dificuldade de identificar a localização dos *bugs*. As desvantagens apontadas por esses dois pesquisadores na verdade são complementares, quando Mattsson (1996) afirma a falta de documentação como desvantagens, Fayad *et al.* (1997) apontam a dificuldade de entendimento do *framework*, sendo esta uma característica de um *framework* com pouca documentação. Portanto, pode-se concluir que estas desvantagens são falhas no desenvolvimento e podem ser evitadas.

3.6 Considerações Finais

Este capítulo apresentou o conceito de *framework* sob a ótica de vários autores. Com isso, criou-se a base teórica para o desenvolvimento do projeto proposto. As formas de classificação de *frameworks* foram apresentadas. Juntamente com algumas das metodologias de construção existentes, descritas com intuito de apresentar as várias possibilidades de implementação. A utilização de um *framework* é tão importante quanto a sua construção e por isso, foram descritos algumas atividades e métodos que auxiliam e otimizam a sua utilização. Foram apresentadas, ainda, as vantagens e desvantagens na construção e utilização de um *framework*. No próximo capítulo é apresentado o panorama atual da utilização de *frameworks* nas diversas áreas, dando ênfase a *frameworks* construídos para aplicações de RV. Além disso, é feita uma análise dos *frameworks* estudados, destacando-se as suas principais características.

4 FRAMEWORKS DE REALIDADE VIRTUAL

A utilização de *frameworks* ocorre nas mais diversas áreas de pesquisa, em temas ligados à área de negócios, entretenimento e treinamento. Neste capítulo serão apresentados alguns *frameworks* utilizados em aplicações de Realidade Virtual, a fim de dar um embasamento científico ao projeto, a partir do conhecimento do estado da arte na área.

Os *frameworks* de RV são conhecidos por sua complexidade, dificuldade de instanciação e por serem utilizados apenas por pesquisadores ou programadores experientes. Segundo Bastos *et al.* (2004), os *frameworks* de RV oferecem *design* e construtos reutilizáveis para o desenvolvimento de aplicações de RV e solucionam problemas de como lidar com dispositivos de entrada e saída ou renderização para diferentes sistemas de projeção.

Bastos *et al.* (2005) afirmam que esses *frameworks* permitem que o usuário se concentre no desenvolvimento da aplicação, pois não é necessário se preocupar com a gerência do sistema de RV. Citam, ainda, que tais *frameworks* podem oferecer: abstração de dispositivos, abstração de sistema de projeção, grafos de cena⁶ especializados, heurísticas de interação com o AV, suporte a sistemas distribuídos e renderização distribuída. Nas próximas seções serão apresentadas as principais características de *frameworks* citados na literatura.

4.1 Avango

O Avango consiste em um *framework* orientado a objetos que permite a criação de aplicações com classes específicas que herdam propriedades de distribuição. Como sua extensão é dada por meio de herança, pode-se classificá-lo como um *framework* caixa branca (*White box*). Tramberend (2001) afirma que o *framework* Avango foi projetado para auxiliar o desenvolvimento de aplicações distribuídas de AV interativos.

A distribuição de dados é alcançada por replicação transparente de um grafo de cena compartilhado entre os processos participantes de uma aplicação distribuída, utilizando um sistema de comunicação para garantir o estado de consistência. Cada processo possui uma

⁶ Grafo de Cena - Um grafo de cena é uma estrutura que guarda a informação sobre a cena, inclusive os dados geométricos, a definição da aparência e os parâmetros de visão, entre outras coisas (SUN, 2006b).

cópia local do grafo de cena e as informações de seus estados são mantidas sincronizadas, proporcionando aos programadores o conceito de um grafo de cena compartilhado.

Foi utilizada a linguagem de programação C++ para desenvolver duas categorias de classes: *Nodes* e *Sensors*. A classe *Nodes* fornece um grafo de cena que permite a representação e a renderização de uma geometria complexa e a classe *Sensors* proporciona ao Avango uma interface com o mundo real, importando dados de dispositivos externos à aplicação. Além de C++ também foi utilizada a linguagem *Scheme* (derivada do *Algol* e *Lisp*). Segundo Tramberend (2001), todos os objetos de alto nível do Avango podem ser implementados e manipulados com a linguagem *Scheme*.

4.2 SSVE – Shared Simple Virtual Environment

O *framework* SSVE foi desenvolvido sob o paradigma de orientação a objetos e projetado para pequenos grupos colaborativos em ambientes virtuais altamente dinâmicos, compartilhados e interativos. Utilizou a linguagem de programação C++ e CORBA (*Common Object Request Broker Architecture*) como mecanismo de comunicação interprocessos. Sua construção ocorreu a partir do SVE (*Simple Virtual Environment*), aplicação monousuária que serve como banco de dados do grafo de cena e renderizador de objetos.

Linebarger *et al.* (2003) afirmam que o SSVE é específico para aplicações multiusuárias de RV, sendo propício para tarefas executadas por grupos compostos com até oito usuários. O SSVE possui acesso exclusivo aos objetos e possibilidade de fornecer a visão que outro participante tem do AV.

A arquitetura de SSVE utiliza *thread*, é portátil, orientada a objetos e *peer-to-peer*. Para desenvolver uma aplicação de SSVE são criadas classes concretas em C++ que herdam e instanciam o núcleo do SSVE. Um objeto do tipo *singleton*, proposto por Gamma *et al.* (1995) é instanciado dentro da aplicação para mediar a comunicação.

4.3 IVORY

Sprenger *et al.* (1998) descrevem o *framework* IVORY como sendo orientado a objetos e desenvolvido para visualização de informação baseada em física (força, massa, aceleração, potência, energia, entre outros), com classes básicas implementadas com a linguagem de programação Java. Para a produção gráfica foi utilizada a linguagem VRML 2.0 que provê um alto nível de descrição da cena. Para controlar os *plugins*⁷ foi utilizada a linguagem IVML (*Information Visualization Modeling Language*).

O IVORY pode ser estendido por meio de herança, categorizando-o como *framework* caixa branca. A estrutura do IVORY possui uma divisão horizontal e outra vertical. Horizontalmente pode-se notar o conceito de *frontend/backend*. Verticalmente são separados os componentes com dados dependentes e os componentes com dados independentes.

O *backend* suporta dois modos de execução: no primeiro modo a execução é diretamente relacionada ao *frontend* e pode ser executado na mesma máquina ou, ainda, em diferentes máquinas sendo gerenciado pelo servidor da aplicação. No segundo caso, o *frontend* e o *backend* comunicam-se através do *socket* de conexão da linguagem Java. Todos objetos instanciados (*plugins*) são armazenados, otimizando os processos de inserção, remoção e consulta dos mesmos, como é possível observar na Figura 23 .

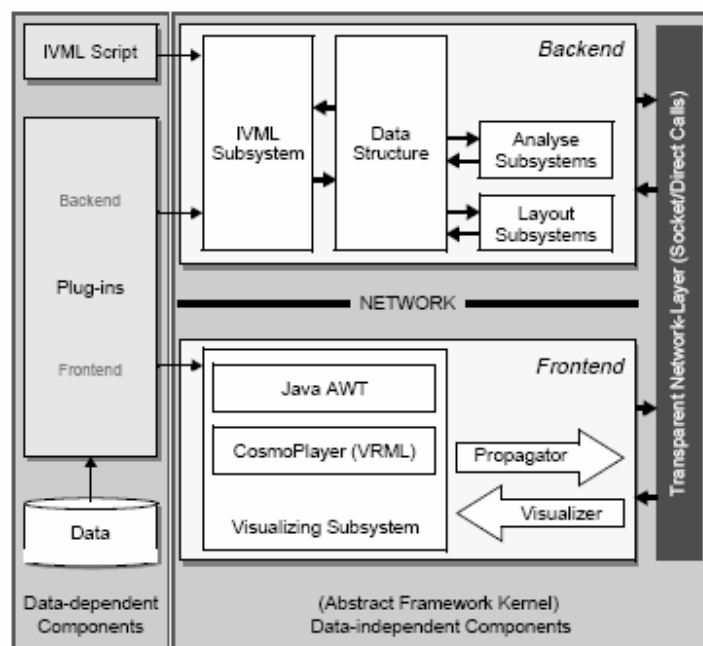


Figura 23 - Esquema dos componentes do IVORY (SPRENGER *et al.*, 1998)

⁷ Plugins: são programas que funcionam anexados a outros programas e suas características torna possível visualizar vários tipos de arquivos (CONTI, 2005)

A estrutura é inicializada pelo subsistema de *IVML* que é responsável por instanciar dinamicamente um objeto de acordo com análise do *script* de configuração. Este subsistema pode carregar e unir objetos desconhecidos através dos *plugins* em tempo de execução. Existem dois subsistemas: o *Layout subsystems*, que tem acesso direto à estrutura de dados do *Kernel* e o *Analyse subsystems*. O segundo tipo do subsistema é fornecido pelo componente da análise que é composto por filtros de seleção e algoritmos de *cluster*.

O *frontend* é dividido em três partes: a primeira consiste em um subsistema de visualização responsável por todas as saídas visuais do sistema; a segunda é um gráfico 2D que é usado como GUI's (*Graphical User Interface*), com todas as entradas e saídas padrões, caixa de diálogo e menus necessários, como pode ser visto do lado esquerdo da Figura 24; e a terceira consiste em uma janela de visualização 3D, que permite ao usuário navegar e interagir com o objeto carregado, como é mostrado no lado direito da Figura 24.

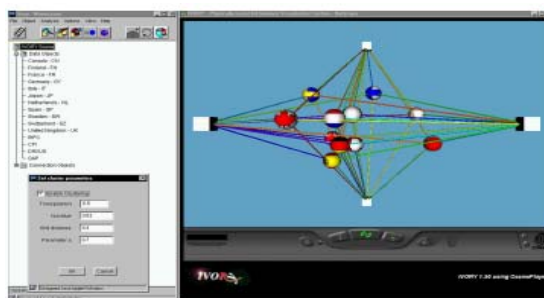


Figura 24 – O IVORY frontend sendo executado em uma máquina com Windows NT4.0 com browser Netscape (SPRENGER et al., 1998)

Ligado ao *frontend* existem dois componentes responsáveis pela troca de mensagens, o *Propagator*: responsável por informar ao *backend* se uma interação com o usuário é válida e se possui integridade com a estrutura de dados. Caso haja alguma alteração o *backend* calcula os novos parâmetros e sincroniza com o *frontend* e o *Visualizer*: que é executado de maneira semelhante, mas na direção oposta da estrutura de dados do *backend* para a visualização do *frontend*.

Os ambientes de visualização de informações baseados em física provêm uma interface de homem-computador natural e oferecem uma sensação de tato ao usuário. Com o *Phanton* torna-se possível escolher, puxar ou empurrar objetos 3D individuais. Assim pode-se sentir virtualmente a força de conexão de objetos em um determinado plano do gráfico.

Em um exemplo o *IVORY* foi utilizado para representar graficamente a influência de quatro indicadores econômicos sobre as taxas de juros em longos períodos de tempo em países diferentes. Na Figura 25 (a) e (b) podem ser observados, respectivamente um exemplo com a utilização de um dispositivo háptico e outro sem ele.

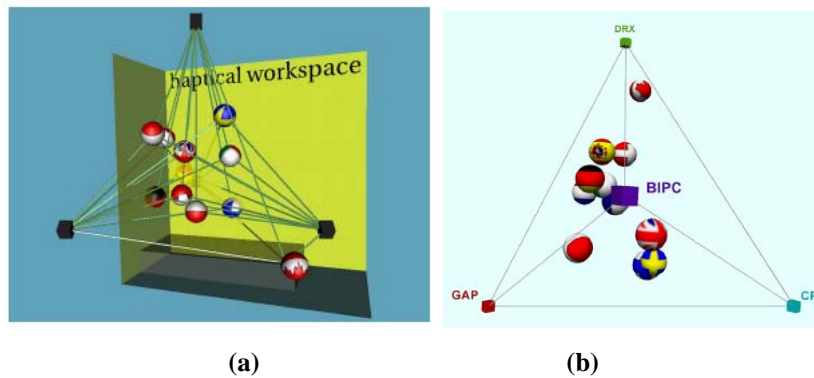


Figura 25 – (a) Exemplo de aplicação utilizando dispositivos de *force-feedback*. (SPRENGER *et al.*, 1998), (b) Aplicação sem o dispositivo de *force-feedback*. (GROSS *et al.*, 1997)

4.4 basho - Virtual Environment Framework

Segundo Hinkejann *et al.* (2004), o *basho* consiste em um *framework* para criação de AV que apóia diferentes renderizadores e possui um núcleo pequeno com facilidade de controle e manutenção. Possui também um renderizador de grafo de cena, baseado em *OpenSceneGraph*, proposto por Osfield (2004).

O *framework basho* pode fundir os resultados dos diferentes renderizadores baseando-se na cor e profundidade da informação. Projeta uma cena atual baseando-se em informações específicas, como: geometria, sons, força, entre outras. As principais características do *framework basho* são: suporte a renderizadores múltiplos, suporte de renderizadores genérico, núcleo pequeno, objetos genéricos, amplo suporte a dispositivos de entrada e saída. Na Figura 26 pode ser observada a estrutura do *basho*.

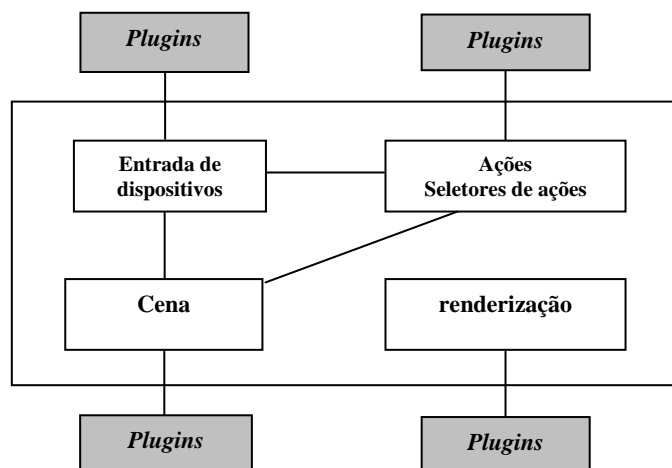


Figura 26-Kernel do *basho* e seus *plugins*. (Adaptada de HINKENJANN *et al.*, 2004)

4.5 ViRAL (Virtual Reality Abstraction Layer)

O ViRAL é um *framework* gráfico, baseado em componentes, com arquitetura caixa cinza desenvolvido com a linguagem de programação C++ e independente de plataforma. Windows é sua plataforma principal, mas já foi testado em Linux e IRIX. Segundo Bastos *et al.* (2004), o projeto ViRAL surgiu com o objetivo de sanar a carência de um *framework* de RV pequeno e simples cuja utilização fosse acessível em sistema de RV de baixo custo e, ainda, que pudesse ser utilizado por pessoal não especializado em computadores pessoais. O ViRAL está sendo utilizado para construir desde pequenos jogos até complexos visualizadores de modelos CAD (*Computer Aided Design*).

Bastos *et al.* (2004) afirmaram que o ViRAL pode ser visto como uma camada interposta entre as aplicações e os sistemas de RV e que tem como objetivo principal facilitar o desenvolvimento de aplicações de RV que sejam operadas por interfaces WIMP (*Windows, Icons, Menus and Pointing Device*).

Bastos *et al.* (2005) explicam que as aplicações são construídas instanciando-se e configurando-se componentes e podem simular vários mundos virtuais concorrentes, sendo capazes de salvar e restaurar os ambientes de RV usando arquivos do tipo XML. No projeto do ViRAL, os dispositivos de entrada, observadores e ambientes virtuais são todos componentes, que podem ser carregados de bibliotecas e integrados em GUI's. O *framework* é portátil para diversos ambientes e pode ser utilizado de duas maneiras: como um conjunto de componentes embutidos em uma aplicação hospedeira ou como uma aplicação autônoma e extensível, como se observa na Figura 27.

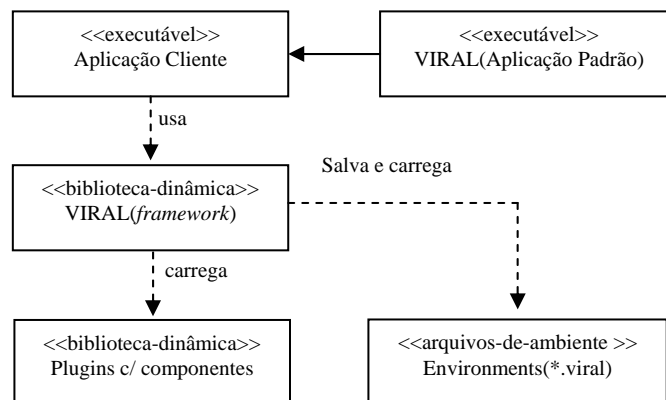


Figura 27- Diagrama de instalação (BASTOS *et al.*, 2004)

A arquitetura do *framework* ViRAL é descrita por uma estrutura hierárquica que contém todos os seus objetos e componentes centrais. Esta estrutura é disposta na forma de uma árvore heterogênea, como mostrado na Figura 28, tendo como raiz o *RootSystem*, que é o padrão *Singleton* criado por Gamma *et al.* (1995). No nível abaixo estão todos os objetos do tipo *System*⁸ que agregam funcionalidades ao *framework*.

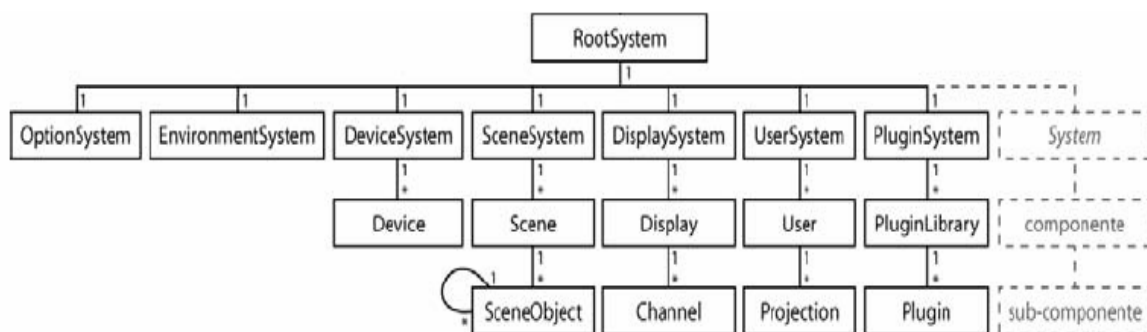


Figura 28 - Uma árvore do ViRAL com os sistemas de padrões (BASTOS *et al.*, 2004).

As principais características dos sistemas padrões do ViRAL são descritas a seguir.

- **Devices** (Sistema de Dispositivo) – este componente tem como funções gerar, processar ou transformar eventos, tratados por *signal/slots*. Por meio dos *devices* pode-se representar, dentro do ViRAL, qualquer tipo de dispositivos reais, tais como: mouse 3D, luva, dispositivos com *force feedback* e rastreadores.
- **Scenes** (Sistema de Cenas) – os componentes do tipo *Scenes* são responsáveis pela simulação de ambientes virtuais imersivos (AVI). Outras funções dele são testar a colisão e produzir som 3D. A interface utiliza o padrão de projeto Método-Gabarito que é usado para desenhar o mundo virtual.
- **Users & Displays** (Sistema de usuários e de exposição) – representa um usuário imerso em um AV. Suas principais propriedades são: a distância interocular, sua *Scene* e sua posição e orientação dentro da *Scene*. Para a *movimentação* dentro de uma cena, um *User* utiliza quatro *slots*, sendo que dois definem a posição e orientação atuais e os outros definem a translação e rotação localmente. Esses *slots* ainda permitem que um *User* se comunique diretamente com *Devices*. Os *Displays* (visores) representam contextos de renderização cujos quadros são expostos em

⁸ Os objetos do tipo *System* são componentes quando utilizam o modelo de *signal/slots* proposto por Trolltech (2004).

uma tela. Segundo Bastos *et al.* (2005), o componente *Display* representa também um *canvas* do OpenGL associado a uma saída de vídeo ou tela de projeção e em ambientes *desktop* é visto como uma janela.

- ***Environments (Sistemas de Persistência para AV)*** – o objetivo deste sistema é fornecer persistência para ambientes virtuais, ou seja, tornar as árvores persistentes. Somente um objeto *Environment* pode estar ativo numa aplicação em um dado momento.
- ***Plugins (Sistema de Plugins)*** – os objetos do tipo *Plugin* servem para instanciar alguns tipos de objetos, como *Scenes (ScenePlugins)* e *Devices (DevicesPlugins)*. Para dispositivos de entrada já foram criados *plugins* para mouse 3D, luvas, dispositivos de rastreamento e um rastreador óptico.

4.6 VPat (*Virtual Patients*)

VPat é um *framework* orientado a objetos que possui classes básicas com finalidades que podem ser compartilhadas e estendidas e, assim, permitir o desenvolvimento de classes mais especializadas que sejam capazes de implementar algoritmos complexos de visualização e simulação de movimento e, ainda, suportar aplicações de RV.

Foi desenvolvido com o propósito de suportar aplicações de computação gráfica na área médica e possui facilidade de extensão, podendo ser classificado como um *framework* caixa branca. O *framework* foi utilizado como base para uma ferramenta de visualização direta de imagens médicas e para modelagem de corpos articulados.

A utilização mais recente do VPat foi uma extensão para uma aplicação de laparoscopia virtual. As classes básicas foram implementadas em C++, tornando-o independente de plataforma. Segundo Freitas *et al.* (2003), o VPat utiliza o padrão MVC (*Model-View-Controller*) garantindo que o núcleo funcional do sistema seja independente da interface. Para a aplicação de laparoscopia virtual foi projetada uma interface específica para simulações de procedimentos cirúrgicos laparoscópicos em RV. Para essa aplicação, além das classes básicas do VPat, foram implementadas classes específicas que ajudam a construir um AV com todas as características de uma laparoscopia real, conforme Figura 29.

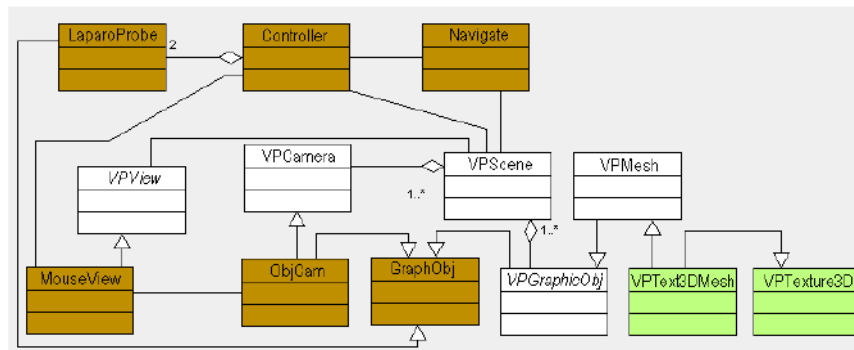


Figura 29 - Diagrama de classes aplicação de Laparoscopia Virtual (FREITAS *et al.*, 2003)

A classe *Controller*, segundo Freitas *et al.* (2003), pode ser considerada o elo entre as ações feitas a partir da interface com o usuário e o modelo definido pela classe *VPScene* (classe básica do *framework*). *GraphObj* herda características da classe *VPGraphicObj* e incorpora ao objeto um sistema de referência local e uma posição. A classe *GraphObj* simula a sonda laparoscópica através da classe *LaparoProbe*. A classe *Navigate* implementa o mecanismo de detecção de colisão. A classe *MouseView* é responsável por atender aos eventos do mouse. E a classe *ObjCam* é encarregada pela representação visual.

Além das classes implementadas para a aplicação de laparoscopia, o VPat utiliza uma interface para laparoscopia virtual da empresa *Immersion Corporation*, específica para simulações de procedimentos cirúrgicos laparoscópicos em RV, como é mostrado na Figura 30. Segundo Freitas *et al.* (2003), o equipamento pode ser visto como um instrumento cirúrgico que se comunica com um computador por meio de uma porta serial e pode ser usado para simular uma grande variedade de ferramentas reais de manipulação cirúrgica.



Figura 30 - Interface de Laparoscopia Virtual (FREITAS *et al.*, 2003)

4.7 Comparação entre os frameworks de RV

Dentre todos os *frameworks* estudados, foi realizada também uma análise das características que auxiliaram no desenvolvimento do *framework ViMeT*. Três destes apresentaram aspectos importantes o ViRAL, VPat e *IVORY*. Seus aspectos comuns indicam que os três são orientados a objetos e multi-plataforma.

Do ViRAL o conceito de *plugins* é uma característica interessante, pois proporciona facilidades para incorporar dispositivos de entrada e saída à aplicação. Outro aspecto importante é a forma de extensão (caixa cinza). O VPat mostrou que tem a capacidade de utilizar um sistema de interface já implementado. O *framework IVORY* tem como característica comum ao presente trabalho a utilização da linguagem de programação Java e a proposta de utilizar também sua *API (Application Programming Interface)* Java3D, além da conexão a um Sistema Gerenciador de Banco de Dados.

Nenhum dos *frameworks* citados utilizou uma ferramenta de instanciação. No ViRAL existe uma interface que facilita a inclusão e exclusão de dispositivos e no VPat foi criada uma interface específica para a aplicação gerada a partir dele.

Para auxiliar a utilização de um *framework* é necessário que haja uma documentação, que pode ser um manual de instanciação, *cookbook*, exemplos, entre outros. Os *frameworks* de RV estudados não possuíam documentação disponível.

Como o *ViMeT* possui um domínio limitado a procedimentos de biópsia, torna-se interessante construir uma ferramenta de instanciação, pois pode ser mantida a mesma interface para todas as aplicações, sendo que as mudanças necessárias para atender especificações de novas aplicações ficam concentradas na instanciação do *framework*. Os *frameworks* estudados possuem domínio diferentes do domínio do *ViMeT* e por esse motivo optou-se por desenvolver um novo *framework* ao invés de reutilizar um deles.

A partir das principais características dos *frameworks* descritos anteriormente, foi composto o Quadro 2 a fim de permitir uma comparação entre eles.

Frame-works	Características avaliadas							
	Tipo de extensão	Utiliza Padrões	Utiliza Componentes	Linguagem	SGBD	Plataforma	Domínio específico	Documentação
Avango	Caixa branca	Não	Não	C++ e Scheme	Não	Linux e IRIX	Desenvolvimento de Aplicações Distribuídas	Não
SSVE	Caixa branca	Não	Não	C++	Não	Windows, IRIX, Linux, e Solaris	Aplicações para Ambientes colaborativos e AV dinâmicos	Não
IVORY	Caixa branca	Não	Sim	Java	SQL, DB2 e DBase	Multi plataforma	Visualização de Aplicações baseados em física	Não
basho	Caixa branca	Não	Não	C++	Não	Linux e Mac OS X	Aplicações com vários renderizadores	Não
ViRAL	Caixa cinza	Sim	Sim	C++	Não	Windows, Linux e IRIX	Jogos e visualizadores de CAD	Não
VPat	Caixa branca	Sim	Não	C++	Não	Multi Plataforma	Laparoscopia	Não
ViMeT	Caixa Cinza	Não	Não	Java e API Java 3D	Derby	Multi Plataforma	Exames de punção	Sim

Quadro 2 - Comparativos dos *frameworks* estudados

4.8 Considerações Finais

Este capítulo apresentou alguns *frameworks* de RV existentes e foram analisadas suas principais características, tais como: a linguagem de programação utilizada, forma de extensão, utilização de Padrões de Projetos, conexão a bancos de dados e plataformas necessárias. Este estudo permitiu traçar o plano de trabalho para a construção do *ViMeT*.

No próximo capítulo é descrito o desenvolvimento do *ViMeT* detalhando a metodologia utilizada.

5 DESENVOLVIMENTO DO *ViMeT*

Como foi apresentado na Seção 3.3, existem várias metodologias que auxiliam na construção de um *framework*. Para o desenvolvimento do *ViMeT* foi adotada a metodologia proposta por Bosch *et al.* (1999), visto que possui atividades parecidas com as exigidas no desenvolvimento de uma aplicação orientada a objetos quaisquer, possui uma fase de testes, além de se adequar ao tempo disponível para a o desenvolvimento do *ViMeT*. Neste capítulo está descrita detalhadamente cada fase desta metodologia.

5.1 Contextualização

O LApIS (Laboratório de Aplicações de Informática em Saúde) possui como um dos objetivos a construção de ferramentas computacionais de baixo custo, desenvolvidas com tecnologia livre. Desta maneira, além de integrar as aplicações já desenvolvidas foi possível estabelecer um padrão para o desenvolvimento de novas técnicas e, ainda, garantir a facilidade de desenvolvimento de novas aplicações.

Como já apresentado, membros da equipe desenvolveram diversas aplicações de RV relacionadas ao treinamento médico. No contexto deste trabalho destacam-se as aplicações de detecção de colisão com precisão (KERA, 2005), estereoscopia (BOTEGA, 2005) e deformação (PAVARINI, 2006). Percebendo que tais aplicações não eram suficientemente genéricas para serem reaproveitadas para a construção de novas ferramentas, decidiu-se reuni-las em um *framework*.

5.2 Tecnologias Utilizadas

Para a implementação do *ViMeT* foram utilizadas a linguagem de programação Java (versão JDK 1.5.0_06) (SUN, 2006a) e a API Java3D (versão 1.3.1) (SUN, 2006b). Suas principais características são descritas a seguir.

A escolha da linguagem Java e da *API* Java3D é devido ao fato da gratuidade uma vez que, como citado anteriormente, podem ser utilizadas em locais onde recursos financeiros são escassos. Outra razão é que a *API* Java3D possui várias classes que auxiliam na implementação de AV, interação e visualização tridimensional.

Segundo Deitel e Deitel (2005), a linguagem Java possui muitas vantagens e toda segurança semelhante a uma linguagem compilada. Não possui ponteiros, tem a coleta automática de lixo, é multitarefa e permite o controle de *threads*. Oferece grande suporte para aplicações de pequeno, médio e grande porte para Internet e Intranet.

A documentação de Java, fornecida pela SUN (2006a), afirma que devido às características de orientação a objetos, sua interface contribui para reutilização de código, utilização de bibliotecas de terceiros com proteção e encapsulamento e, além disso, oferece robustez e segurança.

Java3D é uma *API* desenvolvida pela *Sun Microsystems* (SUN, 2006b), para renderizar gráficos interativos 3D usando a linguagem de programação Java. A *API* Java 3D oferece um conjunto de classes que permite o desenvolvimento de aplicações 3D em alto nível, utilizando-se de recursos como criação e manipulação de geometrias 3D, animações e, ainda, interatividade com dispositivos convencionais e não convencionais. São definidas mais de 100 classes apresentadas no pacote *javax.media.j3d*., possibilitando que o desenvolvimento de aplicações 3D seja totalmente orientado a objetos.

A *API* Java3D constrói AVs usando o conceito de grafo de cena, uma estrutura hierárquica organizadora dos objetos que compõem e definem uma cena, preservando as relações espaciais existentes entre eles. Tais objetos referem-se tanto às geometrias e suas propriedades, como às informações necessárias para que a cena seja renderizada de um ponto particular de visão (SUN, 2006b).

O grafo de cena é composto por objetos das classes *VirtualUniverse*, *Locale SceneGraphObject*, *Node*, *NodeComponent*, *Group*, *Leaf*, *BranchGroup*, *TransfomGroup*, *Shape3D*, *Light*, *Behavior*, *Sound*, *Geometry*, *Appearance*, *Material*, *Texture* entre outras. Um exemplo de grafo de cena é mostrado na Figura 31e sua legenda é apresentada no Quadro 3.

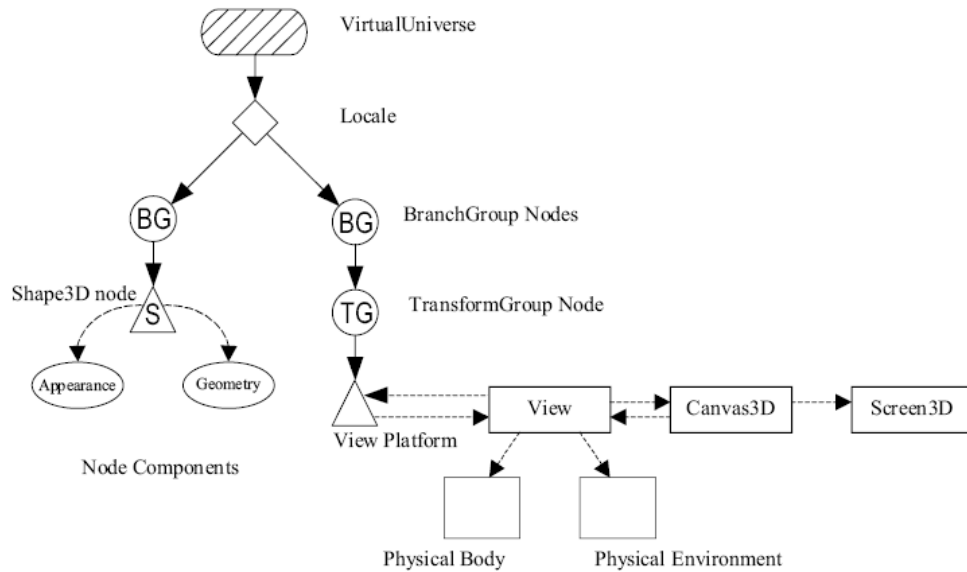


Figura 31 - Exemplo de um grafo de cena (SUN, 2006b)

Símbolo	Nome	Finalidade
	VirtualUniverse	Representa o universo virtual
	Locale	Define uma área geográfica com tamanho e localização no universo Virtual. Por <i>default</i> (0, 0, 0)
	Group	Organiza os dados, controla a ordem de renderização e mudar a posição, orientação e tamanho dos objetos visíveis no AV.
	Leaf	Representa os objetos que estarão na cena gráfica.
	NodeComponent	Especifica as propriedades de um determinado objeto na cena gráfica.
	Outros objetos	Representa outras classes de objetos.
Símbolo	Finalidade	
	Pai e filho	
	Referência	

Quadro 3 – Legenda do Grafo de Cena (SUN, 2006b)

Segundo a SUN (2006b), com esses mecanismos, o programador pode descrever mundos virtuais grandes e com riqueza de detalhes e os programas escritos podem interagir com gráficos tridimensionais. Para a construção de AVs, a API Java3D possui três opções: *Simple Universe*, *Configured Universe* e *Virtual Universe*.

O modelo de visualização da *API Java3D* emula uma câmera virtual e a aplicação pode renderizar imagens em vários dispositivos de visualização sem a necessidade de reprogramação. Essa facilidade existe porque o modelo de visualização *Java3D* separa o ambiente virtual do ambiente físico (SUN, 2006b).

5.3 Metodologia empregada

Como já descrito na Seção 3.3.3 com base nos problemas experimentados e identificados durante o desenvolvimento de *frameworks*, Bosch *et al.* (1999) criou seis atividades para auxiliarem no desenvolvimento de um *framework* simples. A seguir é detalhada cada uma delas de forma adaptada ao presente projeto. Na Figura 32 mostra-se o fluxograma das atividades.

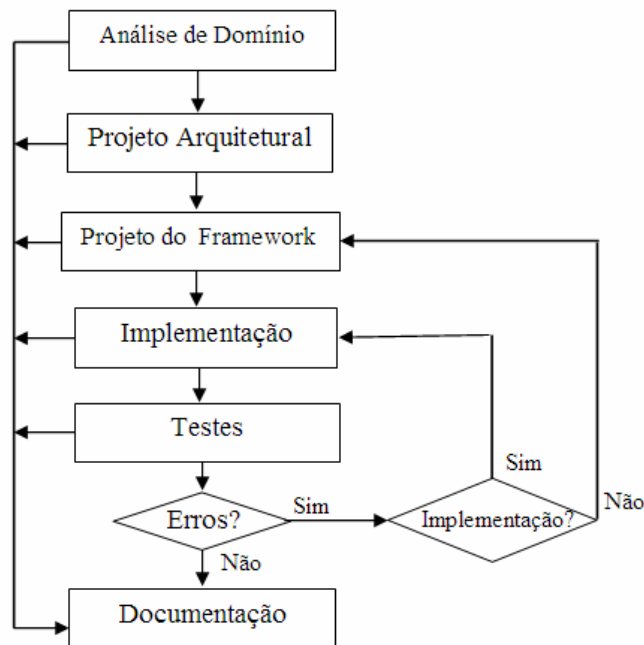


Figura 32 - Fluxograma das atividades do *ViMeT*

Como esta metodologia é iterativa, o resultado final pode ser diferente do inicial. Essa iteração prevê que o desenvolvimento pode sofrer mudanças em qualquer uma das fases.

5.3.1 Análise de Domínio

Segundo Bosch *et al.*(1999), esta atividade tem como objetivo descrever o domínio que é coberto pelo *framework*, capturar seus requisitos e identificar os conceitos. Seu resultado consiste em um modelo de análise de domínio, onde as funcionalidades que o *framework* possuirá serão apontadas. O domínio do presente projeto é o treinamento médico virtual, especificamente os exames de punção.

A análise de domínio do *ViMeT* foi dividida em duas sub-atividades: a primeira foi o levantamento do estado da arte de aplicações de RV para treinamento médico e a segunda consiste em uma pesquisa sobre os procedimentos comuns existentes nos exames de punção.

O levantamento do estado da arte de diversas aplicações de RV para treinamento médico, como pôde ser verificado na Seção 2.1, teve como objetivo mostrar quais os requisitos necessários para a construção de uma aplicação genérica de RV para treinamento médico. Desta ação resultaram as seguintes características: detecção de colisão, deformação, utilização de dispositivos hápticos, interface gráfica com usuário, utilização de objetos modelados tridimensionalmente, ambientes virtuais dinâmicos, conexão com SGBDs e estereoscopia.

Durante a pesquisa sobre os detalhes dos exames de punção, a fim de complementar a fase de análise de domínio, foram analisados os procedimentos necessários para executar esta categoria de exames, apresentados a seguir.

Quando há suspeita de anomalias em pacientes, os médicos, em geral, solicitam exames adicionais para confirmar um diagnóstico. Um desses exames pode ser o exame de punção, também conhecido como biópsia. Consiste, basicamente, em extrair pequenas partes de tecidos do órgão, sobre o qual paira a suspeita de anomalia. Estas partes são enviadas a patologistas que auxiliam na determinação do diagnóstico. Existem exames de punção para diversos órgãos, bem como diversas técnicas para realizá-los.

Segundo informações da Diagnóstico da América (2006), uma das técnicas mais utilizadas é o exame de Punção Aspirativa por Agulha Fina (PAAF), a qual pode ser executada em consultórios ou em pacientes acamados em seus domicílios, não necessitando de internação hospitalar. É, portanto, menos onerosa, mais rápida e menos estressante, sendo considerada inócua, pois é o método de maior segurança para a obtenção de um diagnóstico morfológico. Possui sensibilidade e especificidade altas, com raros resultados falso-positivos e substitui ou complementa a biópsia de congelação, amenizando o estresse intra-operatório

diante de um diagnóstico e viabilizando o assentimento do paciente para uma terapêutica cirúrgica imediata mais radical.

De acordo com Freitas Jr (2001), a segurança da PAAF tem boa aceitabilidade devido ao baixo índice de complicações. A precisão e a confiabilidade da PAAF dependem de alguns aspectos relacionados ao tumor, tais como: o tamanho, a mobilidade e a localização.

A técnica PAAF é especialmente indicada na investigação de lesões parenquimatosas, nodulares, císticas e de localização superficiais, acessíveis por punção, em órgãos como mamas, tireóide, cabeça e pescoço, entre outros.

No entanto, a PAAF igualmente apóia a investigação clínica de lesões viscerais mais profundas, desde que orientadas, simultaneamente, por diagnóstico de imagem. Sobre isso, Ardengh (2006) afirma que a PAAF pode ser associada a Ecoendoscopia⁹ para auxiliar o diagnóstico pré-operatório de tumores neuroendócrinos para a comparação de resultados com outros testes de diagnóstico. Como resultado da análise de domínio específica para o *ViMeT* foram definidos:

- criação de um AV dinâmico, com dois objetos modelados: um para representar o órgão humano e outro para representar o instrumento médico;
- inclusão das funcionalidades de estereoscopia, detecção de colisão e deformação;
- uso de um SBGD para armazenar dados dos objetos modelados e das aplicações geradas;
- interface gráfica pra auxiliar a instanciação;
- integração de dispositivos específicos (luva e equipamento hápticos). No entanto, a inserção de dispositivos não convencionais não foi contemplada no presente projeto, porque estes não estavam disponíveis até a conclusão deste trabalho.

5.3.2 Projeto arquitetural

Com base no modelo de análise de domínio criou-se o projeto arquitetural do *framework* que tem como função traçar um panorama geral do *ViMeT*. Nesta fase foram

⁹ Método de investigação que combina a endoscopia digestiva ao ultra-som e produz, por meio de um transdutor de alta frequência situado na extremidade do endoscópio, imagens de alta resolução (GUARALDI *et al.*, 2005)

previstas a utilização do SGBD Derby (APACHE, 2006), uma camada representada por todas as classes do *ViMeT*, uma camada que representa a ferramenta de instanciação (*Wizard*) e as duas formas de instanciação. Na Figura 33 é mostrado o projeto arquitetural.

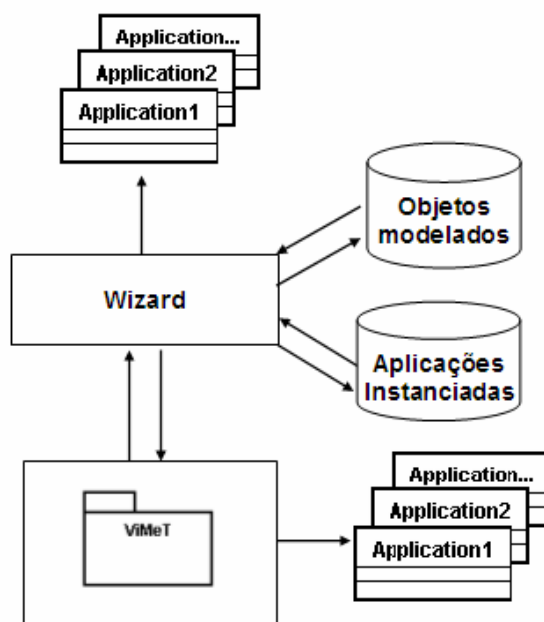


Figura 33 - Arquitetura do *ViMeT*

Salienta-se que o pacote *ViMeT*, apresentado na Figura 33, tem caráter exclusivamente ilustrativo, pois na verdade consiste no Projeto do *framework* e é detalhado na Seção 5.3.3.

A camada da *Wizard* e o projeto de Banco de Dados fazem parte de um trabalho que foi desenvolvido paralelamente e integrado ao *ViMeT*, descrito no Capítulo 6, onde também são apresentadas as formas de instanciação, incluindo os resultados de vários testes realizados com a finalidade de validar a flexibilidade e a facilidade de instanciação do *ViMeT*.

5.3.3 Projeto do *framework*

Esta atividade teve como objetivo construir o diagrama de classes do *framework*, neste projeto foi desenvolvido o diagrama de classes do *ViMeT*. Este diagrama foi desenvolvido já prevendo a inserção de novas classes, correspondentes a novas

funcionalidades e novas formas de interação, não implementadas nesta primeira versão do *framework*.

O ponto de partida desta atividade foi uma análise detalhada nos códigos-fonte, documentação e interface das aplicações a serem reutilizadas, por meio dos diagramas de classes obtidos por engenharia reversa. Na análise realizada no código-fonte destas aplicações surgiram diversas dificuldades, tais como documentação insuficiente, estrutura de classes fortemente acopladas. Além disso, foi percebido que não houvera planejamento para reúso, motivo pelo qual não estava disponível parametrização para permitir expansão do uso para aplicações genéricas.

Salienta-se que estas aplicações são projetos isolados e foram resultados de dois trabalhos de iniciação científica e um de mestrado. Cada uma delas possuía interfaces próprias e método próprio para a importação dos objetos modelados.

5.3.3.1 Detecção de Colisão com precisão

Detectar uma colisão é verificar o momento em que ocorre uma aproximação suficientemente pequena entre objetos de um ambiente virtual a ponto de possibilitar a ocorrência de uma sobreposição entre eles. Kera (2005) desenvolveu um procedimento para detecção de colisão a partir de um refinamento dos métodos *BoundingBox* (BB) e *BoundingSphere* (BS), fornecidos pela *API Java3D*.

A partir da detecção de colisão entre dois objetos por esses métodos é utilizado o conceito de *Octrees* para executar o refinamento. Primeiro, divide-se a área de detecção de colisão em octantes depois, escolhe-se o octante não vazio cujo centro apresenta a menor distância euclidiana em relação ao centro do objeto em colisão para refinamentos sucessivos.

O refinamento utiliza o método BS para envolver cada novo octante obtido. O processo é executado recursivamente até que o octante chegue a um tamanho mínimo (colisão detectada) ou esteja vazio (colisão não detectada). As classes originalmente desenvolvidas para esta funcionalidade estão representadas na Figura 34.

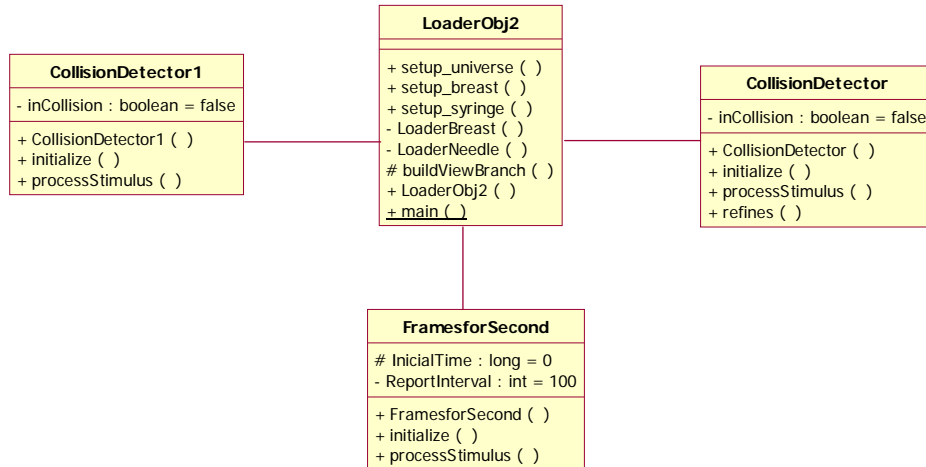


Figura 34 – Diagrama de classes do projeto detecção de colisão (KERA, 2005).

No AV existem dois objetos, sendo que a um deles (seringa) foi adicionado o método de interação utilizando o mouse e o outro objeto (mama) ficou estático na cena. Ao objeto estático foi adicionada a detecção de colisão.

Kera (2005) afirma que o método desenvolvido proporcionou um ganho na precisão em relação aos métodos da *API Java 3D*, sem perda significativa de desempenho. Na Figura 35 mostra-se a interface do método empregado no momento que ocorreu a colisão.

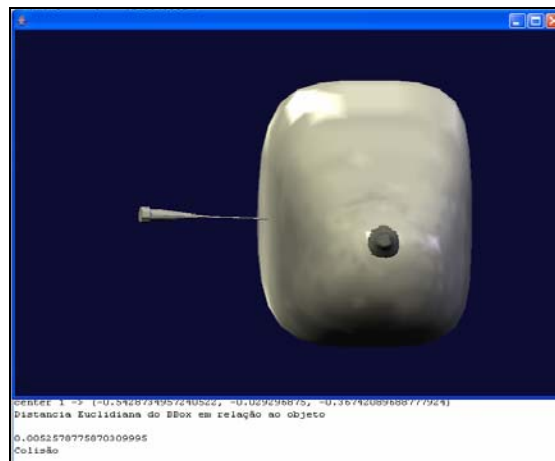


Figura 35 - Indicação de colisão da agulha e da mama (KERA, 2005)

5.3.3.2 Deformação

A deformação aparece em aplicações de RV para proporcionar maior realismo aos ambientes que os contêm. Sendo assim, é necessário que os objetos se modifiquem em

resposta a uma interação do usuário. Segundo Pavarini *et al.* (2005), o ideal é que a deformação ocorra em tempo real, por meio do reposicionamento dos vértices dos objetos modelados a partir de novas posições obtidas por equações matemáticas.

Entre os métodos existentes para deformação citados na literatura, foi implementado o método Massa-Mola, uma técnica baseada nas leis da Física que permite a reformulação de objetos deformados utilizando o conceito de nós de massa conectados por molas, proposto por Choi *et al.* (2002).

Nesta aplicação, escolhe-se primeiro o objeto modelado a ser deformado. Em seguida, seleciona-se o local no qual vai ser aplicada a deformação. O sistema determina o vértice mais próximo do local escolhido e calcula as novas posições do vértice e dos seus vizinhos, a partir de alguns parâmetros. O diagrama da Figura 36 mostra as classes originalmente desenvolvidas.

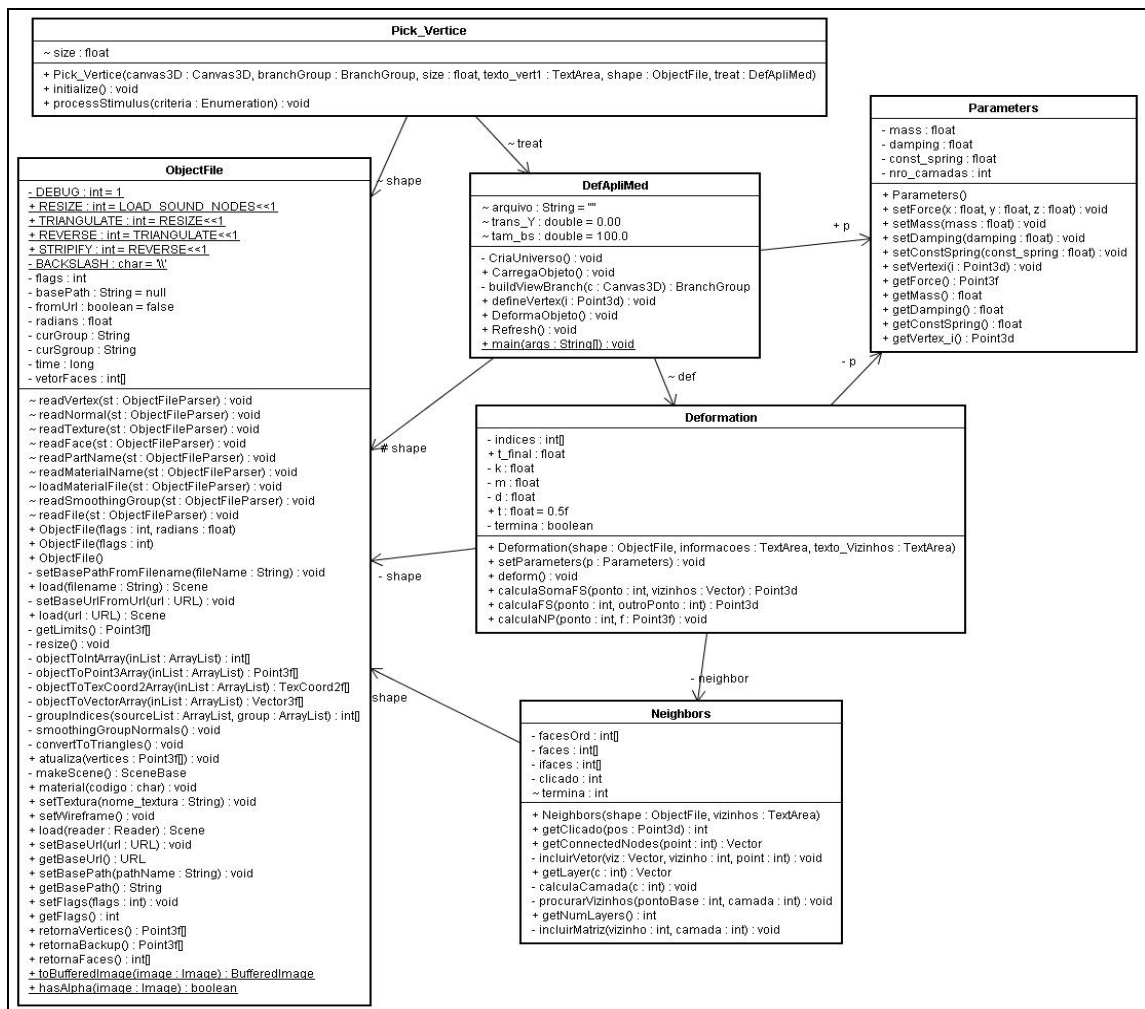


Figura 36- Diagrama de classes do projeto deformação (PAVARINI, 2006)

As classes *Pick_Vertice*, *ObjectFile*, *Deformation*, *Parameters*, *Layer* e *Neighbors* foram implementadas para permitir o reposicionamento dos vértices dos objetos 3D na cena, de forma a simular uma deformação ocorrida após uma interação.

A classe *ObjectFile* é responsável por carregar um arquivo no formato *wavefront* (*.obj*), tendo sido feitas algumas adaptações na classe *ObjectFile* da API Java3D para prover os vértices e as arestas dos objetos modelados.

Segundo Pavarini (2006), os testes da técnica de deformação foram realizados com valores empíricos, atribuídos aos parâmetros relacionados à força, massa, *damping* e constante da mola. A força é um parâmetro que varia conforme a intensidade da interação do usuário. Na Figura 37 são apresentados os resultados dos testes com a variação deste parâmetro com valores (-3,0) e (-0,3) Newtons, respectivamente, para os objetos (A) e (B) e (-0,7) e (-1,5) Newtons para os objetos (C) e (D).

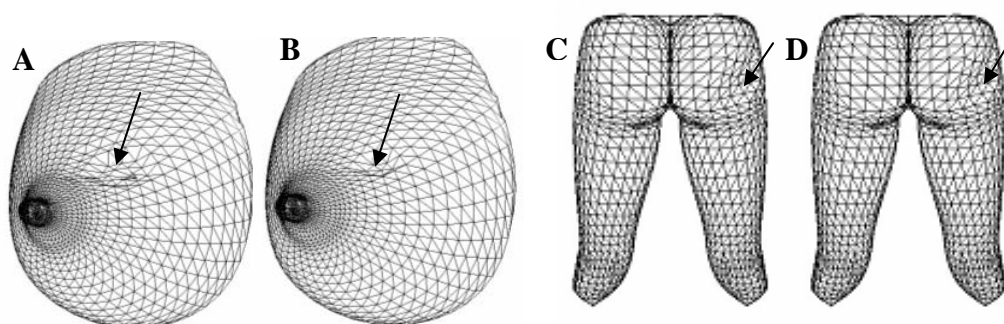


Figura 37 – Teste com variação de força (PAVARINI *et al.*, 2006).

A quantidade de vértices de um objeto permite a visualização de maior ou menor nível de detalhe envolvendo as deformações. Na Figura 38 observam-se os testes realizados no objeto Mama, onde, o objeto (A) possui 442 vértices e 880 faces e o objeto (B) possui 1762 vértices e 3520 faces. O objeto Nádegas da Figura 41 (C) possui 472 vértices e 922 faces e o objeto (D) possui 1890 vértices e 3732 faces.

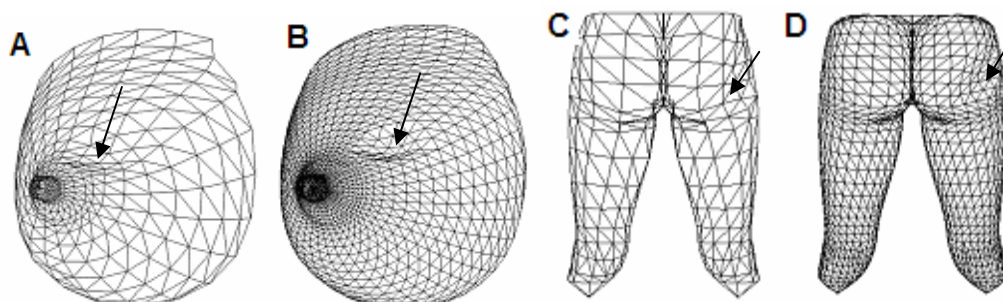


Figura 38 - Teste variando a quantidade de vértices dos objetos modelados (PAVARINI *et al.*, 2006).

O sistema calcula a quantidade de camadas que serão afetadas pela deformação, levando em consideração a força envolvida, os parâmetros que definem o comportamento da estrutura do objeto (constante da mola, constante de amortecimento e a massa) e a distância entre os vértices.

5.3.3.3 Estereoscopia

A visão estéreo é um dos principais mecanismos para o ser humano ter a sensação de profundidade. Tem este nome por precisar do uso de ambos os olhos. O olho esquerdo e o olho direito sempre vêem imagens diferentes (apesar de muito parecidas). De acordo com Santos (2000), o cérebro usa esta diferença para montar a imagem com profundidade.

Para implementar a estereoscopia foi inicialmente escolhido o método de anaglifos (BOTEGA, 2005) e (BOTEGA *et al.*, 2005). Para que este método de anaglifos possa ser utilizado, é necessário disponibilizar um par de imagens em cores diferentes (vermelho-azul ou vermelho-verde), correspondente a duas perspectivas de um mesmo objeto, que só serão vistas, respectivamente, pelas lentes com as mesmas cores de um óculos, que servirá de filtro para a visualização.

A implementação consistiu em um conjunto de classes capazes de separar as visões do olho esquerdo e direito, atribuindo as respectivas cores dos óculos utilizados para a visualização. O diagrama de classes pode ser observado na Figura 39 .

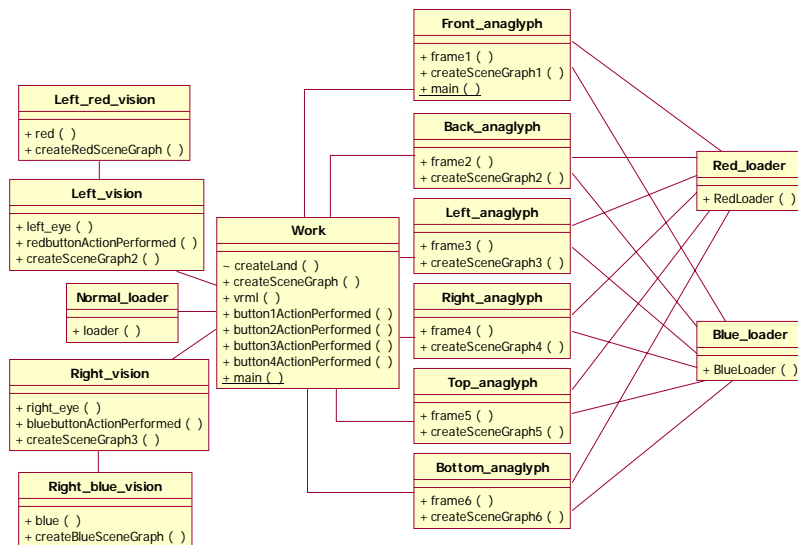


Figura 39 - Diagrama de classes projeto estereoscopia (BOTEGA, 2005)

Para a construção desta aplicação foi utilizado o par vermelho-azul. Estas cores foram escolhidas empiricamente após verificação com diversos objetos tridimensionais. O modelo apresenta uma inclinação no eixo Y e translação no eixo X, de modo a representar o ponto-de-vista de cada olho. Na Figura 40 (a) mostra-se o objeto vermelho e na Figura 40 (b), o objeto azul.

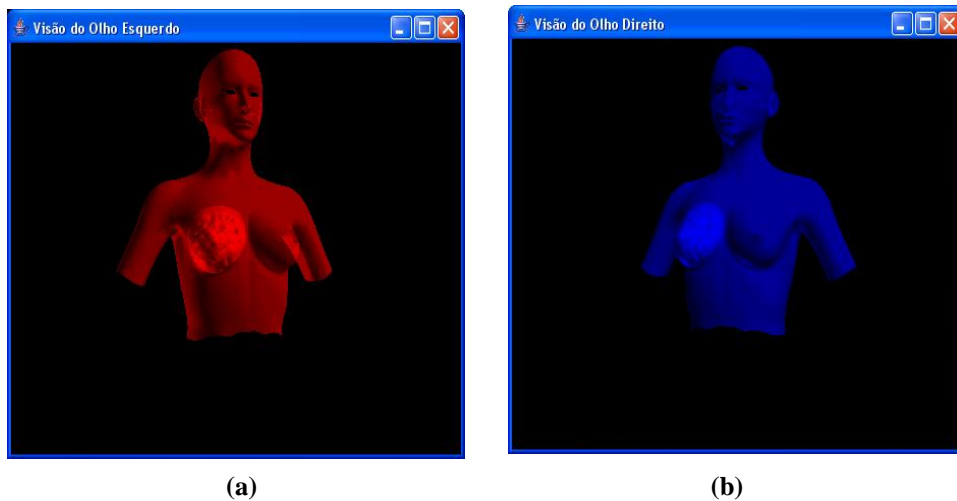


Figura 40 - (a) Representação da “Visão do Olho Esquerdo” com a camada vermelha, (b) Representação da “Visão do Olho Direito” com a camada azul (BOTEGA, 2005)

Posteriormente, essas mesmas visões foram fundidas no mesmo ambiente virtual, respeitando os devidos ângulos de inclinação e coloração, formando o anaglifo, conforme mostrado na Figura 41.

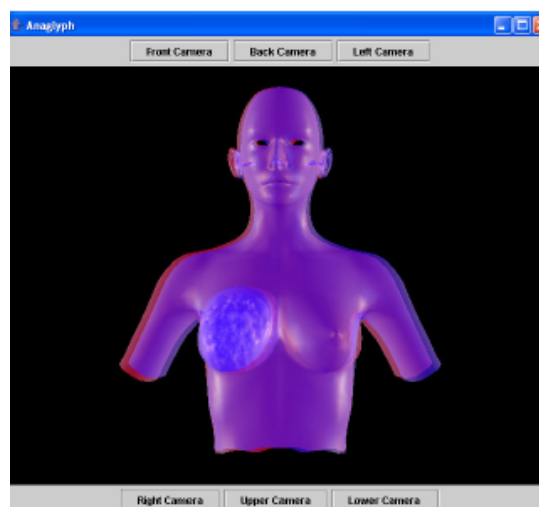


Figura 41 – Geração do Anaglifo (BOTEGA, 2005)

5.3.3.4 Construção do Diagrama de Classes do *ViMeT*

A partir das análises dos códigos das aplicações citadas foi criado um primeiro diagrama de classes do *ViMeT*, representando o resultado da fase de Projeto do *framework*. Estas aplicações, como já discutido anteriormente, foram desenvolvidas de forma isolada e não possuíam documentação adequada. Além disso, nesta fase foi percebido que as estruturas de classes não estavam preparadas para serem reutilizadas diretamente na construção do *ViMeT*.

Para reutilizá-las no *framework* foi necessário retirar as classes e os métodos responsáveis pela interface, carregamento de objetos e criação do AV, pois estas funções deveriam ser genéricas no *ViMeT*. Sendo assim, foram utilizadas no *ViMeT* apenas as classes e métodos responsáveis diretamente pela implementação da detecção de colisão, deformação e estereoscopia. O Quadro 4 apresenta as classes, que inicialmente seriam reutilizadas, indicando o nome, a aplicação à qual pertence e a sua finalidade.

Aplicação	Classes	Finalidade
Detecção de Colisão	<i>CollisionDetector</i>	Detecta quando um objeto interpenetra o espaço de colisão pré-determinado por um cubo ou esfera. Primeiramente detecta a colisão entre dois objetos com os métodos nativos da API Java3D e depois faz um refinamento fazendo com que a área de colisão diminua.
Estereoscopia	<i>Front_Anaglyph</i>	Disponibiliza o Anaglifo na sua posição inicial
	<i>Back_Anaglyph</i>	Rotaciona o Anaglifo em 180 graus positivos no eixo Y
	<i>Right_Anaglyph</i>	Rotaciona Anaglifo em 90 graus negativos no eixo X
	<i>Left_Anaglyph</i>	Rotaciona Anaglifo em 90 graus positivos no eixo X
	<i>Top_Anaglyph</i>	Rotaciona Anaglifo em 90 graus negativos no eixo Y
	<i>Botton_Anaglyph</i>	Rotaciona Anaglifo em 90 graus positivos no eixo Y
Deformação	<i>Pick_Vertice</i>	Permite ao usuário escolher qual vértice do objeto carregado será deformado.
	<i>ObjectFile</i>	Permite prover as faces do objeto e a conversão dessas faces em um vetor de arestas.
	<i>Deformation</i>	Utiliza o método Massa-Mola para reposicionar os vértices e demais estruturas de objetos 3D na cena simulando uma deformação ocorrida após uma interação.
	<i>Parameters</i>	Armazena e trata os parâmetros necessários para gerar a deformação utilizando o método Massa-Mola.
	<i>Layer</i>	Armazena quais vértices pertencem a quais camadas.
	<i>Neighbors</i>	Calcula a deformação que será propagada para os pontos vizinhos do vértice selecionado.

Quadro 4 - Classes reutilizadas no *ViMeT*

É interessante salientar neste ponto que optou-se por apresentar um diagrama de classes inicial (Figura 42) e um mais completo adiante, como forma de destacar a iteratividade da metodologia. O diagrama inicial apresenta as classes *CollisionDetector* da aplicação de detecção de colisão; as classes *PickVertice*, *Deformation*, *Parameters*, *Layers*, *Neighbors* e *ObjectFile* da aplicação de deformação e as classes *Anaglyph_Front*, *Anaglyph_Back*, *Anaglyph-down*, *Anaglyph_right*, *Anaglyph_left* e *Anaglyph_up* são da aplicação estereoscopia. Nesta fase do projeto a classe *Environment* representaria o AV e a integração das demais classes.

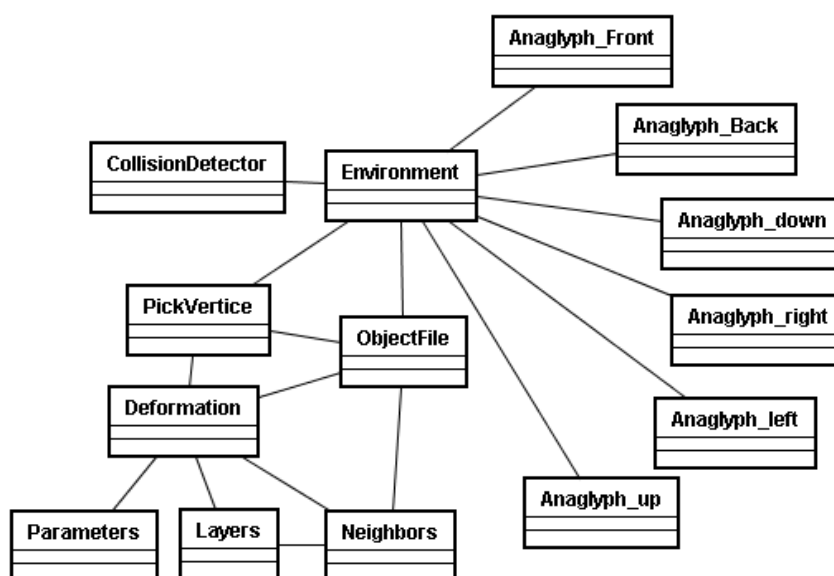


Figura 42 - Primeiro diagrama de classes proposto para o *ViMeT*

Iniciou-se a implementação a partir do primeiro diagrama, mas notou-se que ele não representava de forma adequada todas as características do *ViMeT*. Desta forma, o projeto inicial foi gradativamente sofrendo modificações até obter-se o diagrama representado na Figura 43. Neste diagrama foram previstas todas as classes a serem desenvolvidas para fornecer as funcionalidades necessárias ao *ViMeT*.

As classes que indicam funcionalidades futuras são destacadas pelos retângulos cinza. Tais funcionalidades referem-se a novas técnicas para implementar as funcionalidades de detecção de colisão, deformação e estereoscopia e à inclusão de dispositivos não convencionais (luva e equipamento háptico). As classes dentro do quadrado hachurado representam as classes responsáveis pela criação do AV, importação dos objetos modelados e manipulação do Banco de Dados, implementadas na presente versão do *framework*.

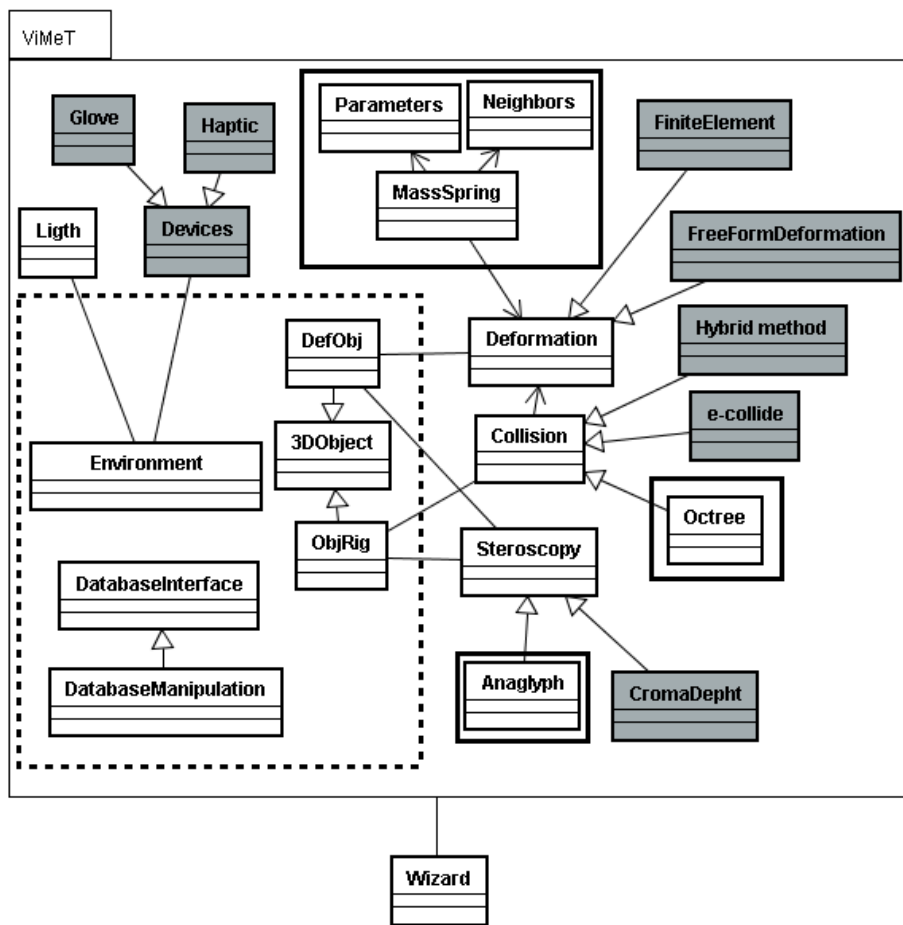


Figura 43 - Diagrama de Classes completo (OLIVEIRA, 2006b).

As classes *Deformation*, *Collision*, *Stereoscopy* e *Ligth* representam aquelas apontadas como necessárias na fase final da atividade de projeto do *framework*. As classes destacadas dentro de retângulos com linhas grossas representam as classes reutilizadas das aplicações de deformação, detecção de colisão e estereoscopia. Para facilitar a continuidade do desenvolvimento do *framework*, essas classes receberam o nome das técnicas implementadas (*Anaglyph*, *MassSpring* e *Octrees*), tendo sido criadas classes genéricas para integrá-las ao *ViMeT* (*Deformation*, *Collision* e *Stereoscopy*). A classe *Wizard* é responsável pela interface gráfica e pela instanciação automática do *ViMeT*.

5.3.4 Implementação do framework

Com base no diagrama de classes e com o código-fonte extraído das aplicações a serem reutilizadas iniciou-se a implementação, considerando que o reuso de software é uma

partes que foram planejadas para implementações futuras. Salienta-se que algumas classes previstas não foram necessárias devido aos recursos providos pela *API Java 3D* em relação à iluminação. Após as constatações iniciais listou-se as fases de implementação do *ViMeT* e cada uma destas fases é detalhada a seguir. As fases (a), (b), (c), (d) e (e) foram implementadas simultaneamente com a fase (f), conforme pode ser observado na Figura 45

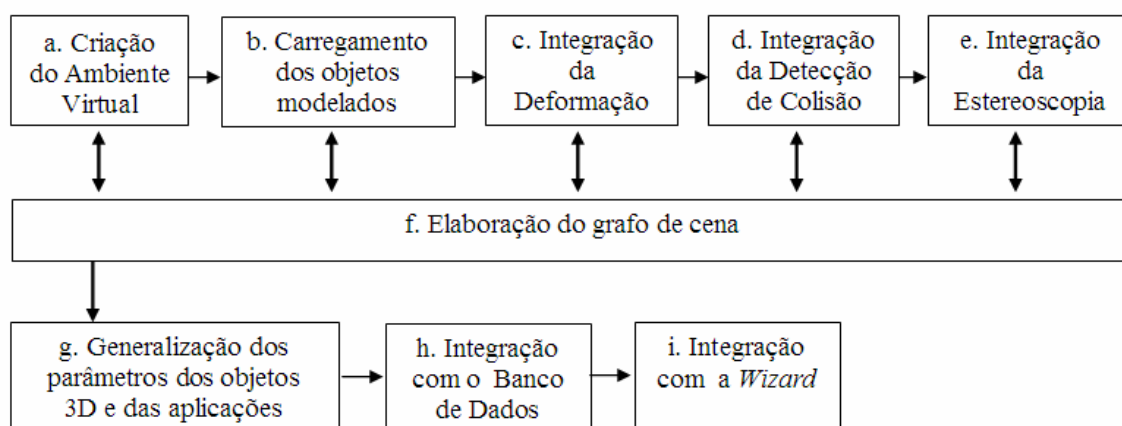


Figura 45 - Fases da implementação

a) Criação do Ambiente Virtual

A classe *Environment* é responsável pela criação do AV e herda características da classe *VirtualUniverse*, da *API Java3D*. As luzes do AV e o *background* também são definidos na classe *Environment* e constituem um dos *frozen spots* do *ViMeT*.

Foi escolhida a opção *VirtualUniverse* por possuir o subgrafo de visualização, com todos os objetos necessários para criar, posicionar, direcionar e movimentar a câmera virtual. O projeto inicial do *ViMeT* não prevê a utilização do subgrafo de visualização, porém caso seja necessário utilizá-lo em alguma aplicação desenvolvida a partir do *ViMeT* não haverá a necessidade de alterar a classe responsável pela criação do AV. Garante-se, assim, a possibilidade de utilização de dispositivos não convencionais para visualização, projeção e navegação.

b) Carregamento dos objetos modelados

Após o AV estar concluído iniciou-se a implementação do método de importação dos objetos modelados para o AV. Cada aplicação implementada anteriormente utilizou um tipo de *Loader*. Na aplicação de deformação foi utilizado o *loader* do *ObjectFile*, que retorna um

objeto do tipo *Shape*; já as aplicações de detecção de colisão e estereoscopia utilizaram o *loader default* da API Java3D retornando um objeto do tipo *Group*.

Os tipos de *loaders* estão diretamente ligados aos tipos de objetos modelados. O objeto que simula um órgão humano necessita somente da manipulação dos vértices e atualmente sua manipulação é realizada por meio do teclado. O objeto que simula um instrumento médico não necessita da manipulação dos vértices e sua manipulação é realizada por meio do mouse, implementada pela classe *Mouse*.

A aplicação de deformação utiliza um pacote próprio para carregar os objetos, que disponibiliza os vértices e as arestas dos objetos carregados para manipulação. Diante desta situação percebeu-se a necessidade da utilização de dois tipos de *loader* no *ViMeT*, um para cada tipo de objeto a ser carregado. Para isso foram desenvolvidas três classes, sendo elas:

- ***3DObject***: classe abstrata que contém os atributos e métodos comuns aos dois tipos de objetos modelados.
- ***DefObj***: subclasse de *3DObj*, que reutiliza o pacote implementado na aplicação de deformação. Nesta classe também são implementados todos os métodos envolvidos na deformação, disponibilizados os valores para a translação, escala, rotação e parâmetros necessários para a simulação da deformação.
- ***RigObj***: subclasse de *3DObj*, que utiliza o método de importação padrão da API Java3D, o *load.objectFile()*. Além dos parâmetros de escala, translação, rotação, disponibiliza o parâmetro distância euclidiana, reutilizado da aplicação de detecção de colisão.

c) Integração da Deformação

Após a definição do AV, decidiu-se que da aplicação de deformação seriam reutilizadas apenas quatro classes: (1) ***Deformation***, que simula a deformação ocorrida após uma interação; (2) ***Neighbors***, que calcula a deformação propagada para os pontos vizinhos do vértice selecionado; (3) ***Parameters***, que armazena e trata os parâmetros necessários para gerar a deformação utilizando o método Massa-Mola e (4) ***Ordена***, que acessa os vértices vizinhos ao vértice selecionado.

No projeto inicial do *ViMeT* foi prevista uma classe abstrata denominada *Deformation*, com a finalidade de conter atributos e métodos similares proveniente das várias

técnicas possíveis de deformação, definindo-se que suas subclasses receberiam o nome da técnica utilizada. Por esse motivo, a classe *Deformation* da aplicação de deformação foi renomeada para *MassSpring*.

A classe *Deformation* do *ViMeT* é uma classe abstrata que possui os métodos para definir os parâmetros da deformação e o método *deform()* que implementa a técnica Massa-Mola implementada. Porém não é possível conhecer antecipadamente os atributos e métodos comuns às diversas técnicas que poderão ser futuramente implementadas. Dessa forma, quando novas técnicas forem integradas ao *ViMeT*, estas deverão ser implementadas como subclasses da classe *Deformation*.

d) Integração da Detecção de Colisão

A integração da detecção de colisão foi relativamente simples, pois como já mencionado, esta aplicação utiliza os métodos *BoundingBox* e *BoundingSphere* da API Java3D. Foi inserido o conceito de *Octrees* para o refinamento recursivo, utilizando-se a distância Euclidiana entre o centro dos dois objetos como critério de finalização do método. Quando esta distância é menor que um determinado valor, a colisão obtém uma precisão aceitável.

Desta aplicação foi reutilizada a classe *CollisionDetector*, responsável por detectar quando um objeto interpenetra o espaço de colisão pré-determinado por um cubo ou esfera. No projeto do *ViMeT* a classe foi renomeada para *Octrees*. Além disso, ela se tornou uma subclasse da classe abstrata *Collision*, devido aos mesmos motivos descritos anteriormente ao integrar-se a aplicação de deformação. Para fins de testes, foi implementada uma subclasse da classe abstrata *Collision*, denominada *CollisionJava*, que é responsável pela percepção da colisão por meio dos métodos nativos da API Java3D.

Foi implementada uma interface denominada *CollisionListener* que possui o método *collisionPerformed*. Este método foi associado à classe abstrata *Collision* e para que ocorra a detecção de colisão, as subclasses *CollisionJava* e *Octrees*, que representam esta funcionalidade, realizam uma chamada a este método quando ocorre uma colisão. Esta estrutura foi implementada para garantir a flexibilidade e também a facilidade de integração de novas técnicas.

e) Integração da Estereoscopia

Após um estudo detalhado da aplicação previamente desenvolvida de estereoscopia, chegou-se à conclusão que não seria viável a sua utilização, pois não era possível incluir a interação por meio de mouse, teclado ou outros dispositivos. Por esse motivo implementou-se novamente a técnica anaglifo. Não houve a necessidade de criar uma classe separada, pois a questão de visualização dos objetos é inerente ao AV e, por isso, foi tratada na classe *Environment*.

Como já foi discutido, a classe *Environment* é responsável pelo AV e nele são adicionados os objetos importados. A estereoscopia inerente a esses objetos, mas opcional ao usuário, é também implementada nesta classe. Para cada objeto modelado e importado no AV, o nó *Shape* foi duplicado e foram aplicados os atributos necessários para se obter a transparência e a paralaxe, necessárias à técnica de anaglifos. A estereoscopia é detalhada na fase (f).

Para garantir a flexibilidade do *ViMeT* a estereoscopia não é uma funcionalidade fixa e as aplicações desenvolvidas a partir do *framework* podem ou não utilizá-la, visto que a escolha desta funcionalidade é feita por passagem de parâmetros, conforme pode ser visto na Figura 46.

```
public Application (Canvas3D c) {
    super(c, true);
}
```

Figura 46 -Trecho do código com o método *super* da classe *Environment* recebendo o parâmetro da estereoscopia

Se o método *super* contiver o valor *true*, a aplicação utilizará a estereoscopia. Por não existirem implementadas outras técnicas de estereoscopia, não foi criada a classe abstrata *Stereoscopy*, prevista no diagrama de classes inicialmente proposto.

f) Elaboração dos Grafos de Cena

O Grafo de Cena do *ViMeT* foi desenvolvido simultaneamente com as fases de implementação, descritas nos itens anteriores. O grafo de cena para aplicações da Figura 47 refere-se a aplicações sem visão estereoscópica. Na Figura 48 é apresentado o grafo de cena para uma aplicação com estereoscopia.

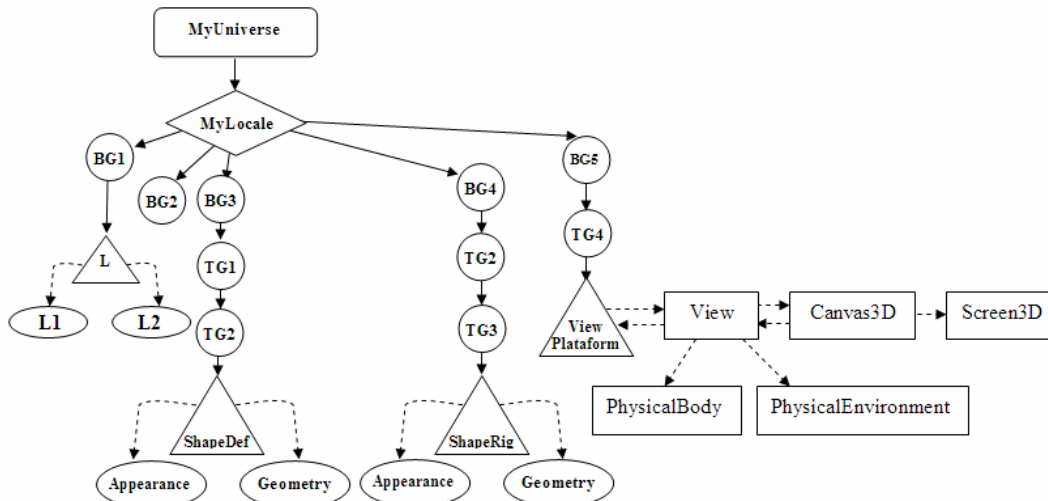


Figura 47 - Grafo de Cena gerado pelo ViMeT (aplicação sem estereoscopia)

Os nós do tipo *Group* são os *BranchGroup* (BG 1 a BG5) que contêm os objetos que podem ser dinamicamente inseridos ou removidos do grafo de cena em tempo de execução e os *TransformGroup* (TG1 a TG9) responsáveis por permitir a mudança de posição, orientação e tamanho de objetos visuais no AV. Todos os nós são associados ao nó *MyLocale* em tempo de execução. Os nós do tipo *Leaf* (*ShapeDef*, *ShapeRig*, *ViewPlataform* e *L*) representam propriedades da cena, como a geometria e luzes. Os nós do tipo *NodeComponent* (*L1*, *L2*, *Appearance* e *Geometry*) são responsáveis por especificar as propriedades de um determinado objeto. Os outros objetos (*View*, *Canvas3D*, *Screen3D*, *PhysicalBody* e *PhysicalEnvironment*) representam outros tipos de classes.

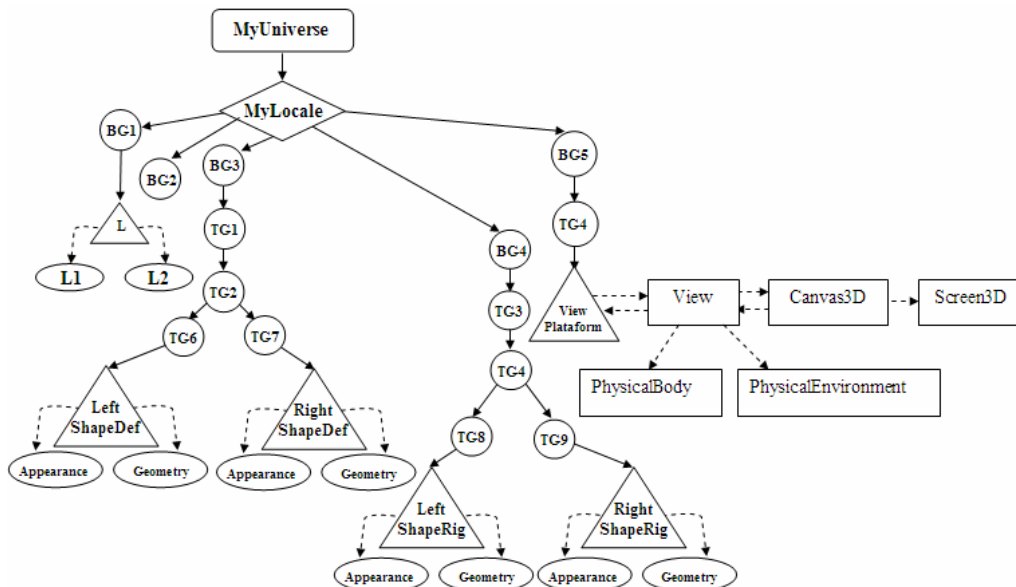


Figura 48 - Grafo de Cena gerado pelo ViMeT (aplicação com Estereoscopia)

Na Figura 49 foram duplicados os nós *TransformGroup*, representados no grafo de cena por TG6 e TG7 para o objeto deformável, e TG8 e TG9 para o objeto rígido. A esses nós foram associados os nós *Shape*, um para a visão esquerda e outro para a direita, onde foram adicionadas a aparência e a geometria em cada um. Salienta-se que a geometria é a mesma, ou seja, não há dois objetos modelados, mas sim dois pontos de vista da mesma geometria. Para facilitar o entendimento do grafo de cena do *ViMeT* foi elaborado o Quadro 5 com os tipos de nós, seus nomes e suas funções. No Quadro 6 é apresentado um mapeamento entre os nós do Grafo de Cena e as classes e seus respectivos métodos envolvidos na construção do AV.

Tipo de Nó		Nome	Finalidade
<i>VirtualUniverse</i>		<i>MyUniverse</i>	Representa o universo virtual
<i>Locale</i>		<i>MyLocale</i>	Gerencia o sistema de coordenadas do universo virtual
<i>Group</i>	<i>BranchGroup</i>	BG1	Responsável pela iluminação do universo virtual
		BG2	Gerencia <i>Background</i> , onde a mudança de cor ocorre em função da funcionalidade de estereoscopia.
		BG3	Possui todas as características do objeto deformável
		BG4	Possui todas as características do objeto rígido
		BG5	Possui as características do subgrafo de visualização.
	<i>TransformGroup</i>	TG1	Gerencia as movimentações do objeto deformável
		TG2	Gerencia as transformações do objeto deformável
		TG3	Gerencia as movimentações do objeto rígido
		TG4	Gerencia as transformações do objeto rígido
		TG5	Controla as transformações ocorridas no subgrafo de visualização
<i>Leaf</i>	<i>ShapeDef</i>	Representa o objeto modelado que irá sofrer a deformação.	
	<i>ShapeRig</i>	Representa o objeto modelado responsável pela detecção de	
	<i>View Plataform</i>	Representa a localização da câmera no grafo de cena, sendo um ponto de coordenadas (x,y,z).	
	<i>L</i>	Controla os tipos de luzes utilizadas para a iluminação do AV.	
<i>NodeComponent</i>	<i>Appearance</i>	Controla a aparência da geometria como transparência e textura dentre outras	
	<i>Geometry</i>	Gerencia as características da geometria representada pelos <i>Shapes</i>	
	<i>L1</i>	Representa o tipo (<i>DirectionalLight</i>) de luz direcional.	
	<i>L2</i>	Representa o tipo (<i>ambientLight</i>) de luz do próprio AV	
Outros Objetos	<i>View</i>	Armazena todas as informações necessárias para renderizar o grafo de cena, além de coordenar todo o processo de renderização.	
	<i>Physical Body</i>	Possui informações para calibração dos objetos que representam o corpo físico do usuário, como a localização dos olhos e a distância	
	<i>Physical Environment</i>	Contém as informações para calibração do mundo físico, por meio de dispositivos de áudio e sensores de movimento.	
	<i>Canvas 3D</i>	Representa a classe que garante a apresentação gráfica da cena enquadrada em uma janela de visualização	
	<i>Screen3D</i>	Contém as propriedades físicas da tela.	

Quadro 5 - Descrição dos componentes do grafo de cena do *ViMeT*

Nó	Objetos	Classes	Métodos	
<i>MyUniverse</i>	VirtualUniverse	Environment	public Environment()	
<i>MyLocale</i>	Locale		public Environment()	
BG1	Branchgroup		myLocale.addBranchGroup()	
BG2			myLocale.addBranchGroup()	
BG3			DefObj	myLocale.addBranchGroup()
BG4			RigObj	myLocale.addBranchGroup()
BG5			Enviroment	myLocale.addBranchGroup()
TG1		TransformGroup	DefObj e Mouse	public void setMotionTransform(TransformGroup tg) public TransformGroup getMotionTransform()
TG2	DefOb		public void setScale(Vector3d s) public void setTranslation(Vector3d t) public void setRotation(AxisAngle4d r)	
TG3	RigObj		public void setMotionTransform(TransformGroup tg) public TransformGroup getMotionTransform()	
TG4	RigObj		public void setScale(Vector3d s) public void setTranslation(Vector3d t) public void setRotation(AxisAngle4d r)	
TG5	Environment		addBranchGroup()	
<i>ShapeDef</i>	Leaf		Object3D e ObjDef	public Deformation getDeformation() public Collision getCollisionDetector() public Shape3D getShape() public Shape3D getStereoShape()
<i>ShapeRig</i>		Object3D e ObjRig	public Shape3D getShape()	
<i>Appearance</i>	NodeComponent	ObjDef e ObjRig	getAppearance()	
<i>Geometry</i>		ObjDef e ObjRig	getGeometry()	
<i>L</i>	Leaf	Environment	myLocale.addBranchGraph(light);	
<i>View Plataform</i>			Vector3f light1Direction = new Vector3f(4.0f, -7.0f, -12.0f);	
<i>L1</i>	NodeComponent		ambientLightNode = new AmbientLight(ambientColor);	
<i>L2</i>			buildViewBranch()	
<i>View</i>	Outros objetos		buildViewBranch()	
<i>Physical Body</i>			buildViewBranch()	
<i>Physical Environment</i>			buildViewBranch()	
<i>Canvas 3D</i>			buildViewBranch()	
<i>Screen3D</i>			buildViewBranch()	

Quadro 6 - Mapeamento entre os nós do Grafo de Cena com as classes e métodos do *ViMeT*

g) Generalização dos parâmetros dos Objetos 3D e das Aplicações

Uma das dificuldades na implementação das classes em um *framework* é torná-las genéricas o suficiente para garantir uma instanciação eficiente. No *ViMeT*, as funcionalidades de cada aplicação gerada recebem valores por parâmetros que podem ser registrados em uma base de dados, juntamente com os objetos modelados, a fim de permitir a geração de novas aplicações. Após a integração das funcionalidades do *ViMeT* foram feitos alguns testes para verificar se uma aplicação implementada era capaz de executar as três funcionalidades de forma correta. Após ter verificado que a integração das funcionalidades estava correta iniciou-se a generalização dos parâmetros, ou seja, revisão dos atributos que deveriam ser flexíveis, tornando-os variáveis no código:

- valores para escala, translação e rotação dos objetos modelados;
- valores da massa, *damping*, constante da mola e força, para técnica de deformação Massa-Mola;
- valor da Distância Euclidiana para a técnica de detecção de colisão *Octrees*;
- valor da Paralaxe para a Estereoscopia;
- seleção das técnicas de detecção de colisão e estereoscopia;
- seleção dos objetos modelados a serem utilizados.

g) Integração com o Banco de Dados

A integração do *ViMeT* com o BD foi realizada juntamente com a generalização dos parâmetros e consiste em consultar, alterar e gravar os parâmetros no BD, por meio da ferramenta *Wizard*. Por não constituir o objetivo diretamente relacionado ao *framework*, mas uma ferramenta para auxiliar a sua utilização, suas classes serão descritas no Capítulo 6.

i) Integração com a Wizard

A ferramenta *Wizard* constitui apenas uma classe com vários métodos e a sua integração consistiu em utilizar esta classe para construir a interface que permite a instanciação das classes do *ViMeT*. A interface possibilita as seguintes tarefas: definir os parâmetros para os objetos, permitir a escolha de funcionalidades, gerar o código-fonte em

Java e, finalmente, compilar o código gerado para obter a aplicação. O Capítulo 6 apresentará detalhes da implementação da classe *Wizard* e do projeto do Banco de Dados.

5.3.5 Teste do framework

Esta atividade teve como objetivo verificar se o *ViMeT* implementava corretamente as funcionalidades planejadas nas Seções 5.3.3 e avaliar a sua usabilidade. Testar um *framework* consiste em desenvolver aplicações a partir dele, verificando se as aplicações derivadas contêm as mesmas funcionalidades da aplicação inicial.

No caso do *ViMeT* as funcionalidades existentes são deformação, detecção de colisão e estereoscopia, com seus parâmetros e transformações. Então uma aplicação desenvolvida a partir do *ViMeT* deverá possuir as funcionalidades, os parâmetros para cada uma delas e suas respectivas transformações (escala, translação e rotação). Ao final, o funcionamento deve ocorrer da mesma forma que em uma aplicação desenvolvida sem o *framework*, dentro do domínio definido.

Para fins de teste foi criada uma classe *Application* e nela foram instanciadas as classes do *ViMeT* à medida que eram implementadas. Criaram-se as classes *Environment*, *3DObject*, *DefObj* e *RigObj*, e foram feitos os primeiros testes na classe *Application*. Logo após a integração das classes responsáveis pela deformação e detecção de colisão foram, novamente, feitos testes com a classe *Application*.

Esta fase foi muito importante, porque ajudou a descobrir erros e limitações nas classes desenvolvidas paralelamente à implementação. No momento de instanciá-las notou-se que haviam *hot spots*, que poderiam comprometer a flexibilidade do *ViMeT*. Todos os testes são apresentados em forma de resultados obtidos e discutidos detalhadamente no Capítulo 7.

5.3.6 Documentação

Em qualquer *framework* a documentação é um item primordial para viabilizar sua utilização. No *ViMeT* a documentação foi feita por meio de um *Cookbook*, que auxiliará o desenvolvedor de aplicações a partir do *ViMeT* e integrar novas funcionalidades a ele. O

Cookbook do *ViMeT* está detalhado no Apêndice A e pode ser consultado em http://galileu.fundanet.br/bcc_bsi/bcc/lapis/projetos/ana/cookbook.php.

Outra forma usada para documentação foi a *javadoc* (SUN, 2006a), que está parcialmente disponibilizado no Apêndice B e pode se consultado no endereço eletrônico http://galileu.fundanet.br/bcc_bsi/bcc/lapis/projetos/ana/javadoc.php. Todas as classes do *ViMeT*, assim como as classes das aplicações reutilizadas estão ali documentadas e auxiliam na utilização do *ViMeT*.

5.4 Considerações Finais

Neste capítulo foi apresentada a metodologia empregada no desenvolvimento do *ViMeT*. Inicialmente foram descritas as aplicações que foram reutilizadas para a construção do *framework* e que representam as suas principais funcionalidades. Durante o processo de construção utilizando a metodologia proposta por Bosch *et al.* (1999), verificou-se a importância de cada uma das fases e dos testes durante toda a implementação, os quais permitiram executar correções durante todo o processo. O próximo capítulo irá apresentar as implementações da *Wizard* e do projeto do Banco de Dados.

6 WIZARD E BANCO DE DADOS

Existem duas formas para a utilização do *ViMeT*: instanciação direta das classes (Caixa branca) e instanciação por meio da *Wizard* (Caixa Preta), uma ferramenta que auxilia usuários a utilizarem as classes do *ViMeT* sem prévio conhecimento e gera o código-fonte (*.java*) e a aplicação (*.class*). Salienta-se que são considerados usuários do *ViMeT* os desenvolvedores de aplicações. Este capítulo apresenta o Projeto do BD desenvolvido por Rossato (2006), além de descrever a implementação da *Wizard* e apresentar a interface disponibilizada.

6.2 Projeto do Banco de Dados

Como já foi mencionado, o projeto do Banco de Dados foi desenvolvido em um projeto paralelo, e teve como objetivos pesquisar entre os SGBDs gratuitos um que fosse ideal para ser integrado ao *ViMeT*. Uma vez realizada a escolha, deveria ser construídos um BD e uma interface para sua manutenção. Na Figura 49 pode ser observado o esquema geral do BD.

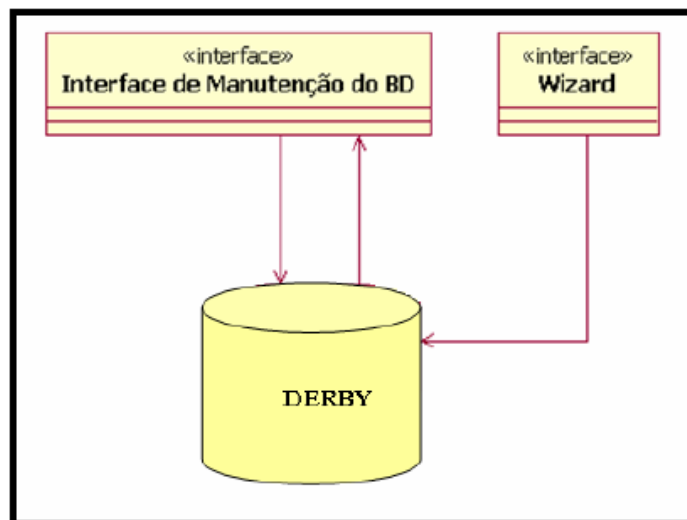


Figura 49 - Esquema geral do BD (ROSSATO, 2006).

Ao ser feita a integração do BD ao *ViMeT*, foi implementada a alteração das aplicações geradas no BD, visto que originalmente não permitia a manutenção de aplicações

geradas (ROSSATO, 2006). O projeto do BD possui duas tabelas: uma de armazenamento dos objetos e outra das aplicações desenvolvidas por meio da *Wizard*. O relacionamento das duas tabelas é de dois para vários, ou seja, uma aplicação tem sempre dois objetos e um objeto por estar em muitas aplicações, conforme pode ser observado na Figura 50.

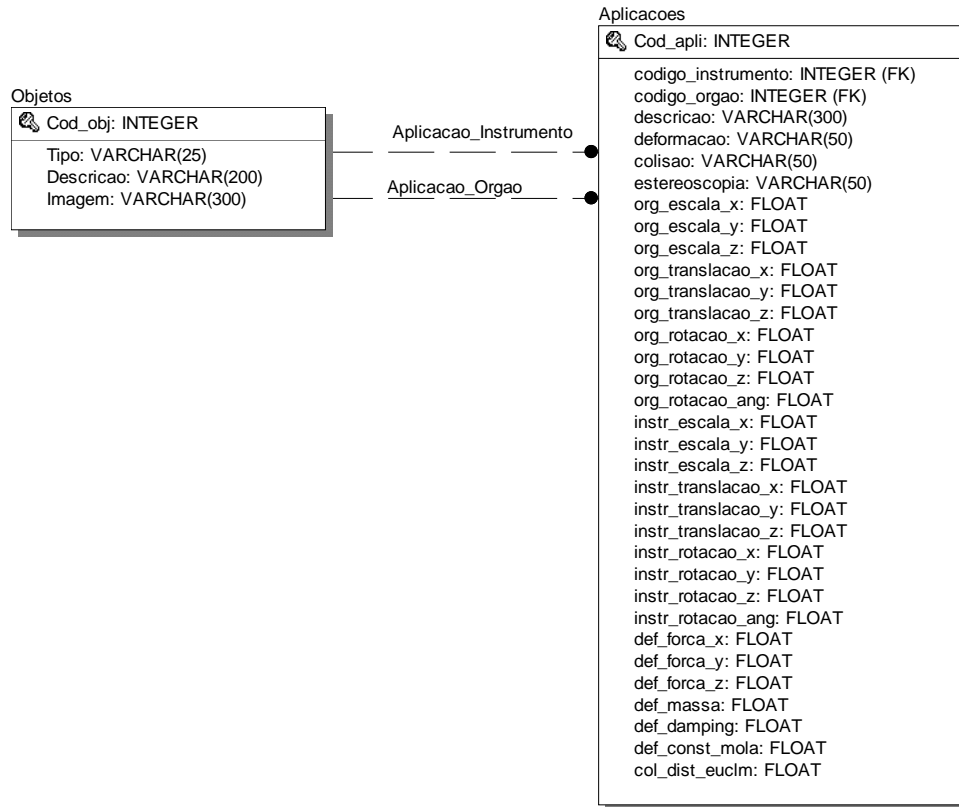


Figura 50 - Modelo relacional do BD da Wizard (ROSSATO, 2006)

A tabela OBJETOS armazena os órgãos humanos modelados e os instrumentos médicos. Contém o código do objeto como chave primária, o tipo (órgão ou instrumento médico), uma descrição detalhada do objeto e o caminho da imagem, isto é, o local de armazenamento do objeto inserido.

A tabela APLICACOES é utilizada para gravar as aplicações desenvolvidas por meio da *Wizard* para posterior recuperação e alteração. Contém como chave primária o código da aplicação, os códigos do órgão e do instrumento médico, provenientes da tabela OBJETOS, como chaves estrangeiras, descrição da aplicação desenvolvida, as funcionalidades que foram utilizadas e os seus respectivos parâmetros.

O projeto *Wizard* contém algumas classes que são descritas juntamente com suas respectivas funções no Quadro 7. Na Figura 51 pode ser observado o Diagrama de Classes da *Wizard* original.

Classes	Finalidade
<i>Wizard</i>	Instancia as classes do <i>ViMeT</i> , <i>FrameworkDatabaseApplication</i> , <i>BuildCode</i> , <i>Parameters</i> , <i>ParamOrgan</i> , <i>ParamMedInstr</i> , <i>ParamCollision</i> , <i>ParamDeformation</i> e <i>ParamEst</i> .
<i>Application</i>	Classe desenvolvida por meio do <i>ViMeT</i>
<i>Environment</i>	Responsável pela criação do ambiente 3D
<i>FrameworkDatabaseApplication</i>	Responsável por tratar o BD
<i>Interface</i>	Interface de manutenção do BD
<i>BuildCode</i>	Responsável por gerar o código Java da aplicação
<i>Parameters</i>	Responsável por tratar os parâmetros dos objetos e das técnicas de colisão, deformação e estereoscopia
<i>ParamOrgan</i>	Responsável por ler os parâmetros dos órgãos
<i>ParamMedInstr</i>	Responsável por ler os parâmetros dos instrumentos médicos
<i>ParamCollision</i>	Responsável por ler os parâmetros da técnica <i>Octrees</i>
<i>ParamDeformation</i>	Responsável por ler os parâmetros da técnica <i>MassSpring</i>
<i>ParamEst</i>	Responsável por ler os parâmetros da técnica <i>Anaglyph</i>

Quadro 7 - Descrição das finalidades das classes da *Wizard* (ROSSATO, 2006)

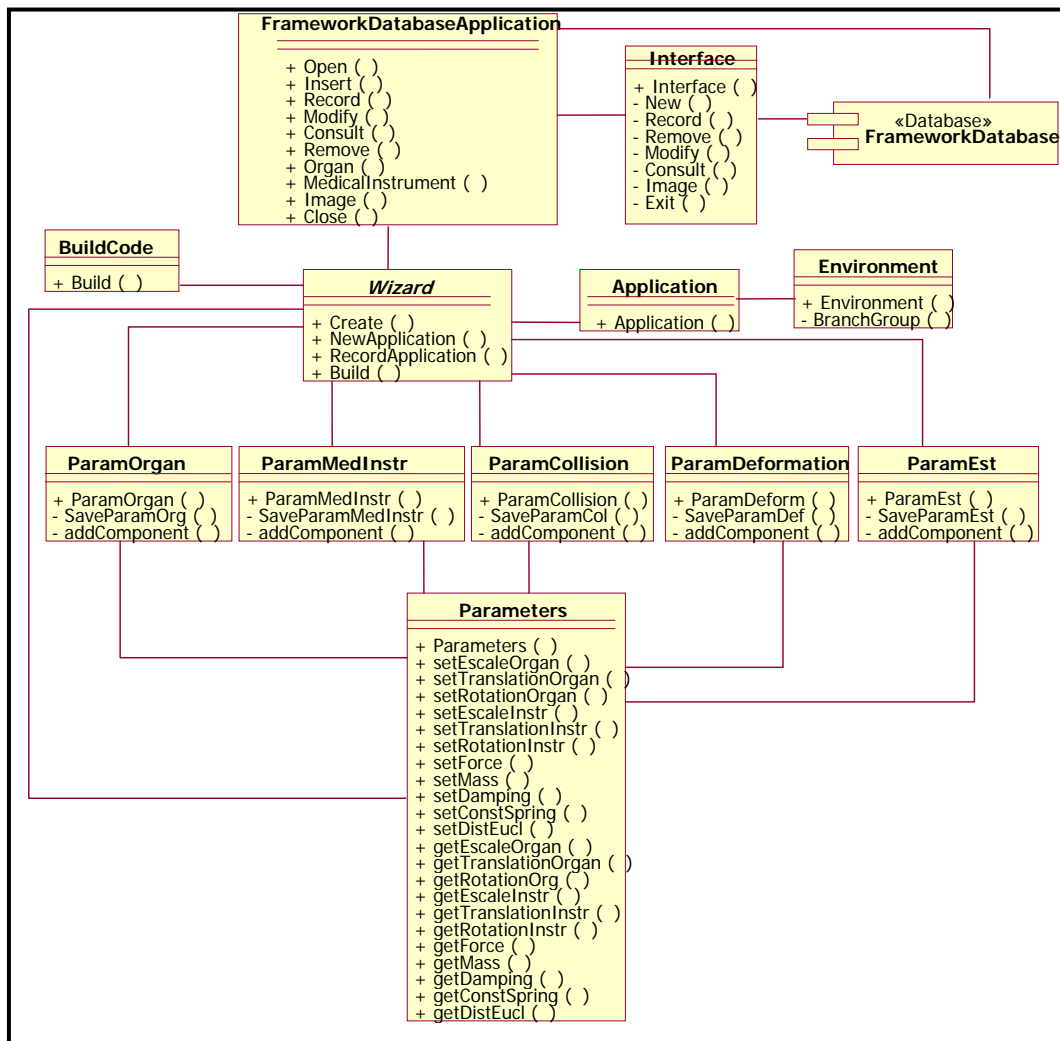


Figura 51 - Diagrama de Classes do BD (ROSSATO, 2006).

Este diagrama foi desenvolvido para a primeira interface implementada, conforme mostrado na Figura 52, porém resolveu-se implementar outra interface mais adequada ao *framework* fazendo com que algumas classes projeto BD se tornassem desnecessárias

6.2 Desenvolvimento da Wizard

Entre os *frameworks* de RV estudados, descritos no Capítulo 4, verificou-se que nenhum possui uma *Wizard*, sendo este um dos diferenciais do *ViMeT*. Como embasamento teórico utilizou-se a *Wizard* construída a partir de um *framework* da área de negócios, descrita em Braga (2002), denominado *GrenWizard*. Foi construído um protótipo da interface da *Wizard* no início do projeto do *ViMeT*, observado na Figura 52.

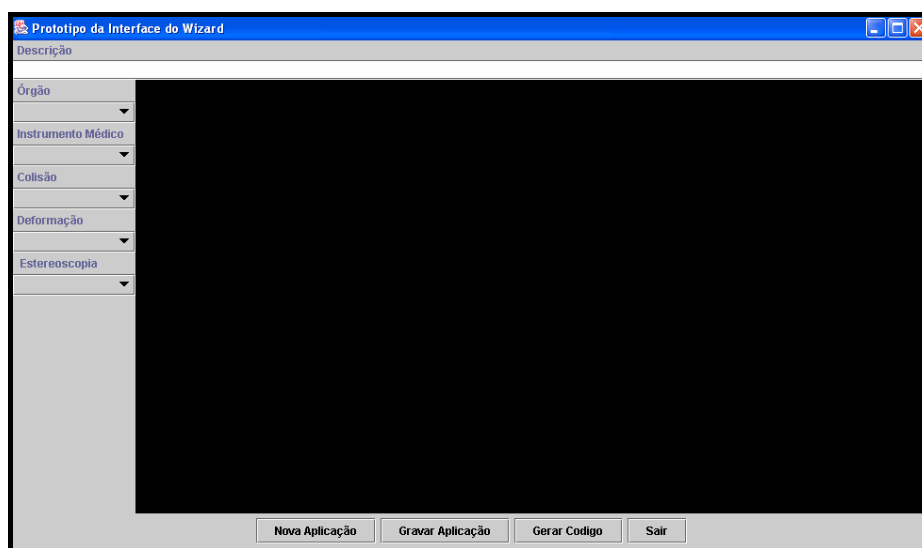
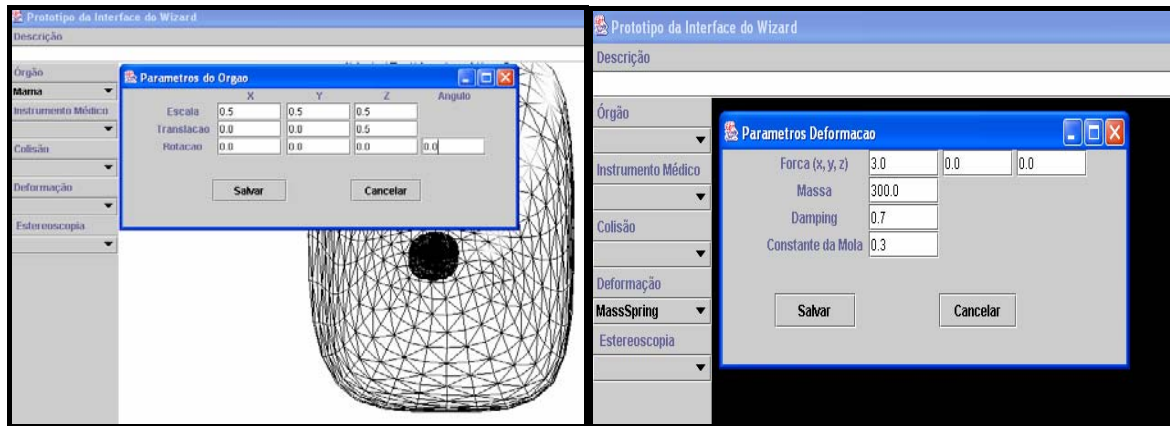


Figura 52 – Protótipo da interface do *Wizard* (ROSSATO, 2006)

Nesta interface o usuário poderia escolher qual órgão e instrumento médico, ambos objetos modelados, definir os parâmetros de escala, translação e rotação para cada um deles, (Figura 53 a), escolher quais as funcionalidades necessárias para as novas aplicações, bem como seus parâmetros (Figura 53 b).

Os botões *Nova Aplicação*, *Gravar Aplicação*, *Gerar Código* possuem, respectivamente, as funções de criar uma nova aplicação, gravar a aplicação no BD e gerar o código-fonte com os parâmetros definidos nesta interface.



(a)

(b)

Figura 53 - (a) Interface de definição dos parâmetros para o órgão, (b) Interface para a definição dos parâmetros da funcionalidade de deformação

No entanto, esta interface não foi utilizada no projeto final, pois ao integrá-la ao *ViMeT* foi percebido que ela não possuía usabilidade ideal, não possibilitava a visibilidade de todas as funcionalidades e parâmetros modificados, além de não disponibilizar a manutenção no Banco de Dados. Ou seja, não era possível ter uma visão geral da aplicação que seria desenvolvida e nem das aplicações já desenvolvidas e gravadas no Banco de Dados. Por esse motivo implementou-se a interface apresentada na Figura 54. Para a implementação desta interface foi criada uma classe *Wizard*, que possui, também, os métodos para instanciar o *ViMeT*.



Figura 54 – Nova Interface da Wizard

Por meio da interface o usuário pode selecionar os objetos que representam órgãos e instrumentos médicos; definir características destes objetos em relação a escala, translação e rotação e definir quais as funcionalidades que farão parte da aplicação.

Com as características e objetos escolhidos o sistema cria o AV, gera código-fonte e compila o código, gerando uma aplicação. Todas as características selecionadas são armazenadas no BD para alterações ou consultas futuras. Com o código-fonte gerado, o usuário pode incluir características particulares da aplicação gerando uma aplicação derivada.

Na interface, o usuário pode navegar por meio das guias. A guia denominada *Aplicação* possui os botões: *Nova*, *Abrir*, *Salvar*, *Gerar Código* e *Gerar Aplicação*. Esta guia está sempre habilitada juntamente com as guias *Documentação* e *Banco de Dados*. As demais guias somente são habilitadas quando o usuário inicia uma nova aplicação ou abre uma aplicação já existente.

Quando o botão *Nova* é selecionado é solicitado o nome da nova aplicação, É realizada uma consulta ao Banco e Dados, a fim de verificar se já existe uma aplicação com o nome fornecido e, caso já exista, é apresentada uma mensagem de erro.

A solicitação de uma nova aplicação inicia a instanciação do *ViMeT*. Como foi discutido na Seção 5.3.4 foi criada uma classe *Application* que possui todos os comandos de importação necessários e os *frozen spots*, sendo os atributos variáveis preenchidos pelos parâmetros fornecidos pelo usuário.

Selecionando-se o botão *Abrir* é disponibilizada uma lista com o nome de todas as aplicações contidas no BD. Selecionando-se uma aplicação, seus parâmetros originais são disponibilizados nas guias *Carrega* e *Funcionalidades*. Os objetos ficam visíveis no AV da *Wizard*, permitindo que o usuário visualize o posicionamento e a escala dos objetos, antes de gerar o código e a nova aplicação. O botão *Salvar* permite a gravação dos dados da aplicação no BD, após terem sido definidos os parâmetros necessários. O botão *Gerar Código* cria um arquivo com a extensão *.java* com todos os parâmetros que foram definidos na *Wizard* e mostra o código gerado na interface.

Como já foi mencionado, para fins de testes, foi criada uma classe *Application* e nela eram feitas as instanciações diretamente das classes do *ViMeT* e os parâmetros eram definidos no código. Para a implementação do método *GerarCódigo* a classe *Application* foi utilizada como um modelo. Este método foi implementado da seguinte forma: foi criado um vetor de *String* e a ele foram atribuídas as linhas da classe *Application*. As linhas que possuíam variáveis foram montadas considerando os parâmetros recebidos na interface *Wizard*. Em seguida, todas as posições do vetor foram impressas em arquivo com extensão *.java*. Na Figura 55 pode ser observado um trecho do código gerado.

```

Código Gerado
public Teste(Canvas3D c) {
    super(c, true);
//Instanciação dos atributos
// - Instanciação dos objetos
    objetos = new Object3D[2];
    objetos[0] = new ObjDef("E:\\ObjetosModelados\\mama_ac.obj",
Object3D.OCTREE + Object3D.DEFORMATION + Object3D.STEREOCOPY, ObjectFile.RESIZE);
    objetos[1] = new ObjRig("E:\\ObjetosModelados\\seringa pronta.obj",
Object3D.STEREOCOPY, ObjectFile.RESIZE);
//Adição dos objetos no universo
    this.add(objetos[0]);
    this.add(objetos[1]);
// - Instanciação da deformação
    Parameters p = new Parameters();
    def = ((ObjDef)objetos[0]).getDeformation();
    p.setForce(3.0f, 0.0f, 0.0f);
    p.setMass(300.0f);
    p.setDamping(0.7f);
    p.setConstSpring(0.3f);
    def.setParameters(p);
// - Instanciação da detecção de colisão
    cd = ((ObjDef)objetos[0]).getCollisionDetector();
    ((Octree)cd).setPrecision(0.0010);
    cd.setCollisionListener(def);
    BranchGroup bgTemp = new BranchGroup();
    bgTemp.addChild(cd);
    super.myLocale.addBranchGraph(bgTemp);
}

```

Figura 55 - Trecho de código-fonte gerado pela Wizard

Gerar Aplicação é o botão responsável por gerar o *bytecode* a partir do arquivo criado pelo botão *Gerar Código*. A aplicação gerada possui as funcionalidades escolhidas na *Wizard*, assim como seus parâmetros, conforme pode ser visto na Figura 56.

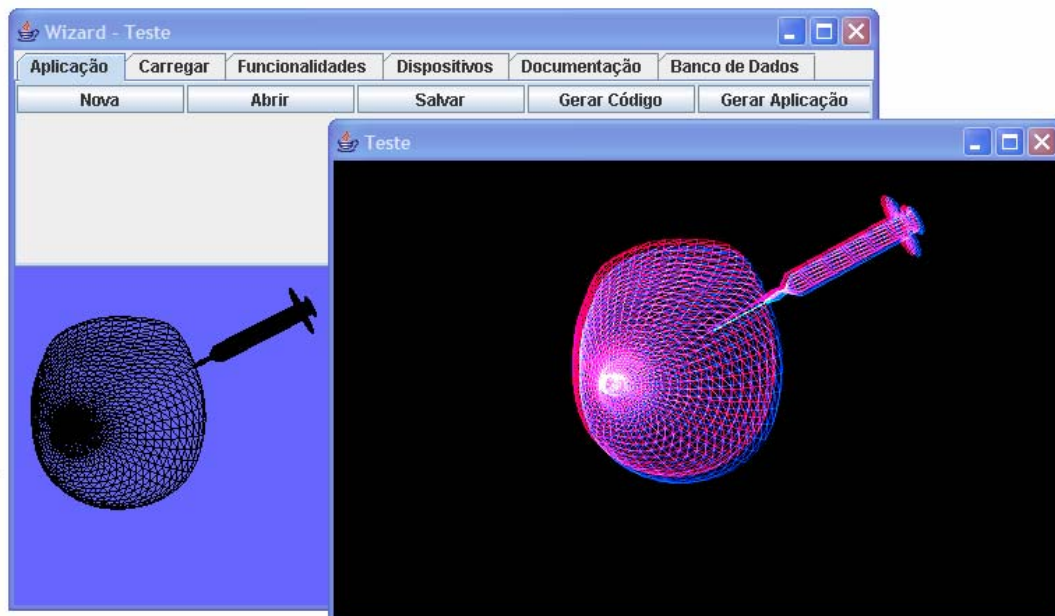


Figura 56 – Aplicação gerada por meio da Wizard

A guia *Carregar* possui duas áreas responsáveis, respectivamente, pelo carregamento do objeto modelado que simula o órgão humano e o instrumento médico, ambos armazenados no Banco de Dados. Por meio da interface, é possível alterar os parâmetros de escala, translação e rotação nos eixos X, Y e Z. No caso da rotação é também possível definir a

medida do ângulo a ser usado. As medidas definidas são armazenadas no BD e podem ser consultadas e alteradas.

Por meio da guia *Funcionalidades* é possível definir quais funcionalidades a nova aplicação irá conter. Para a deformação, os parâmetros que podem ser alterados são: força nos eixos *X*, *Y* e *Z*, constante da mola, massa e *damping* referentes ao método Massa-Mola implementado. Para a detecção de colisão estão disponíveis a técnica *Octrees* e os métodos padrões da *API Java3D*. Para a técnica *Octrees* deve ser definida a distância mínima, descrita na Seção 5.3.3.1. Para a estereoscopia está implementada a técnica de anaglifos e o parâmetro que deve ser definido é a paralaxe.

A guia *Dispositivos* servirá para definir quais dispositivos não convencionais serão utilizados e qual objeto terá interação com este dispositivo. No projeto do *ViMeT* foi prevista a utilização de dispositivos não convencionais, porém sua inserção será implementada futuramente.

Documentação é o nome dado à guia responsável pela documentação do *ViMeT*. O *CookBook* é a forma mais utilizada para a documentação de qualquer *framework*, porém, o projeto do *ViMeT* disponibiliza também o *javadoc* e o Diagrama de Classes.

A guia *Banco de Dados* permite executar a manutenção no BD. O usuário pode inserir novos objetos modelados, fornecendo a sua descrição (nome) e o local de armazenamento. Os botões *Novo*, *Gravar*, *Remover*, *Alterar* e *Consultar* são utilizados para manipular a tabela com os dados dos objetos modelados. O botão *Manutenção* é responsável por permitir alterações na tabela de aplicações gravadas no BD. Quando este botão é selecionado abre-se uma interface que disponibiliza todas as aplicações gravadas no BD. Após a seleção de uma aplicação, seus parâmetros podem ser alterados e regravados. A aplicação pode ainda ser removida do BD.

6.3 Considerações Finais

Este capítulo detalhou a implementação da *Wizard*, apresentando todas as interfaces e explicando o seu funcionamento. Apresentou também o projeto do Banco de Dados, bem como as tabelas, diagramas de classes e quadro descritivo das funções de cada uma das classes. O próximo capítulo apresenta os testes realizados e os resultados obtidos com a utilização do *ViMeT*.

7 RESULTADOS E DISCUSSÕES

Na proposta de desenvolvimento de um *framework* têm-se como resultados esperados a reutilização de classes para gerar novas aplicações, a facilidade de desenvolvimento e a garantia de que as aplicações geradas possuam as características definidas na análise de domínio, por exemplo, detecção de colisão, deformação, estereoscopia, entre outros. Neste trabalho, há a possibilidade de gerar aplicações por meio da *Wizard* ou diretamente das classes do *ViMeT*, particularmente em relação ao *ViMeT*, definiu-se que a forma de instanciação fosse do tipo *White-Box*, isto é sua instanciação é realizada por meio de herança.

Neste capítulo são apresentadas as discussões pertinentes aos resultados obtidos a partir de seis estudos de caso realizados.

7.1 Estudos de Casos

Para avaliar a adequação do *ViMeT* foram realizados seis estudos de caso, divididos em aplicações desenvolvidas diretamente das classes do *ViMeT* (casos 1 e 2) e por meio da *Wizard* (casos 3 a 6). Para a realização dos estudos de caso foram utilizados dois objetos que simulam alguns dos tipos existentes de seringa para o procedimento de punção e três tipos de objetos que simulam órgãos humanos e como podem ser observados, respectivamente, na Figura 57 e na Figura 58.

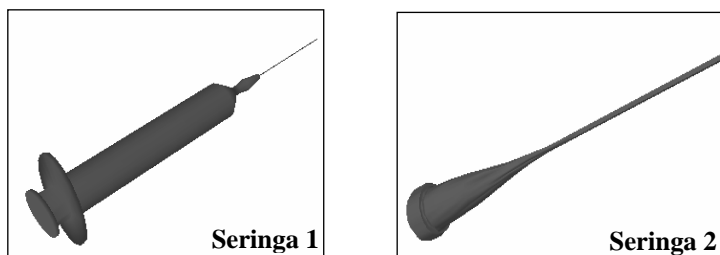


Figura 57 - Objetos modelados que simulam instrumentos médicos

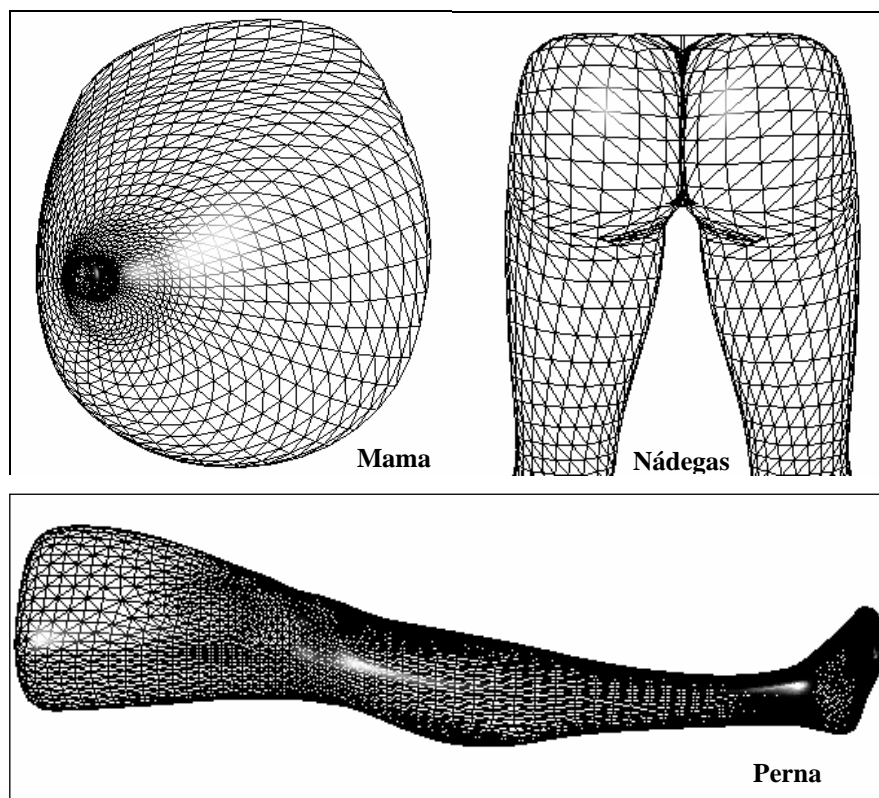


Figura 58 - Objetos modelados que simulam órgãos humanos

O Quadro 8 apresenta as configurações usadas em cada aplicação. Os parâmetros e as funcionalidades utilizadas em cada estudo de caso serão apresentados a seguir.

Estudo de caso	Objeto órgão humano	Objeto equipamento	Colisão	Deformação	Estereoscopia
1	Mama	Seringa 1	<i>Octrees</i>	Sim	Sim
2	Nádegas	Seringa 1	CJava	Sim	Não
3	Mama	Seringa 2	<i>Octrees</i>	Sim	Sim
4	Mama	Seringa 1	CJava	Sim	Não
5	Perna	Seringa 1	<i>Octrees</i>	Sim	Não
6	Nádegas	Seringa 1	<i>Octrees</i>	Sim	Sim

Quadro 8 - Configuração dos estudos de casos

1. Estudo de caso 1

Neste estudo de caso é apresentada uma aplicação desenvolvida instanciando-se manualmente as classes do *ViMeT*. Para a realização dos testes foi implementada uma classe

denominada *Application* e nesta classe são instanciados as classes e métodos diretamente do *ViMeT*. A classe *Application* é sugerida como um *template* para a criação de novas aplicações.

Nesta classe os parâmetros das funcionalidades possuem valores *default*, mas há a possibilidade de alterá-los. Na Figura 59 é apresentado um trecho do código, em que podem ser observados as funcionalidades escolhidas, e a adição dos objetos modelados no AV. A classe gerada, assim como o método construtor, recebem o nome fornecido na interface da *Wizard*, conforme destacado na Figura 59.

```

public ApplicationStereo(Canvas3D c)
{
    super(c, true);

    super.setEyeOffset(0.017F);

    // - Instanciação dos objetos
    objetos = new Object3D[2];
    objetos[0] = new ObjDef("mama_ac.obj", Object3D.OCTREE + Object3D.DEFORMATION +
        Object3D.STEREOCOPY, ObjectFile.RESIZE);
    objetos[1] = new ObjRig("seringaAAF.obj", Object3D.STEREOCOPY, ObjectFile.RESIZE);

    //Adição dos objetos no universo
    this.add(objetos[0]);
    this.add(objetos[1]);
}

```

Figura 59 - Trecho do código do estudo de caso 1

Nesta aplicação foram utilizadas as funcionalidades *default* (deformação (Massa-Mola, detecção de colisão (*Octrees*) e estereoscopia (Anaglifos)) e foram alterados os parâmetros da massa (200), força (1.0, 0.0, 0.0) e a constante da mola (0.1). Para os demais foram mantidos os valores padrões. Na Figura 60 pode ser observado o resultado da aplicação gerada.

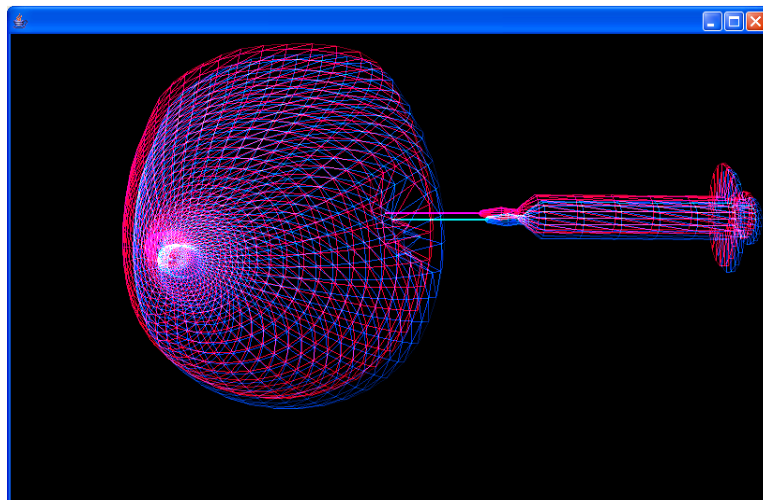


Figura 60 - Aplicação do estudo de caso 1

2. Estudo de caso 2

Neste estudo de caso a aplicação desenvolvida possui apenas as funcionalidades de deformação e detecção de colisão com o método *default* da API Java3D. O objeto que simula um órgão humano agora representa as nádegas. Os parâmetros alterados são: força (1.5, 0.0, 0.0) massa (300), *damping* (0.35) e constante da mola (0.15). Apresenta-se na Figura 61 o resultado da aplicação gerada.

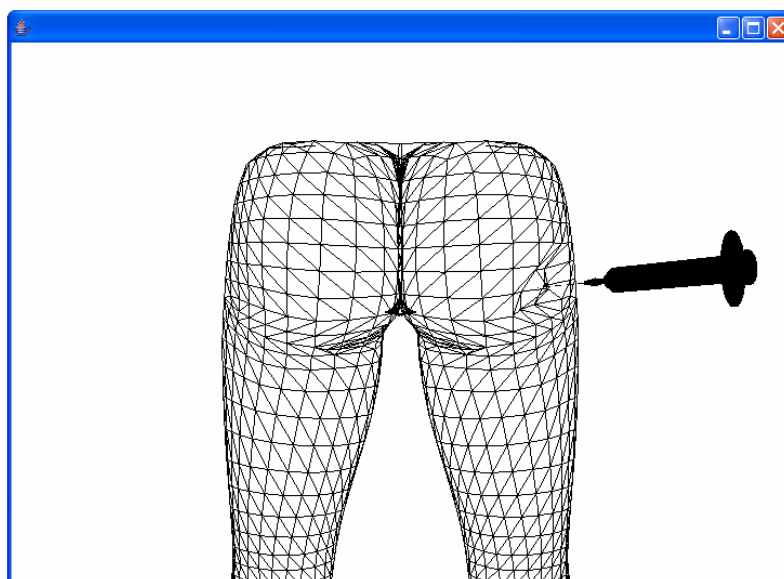


Figura 61 - Aplicação do estudo de caso 2

Nos estudos de caso 1 e 2 não foi utilizado o BD, pois as classes do *ViMeT* foram instanciadas diretamente pelo programador.

Este tipo de instanciação pode ser vista como uma forma de proporcionar a possibilidade de adaptação de algumas características, gerando uma outra aplicação. Salienta-se que mesmo a classe *Application* sendo um *template*, para o uso efetivo do *ViMeT* por meio da instanciação direta (caixa-branca) é recomendado a utilização do *cookbook* (Apêndice A) como apoio para o desenvolvimento de novas aplicações.

3. 3. Estudo de caso 3

Como já foi explicado no Capítulo 6 a *Wizard* também utiliza a classe *Application* como *template*, porém com a utilização da *Wizard* a flexibilidade do *ViMeT* fica mais evidenciada, assim como a facilidade de desenvolvimento. Além disso, proporciona maior

rapidez para a alteração dessas características, bem como o desenvolvimento da nova aplicação.

A aplicação desenvolvida neste estudo de caso foi gerada utilizando as três funcionalidades com seus parâmetros *default*; no objeto que simula a mama não foi utilizada nenhuma transformação e no objeto que simula a seringa com agulha grossa foram utilizados os seguintes valores: para a escala 0.4, 0.8, 0.8, para os eixos x, y, z, respectivamente, e translação 0.2, 0.5, 0.5, respectivamente para os eixos x, y, e z. O resultado pode ser observado na Figura 62.

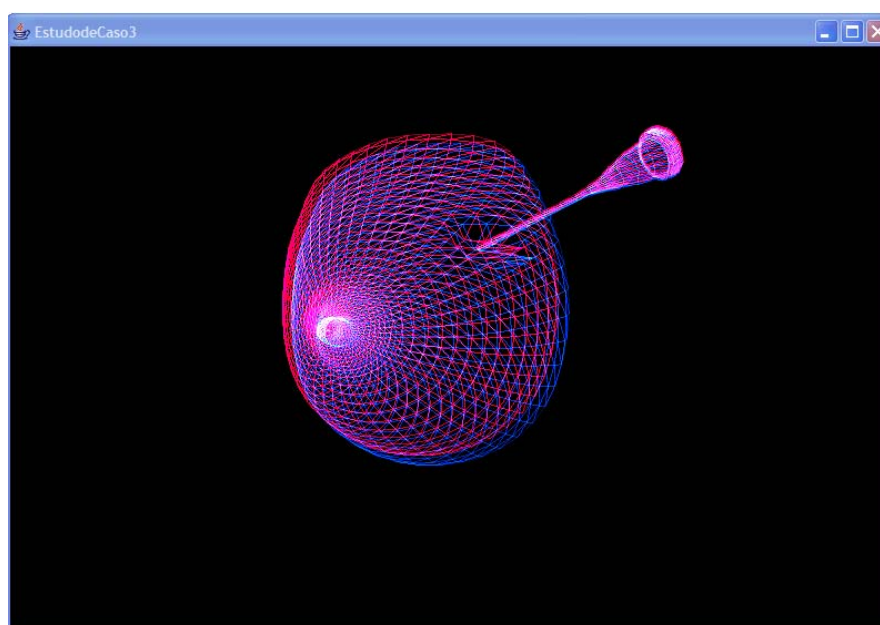


Figura 62 - Aplicação do estudo de caso 3

4. Estudo de Caso 4

No estudo de caso 4 utiliza-se o mesmo objeto que simula a mama, do estudo de caso 3. O objeto que simula o instrumento médico foi modificado. Os valores dos parâmetros foram definidos da seguinte forma: para os eixos x, y e z: escala 0.5, 0.5, 0.5; translação 0.7, 0.5, 1.2, e rotação 0.0, -0.5, 0.0, com valor 0.5 para o ângulo. O resultado da aplicação pode ser observado na Figura 63.

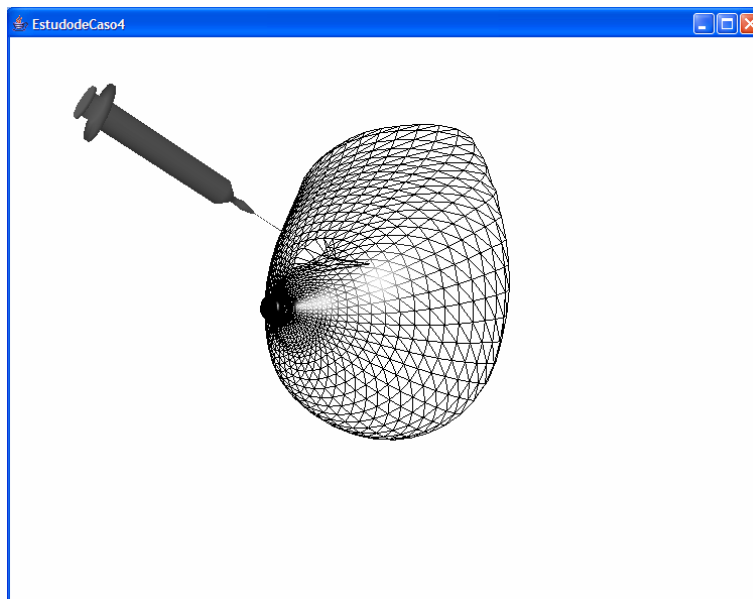


Figura 63 - Aplicação do estudo de caso 4

As alterações nos valores de escala, translação e rotação não afetam o desempenho do sistema. Podem ser utilizados para facilitar a interação e também deixar os tamanhos dos objetos proporcionais. Desta forma, o *ViMeT* ganha independência em relação ao processo de modelagem dos objetos.

Os parâmetros que podem afetar o desempenho do sistema e também o aspecto visual são os parâmetros da técnica Massa-Mola e da *Octrees*. Os testes realizados por Pavarini (2006) utilizam valores empíricos. Não é o foco deste trabalho a avaliação do impacto da variação destes parâmetros.

5. Estudo de caso 5

Para o estudo de caso 5 foram utilizadas apenas as funcionalidades deformação e detecção de colisão (*Octrees*). O objeto modelado que simula uma perna possui um número maior de vértices que os objetos que simulam a mama e as nádegas. Utilizando os parâmetros *default* da técnica Massa-Mola observou-se que ao invés de ocorrer uma depressão dos vértices o resultado foi um relevo, como pode ser observado na Figura 64.

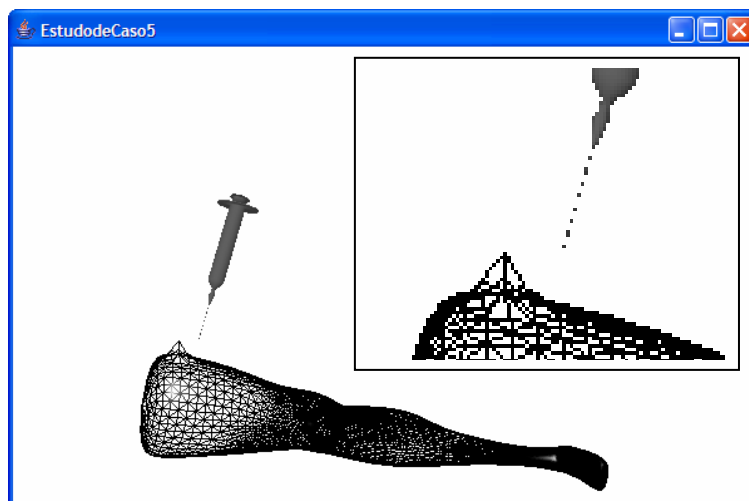


Figura 64 - Primeiro resultado do Estudo de Caso 5

A técnica Massa-Mola calcula os vértices vizinhos do vértice onde houve a colisão a partir de fórmulas matemáticas empregadas para gerar a deformação, utilizando os valores dos parâmetros fornecidos pelo usuário. Desta forma, este resultado evidencia a influência da modelagem e dos parâmetros no resultado da técnica de deformação.

A partir destes resultado foram alterados os valores da força para 0.0, 0.0, 0.4 relacionados aos eixos x,y, e z e *damping* para -0.5 e o resultado pode ser observado na Figura 65. O objeto que simula a perna foi rotacionado no eixo z (2.0) com ângulo 1.5 para ficar inteiramente visível.

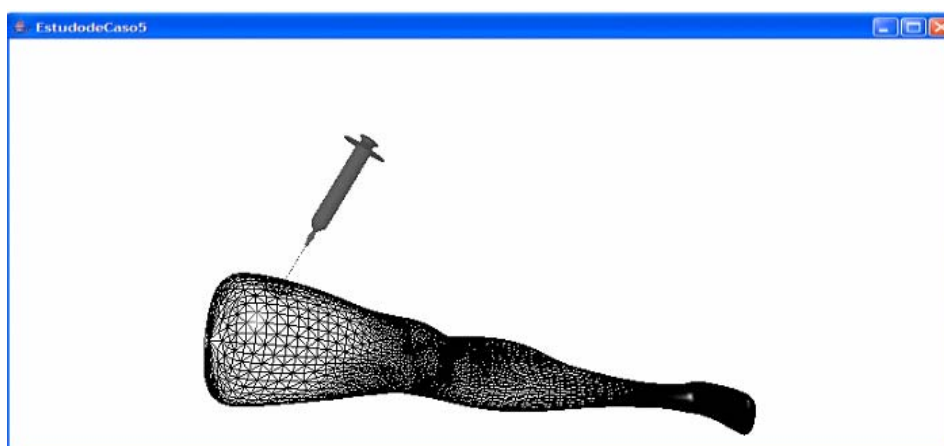


Figura 65 - Aplicação do estudo de caso 5

Como um dos resultados esperados na proposta deste trabalho era a facilidade para desenvolver novas aplicações, na Figura 66 é apresentada a tela de manutenção de aplicações, por meio do qual é possível alterar os valores dos parâmetros desejados. Ela foi utilizada para

alterar os parâmetros da deformação deste estudo de caso 5, até ser obtido um resultado satisfatório.

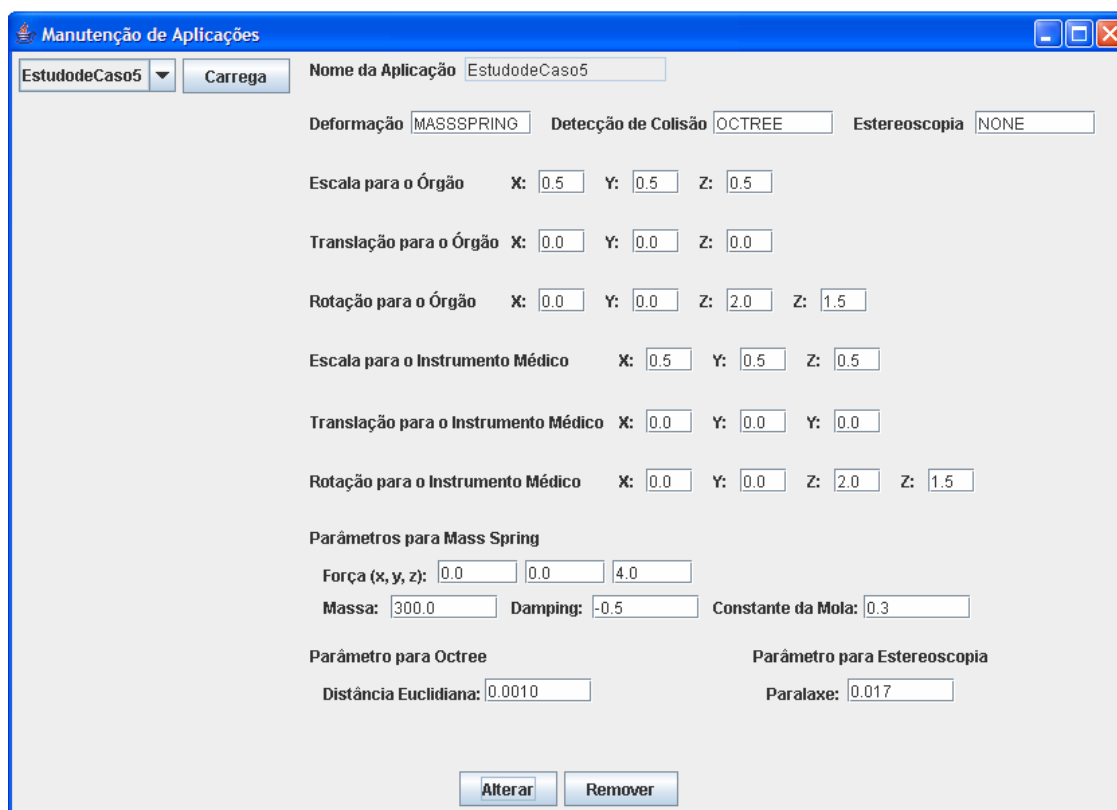


Figura 66- Tela de manutenção com o estudo de caso 5

Este estudo de caso foi de grande importância para enfatizar a relevância da utilização de uma ferramenta *Wizard* em um *framework*. Para melhor visualização do resultado da deformação foi feito um recorte na Figura 65, mostrado na Figura 67.

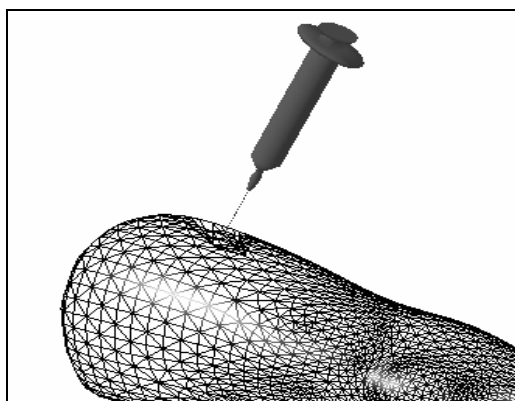


Figura 67 – Resultado do Estado de caso 5 – Recorte

6. Estudo de caso 6

Na aplicação do estudo de caso 6 foram utilizadas as três funcionalidades. No objeto que simula as nádegas nenhum valor foi alterado, já no objeto que simula o instrumento médico foram aplicados os seguintes valores para translação nos eixos x, y, e z : 2.0, 0.5, 0.7.

Para os parâmetros da deformação foi alterado o valor do *damping* de 0.7 para -0.4, a fim de deixar a deformação mais realista (Figura 68).

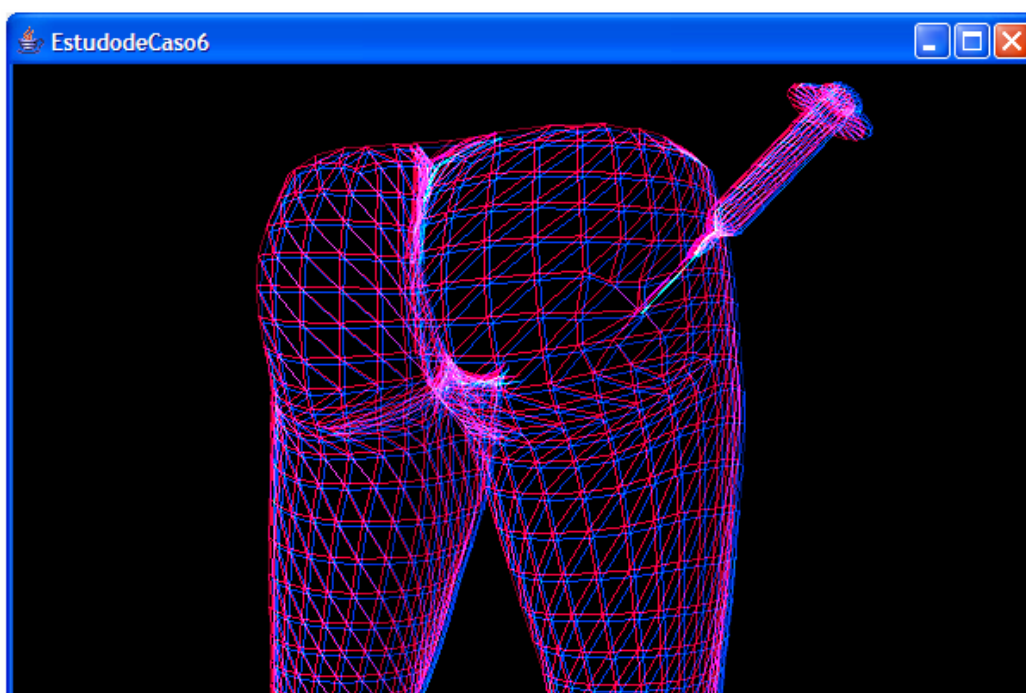


Figura 68 - Aplicação do estudo de caso 6

7.2 Reutilização de Software

A proposta inicial do *ViMeT* era simplesmente reutilizar aplicações previamente implementadas que executavam funcionalidades importantes para a simulação de treinamento médico. Dentro deste contexto, uma primeira discussão que merece ser destacada é a necessidade de planejamento de um *framework*.

As aplicações reutilizadas não haviam sido planejadas para fazer parte de um *framework*. Por isso não possuíam documentação adequada e também estavam muito específicas. A estrutura de classes apresentava forte coesão e acoplamento o que dificultou o

reúso do código. Por esse motivo foram despendidos esforço e tempo para compreender e adaptar os códigos-fonte, a fim de torná-los genéricos.

Como o *ViMeT* será utilizado para o desenvolvimento de novas aplicações de treinamento médico, foi desenvolvido sob o paradigma da Orientação a Objetos. Sua estrutura hierárquica de classes foi planejada prevendo-se a inclusão de novas funcionalidades, novas técnicas para as funcionalidades existentes e, ainda, dispositivos não convencionais.

7.3 Modelagem 3D e Interação

Os objetos 3D que foram utilizados nos estudos de caso foram modelados na ferramenta de modelagem *3D Studio Max* (3D Studio Max, 2006). Por se tratar de um *software* proprietário, utilizou-se uma distribuição educacional. Como o *ViMeT* faz parte dos projetos do LApIS, que tem como um dos objetivos o desenvolvimento de ferramentas de baixo custo, estes mesmos objetos deverão ser modelados utilizando a ferramenta de modelagem 3D Blender (BLENDER, 2006) que é um *software* livre.

Os objetos modelados influenciam diretamente no desempenho da técnica de deformação, pois a obtenção de um resultado mais realístico exige modelagem com um grande número de vértices e arestas.

Como a técnica *Octrees* define a existência de colisão por meio do cálculo da distância euclidiana entre os centros dos dois objetos envolvidos, a modelagem deve ser realizada a partir de um único *shape*, visto que a existência de dois ou mais *shapes* causará a presença de mais de um centro do objeto, dificultando o funcionamento da técnica.

A interação é um fator muito importante para uma aplicação de RV em treinamento médico. No entanto, até a finalização deste trabalho, o LApIS ainda não possuía dispositivos não convencionais. Por isso, a interação ocorre por meio de mouse e teclado.

O objeto que simula o órgão é fixo e não possui interação. Por meio do teclado com a utilização das quatro setas do teclado é possível movimentar a câmera virtual, gerenciada pelo objeto *View* do subgrafo de visualização da *API Java3D* permitindo alterar o ponto-de-vista da cena. Para o objeto que simula o instrumento médico, a interação é feita com o mouse. Com o botão direito do mouse é possível girar o objeto em torno dos eixos x e y e com o botão esquerdo é possível transladar pelos mesmos eixos.

7.4 Processo de Construção do *ViMeT*

No processo de construção do *ViMeT* optou-se por utilizar a metodologia proposta por Bosch (1999). Esta metodologia não possuía instruções para a reutilização de código-fonte de outras aplicações no desenvolvimento de um *framework*. Mas como proporciona a possibilidade de modificar o projeto, de acordo com os progressos da implementação e com a verificação de erros na fase de testes, foi possível empregá-la de maneira satisfatória.

A reutilização de código-fonte de aplicações desenvolvidas isoladamente representou uma grande dificuldade no processo de desenvolvimento, pois estas aplicações não foram planejadas fazer parte do *framework*. Para que a integração acontecesse foi necessária uma padronização no código-fonte, excluindo as classes e métodos responsáveis pela interface, iluminação, importação de objetos, dentre outros.

Diante desta dificuldade o *ViMeT* foi construído de forma a servir de padrão para o desenvolvimento de novas funcionalidades, de forma que sejam desenvolvidas para serem integradas e não como uma aplicação isolada.

O estágio atual do *framework* permite que desenvolvedores futuros preocupem-se somente com implementação de novas funcionalidades, visto que interface e AV já são providos pelo *ViMeT*.

7.5 Vantagens e Limitações

As principais vantagens do *ViMeT* são: facilidade e rapidez de desenvolvimento de novas aplicações no domínio do *framework*; alteração dos valores dos parâmetros utilizados; possibilidade de manipulação das aplicações geradas, por meio da *Wizard* e facilidade de integração de novas classes, que representarão novas funcionalidades e novas técnicas.

Como foi apontado na Seção 3.5 um *framework* orientado a objetos possui vantagens como a diminuição das linhas de código, aumento da qualidade de *software*, empacotamento do conhecimento e reutilização otimizada (análise, código-fonte e testes). Pode-se concluir que no desenvolvimento do *ViMeT* estas vantagens foram confirmadas.

A interação é uma das limitações das aplicações desenvolvidas pelo *ViMeT*, pois para promover a deformação o objeto que simula o instrumento médico deve tocar o objeto que

simula o órgão humano. Na versão atual, esta ação é executada com a utilização do mouse, planejando-se incluir, em um futuro próximo, uma luva de dados e um equipamento háptico.

A modelagem de objetos também é um fator limitante, visto que ainda há a necessidade de representar objetos de forma mais realística, considerando estruturas internas.

7.6 Considerações Finais

Neste capítulo foram apresentados os resultados esperados e obtidos do *ViMeT*. Os estudos de caso realizados comprovaram a flexibilidade e corretitude do *framework* desenvolvido, bem como a facilidade de utilização da *Wizard*.

8 CONCLUSÕES

Neste trabalho foi apresentado o processo completo de desenvolvimento de um *framework* orientado a objetos para treinamento médico utilizando técnicas de RV, incluindo técnicas de deformação, detecção de colisão, estereoscopia, construção dinâmica e manipulação do AV. Para o seu desenvolvimento foi prevista a reutilização de três aplicações previamente desenvolvidas, que representaram as suas principais funcionalidades. Porém, somente duas delas foram utilizadas e uma implementada novamente.

Um dos requisitos ao implementar o *ViMeT* foi deixá-lo com uma estrutura de classes que facilitasse a integração de novas funcionalidades e de outros dispositivos de interação. Este objetivo foi alcançado de maneira satisfatória com auxílio do paradigma de Orientação a Objetos, utilizando-se a linguagem de programação Java e sua *API Java3D*. Desta maneira, comprovou-se que com um *framework* é possível evitar a dependência de um único desenvolvedor.

A *API Java 3D* também colaborou para a redução do número de classes previstas, pois possibilitou a construção de uma estrutura hierárquica de classes, composta por super classes, subclasses e métodos que facilitaram a implementação de AVs com várias funcionalidades já previstas. Um exemplo desta estrutura é a classe *Light*, responsável pela iluminação do AV, que proporcionou a facilidade de somente instanciar os métodos necessários sem ter que implementar uma classe ou método para tratar as luzes do AV gerado.

Nota-se que além da preocupação da construção de um *framework* Orientado a Objetos, há a preocupação de respeitar a hierarquia das classes, imposta pela estrutura do Grafo de Cena. Esta estrutura facilitou a generalização dos parâmetros, tornando dinâmicas as transformações dos objetos, bem com a adição e remoção de objetos no AV.

A *Wizard* facilitou a instanciação do *ViMeT* no que diz respeito a alterações de parâmetros, mostrando que a manipulação destes parâmetros no BD é simples e rápida e não implica em perda do desempenho do *ViMeT*. O código-fonte gerado pela *Wizard* com a extensão *.java* pode ser adaptado por desenvolvedores familiarizados com a linguagem de programação Java para customizar novas aplicações. A aplicação gerada por meio da *Wizard* pode ser gerada inúmeras vezes até ser obtido o resultado esperado.

Uma outra preocupação era quanto ao desempenho das aplicações geradas por meio do *ViMeT*. Verificou-se que o desempenho não é alterado pelo número de funcionalidades, mas sim pela estrutura dos objetos modelados.

Diante dos resultados obtidos por meio de estudos de casos pode-se concluir que o desempenho do *ViMeT* é plenamente satisfatório e que todos os resultados esperados com a sua implementação foram alcançados. No entanto, verifica-se que os testes realizados não foram suficientes e pretende-se realizar novos testes para validação dos parâmetros utilizados. Também serão feitos testes juntamente com profissionais da área da saúde.

Destacam-se como contribuições do *ViMeT*: (1) para a área de treinamento médico – possibilidade de gerar ferramentas de simulação de procedimentos de biópsia de forma eficaz e rápida, sem erros de código, podendo diminuir custos de treinamento a médio e longo prazo; (2) para a área de RV – disponibilização de um conjunto de classes reutilizáveis para criar um AV, manipular um AV e implementar técnicas de deformação, detecção de colisão e estereoscopia em diferentes tipos de aplicações de treinamento médico, ou até mesmo em outras áreas da RV, utilizando-se uma tecnologia de software gratuita; (3) para a área de Engenharia de Software – definição de um processo de criação de *framework* e *Wizard*, aplicados especificamente na área de RV, mas que pode ser estendido para outras áreas de conhecimento, garantindo a padronização e a qualidade do *software* desenvolvido.

Salienta-se, além dos resultados apresentados neste trabalho, que o processo de construção do *ViMeT*, obteve como resultado as seguintes publicações: Oliveira *et al.* (2005), Oliveira *et al.* (2006a), Oliveira *et al.* (2006b), Oliveira *et al.* (2006c) e Oliveira *et al.* (2007).

8.2 Trabalhos Futuros

O *ViMeT* poderá ajudar ainda mais no desenvolvimento de aplicações de treinamento médico quando outras técnicas já estiverem implementadas e integradas a ele. Por enquanto, o foco são os exames de punção, mas ele poderá ser estendido para outros tipos de procedimento e diversos testes poderão ser feitos com as combinações das várias técnicas existentes.

A partir dos resultados obtidos, prevê-se a continuidade deste trabalho por meio das seguintes ações:

- Integração com o sistema de reconstrução 3D a partir de imagens mamográficas desenvolvido por Delfino (2007);
- Integração com o sistema de avaliação do treinamento médico, desenvolvido por Monteiro (2006);
- Integração ao Atlas Virtual de Mamas implementado por Ramos (2005);
- Integração do equipamento háptico e luva como forma de melhorar a interação com o AV;
- Integração de novas técnicas de deformação, detecção de colisão e estereoscopia;
- Incremento de objetos modelados que representem, além da superfície, a estrutura interna de órgãos humanos;
- Estudo da deformação para órgãos humanos, considerando sua estrutura interna e parâmetros realísticos, obtidos a partir da verificação da composição e características físicas dos tecidos humanos;
- Melhoria na simulação da finalização do exame de punção, incrementando a representação da coleta do material obtido no procedimento.

Finalmente, espera-se que por meio do *ViMeT* os desenvolvedores de software, na área de RV possam desenvolver novas aplicações, com menor tempo e esforço, com qualidade e confiabilidade e, ainda, que seja possível tornar as aplicações cada vez mais interativas e imersivas.

REFERÊNCIAS

3D Studio Max – Autodesk 3ds Max - Site Oficial, Disponível em: <<http://www4.discreet.com/3dsmax>>, acesso em: nov. 2006.

AFONSO, J. S. (2000) E-book histeroscopia. Disponível em: <[http://www.histeroscopia.med.br/.](http://www.histeroscopia.med.br/)> Acesso em: 14 dez. 2005.

ALBERIO, M. de V.; OLIVEIRA, J. C. ACONTECe-Cardio: um Ambiente Colaborativo para Treinamento em Cirurgia Cardíaca. In: SVR 2006 – VIII SYMPOSIUM ON VIRTUAL REALITY, 2006, Belém. **Proceedings...**Belém: CESUPA, 2006. p. 397- 408.

APACHE. **Derby Tutorial**. Disponível em: <db.apache.org/derby/papers/DerbyTut/index.html> . Acesso em: dez. 2006

ARDENGH, J. C. **O papel ecoendoscopia no diagnóstico dos tumores neuroendócrinos pancreáticos** Disponível em: <<http://www.siicsalud.com/dato/dat051/06d26000.htm>> Acesso em: jan. 2006.

BALANIUK, R. ; COSTA, I.; MELO, J. Cosmetic Breast Surgery Simulation. In: SVR 2006 – VIII SYMPOSIUM ON VIRTUAL REALITY, 2006, Belém. **Proceedings...**Belém: CESUPA, 2006. p. 387- 396.

BASTOS, T. A.; RAPOSO, A. B.; GATTAS M. Um *Framework* para o Desenvolvimento de Aplicações de Realidade Virtual Baseados em Componentes Gráficos. In: XXV CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO, 2005, São Leopoldo. **Anais...** pp. 213-223.

BASTOS, T. de A.; SILVA, R. J. M. da; RAPOSO, A. B.; GATTAS, M.; ViRAL: Um *Framework* para o Desenvolvimento de Aplicações de Realidade Virtual. In: SVR 2004 - VII SYMPOSIUM ON VIRTUAL REALITY, 2004, São Paulo. **Proceedings...** São Paulo: SBC, 2004. p. 51 – 62.

BENES, J. A.; BUENO, R. P.; PINHO, M. S.; ZANCHET, D. J. Aspectos de Implementação de um Sistema de Simulação Tridimensional para Treinamento e Planejamento de Hepatectomia. In: SVR 2004 - VII SYMPOSIUM ON VIRTUAL REALITY, 2004, São Paulo. **Proceedings...** São Paulo: SBC , p. 377- 379, 2004.

BERTI, C. B.; NUNES, F. L. S. Visualização de informações de bases de imagens médicas utilizando realidade virtual. In: V Workshop de Informática Médica, 2005, Porto Alegre. **Anais...CD-ROM**.

BLENDER, Ferramenta de Modelagem 3D - Site Oficial. Disponível em: <<http://blender.org/cms/Home.2.0.html>>. Acesso em: 27 de março 2006.

BLEZEK, D. J.; ROBB R. A.; MARTIN, D. P. Virtual Reality Simulation of Regional Anesthesia for Training of Residents. In: HICSS 33rd Hawaii international Conference on System Sciences, 2000. **Proceedings...**Washington, DC: IEEE Computer Society, Vol 5, p. 5022.

BOSCH, J.; MOLIN, P.; MATTSSON, M.; BENGTTSSON, P.; FAYAD, M. E. *Framework Problems and Experiences*. In: FAYAD, M.; JOHNSON, R.; SCHMIDT D. **Building Application Frameworks: Object-Oriented Foundations of Framework Design**. Nova Iorque : John Willey and Sons, 1999. p. 55-82.

BOTEGA, L. C. **Implementação de Estereoscopia de Baixo Custo para Aplicações em Ferramentas de Realidade Virtual para Treinamento Médico**. 2005. 105 f. Grau: Monografia (Bacharelado em Ciência da Computação) Centro Universitário Eurípides de Marília – Fundação de Ensino Eurípides Soares da Rocha, Marília, 2005.

BOTEGA, L. C.; NUNES, F. L. S. Implementação da Estereoscopia de baixo custo para aplicações em ferramentas de Realidade Virtual para treinamento médico. In: SIBGRAPI 2005. **Anais...** CD-ROM.

BRAGA, T. V. **Um Processo para Construção e Instanciação de Frameworks baseados em uma linguagem de Padrões para um Domínio Específico**. São Paulo : Instituto de Ciências Matemáticas e de Computação, 2002, 232 f. Grau: Tese (Doutorado em Ciência da Computação e Matemática Computacional) Instituto de Ciências Matemáticas e de Computação. São Carlos, 2002.

CHOI, K. S.; SUN, H.; HENG, P. A.; CHENG, J. C. Y. Scalable Force Propagation Approach for the Web – Based Deformable Simulation on Soft Tissues. In: Web3D'02 ACM, 2002. **Proceedings...** pp.185-193.

CONTI, F. Programas, Funções e tipos. Disponível em: <<http://www.cultura.ufpa.br/dicas/progra/protipos.htm>>. Acesso em: 12 dez. 2005.

COPLIEN, J. O. Software design patterns: common questions and answers. In: L. Rising – The Patterns HandBook: Techniques, Strategies, and Applications, Cambridge University Press, p. 311-330, 1998.

DELFINO, S.; NUNES, F. L.S. Geração de casos de estudo para treinamento médico virtual a partir de imagens reais. In: II Workshop de Aplicações em Realidade Virtual, 2006, Recife. **Anais...** CD-ROM.

DEITEL, H.M. e DEITEL, P.J. **Java Como Programar**. 6º ed. São Paulo: Pearson, 2005.

Diagnóstico da América. **Índice de exames de apoio ao diagnóstico** Disponível em: <http://www.diagnosticosdaamerica.com.br/exames/citopatologia_aspirativa.shtml> Acesso em: dez. 2006.

FAYAD, M. E.; SCHMIDT, D. C. Object-oriented Application *frameworks*. In: Communications of the ACM, vol.40, p.10, 1997.

FAYAD, M.; JOHNSON, R.; SCHMIDT D. **Building Application Frameworks: Object-Oriented Foundation of Frameworks Design**. Nova Iorque: John Wiley & Sons, 1999.

FREIBERGER, E. C. **Suporte ao uso de Frameworks Orientados a objetos com base no histórico do desenvolvimento de aplicações**. 2002. 262 f. Grau: Dissertação (Mestrado em Ciência da Computação) Universidade Federal do Rio Grande do Sul, Porto Alegre, 2002.

FREITAS Jr, R. **Punção Aspirativa por Agulha Fina: Estudo Comparativo entre dois diferentes Dispositivos para a Obtenção da Amostra Citológica**. 2001. 136 f. Grau: Tese (Doutorado em Tocoginecologia) - Faculdade de Ciências Médicas Universidade Estadual de Campinas, Campinas. Disponível em:< <http://www.rau-tu.unicamp.br/nou-rau/demopt/document/?view=57>>. Acesso em: 12 dez. 2005.

FREITAS, C. M. D. S.; MANSSOUR, I. H.; NEDEL, L. P.; GAVIÃO, J. K.,PAIM, T. C.; MACIEL, Â. *Framework* para Construção de Pacientes Virtuais: Uma aplicação em Laparoscopia Virtual. In: SVR 2003 - SBC SYMPOSIUM ON VIRTUAL REALITY, 2003, Ribeirão Preto. **Proceedings...** Porto Alegre: SBC, 2003, v. 6, p. 283-294.

FROEHLICH, G.; HOOVER H. J.; LIU, L.; SORENSON, P. G.. Hooking into Object-Oriented Application *Frameworks*. In: International Conference on Software Engineering, 1997, Boston. **Proceedings ...** May 17-23.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. **Design Patterns: Elements of Reusable Object- Oriented Software**. Addison-Wesley, 1995

GUARALDI, S.; SÁ, E.; ROMANO, S.; CARVALHO, A. C. P. (2005) **Usefulness of echoendoscopy in the diagnosis of primary cystic neoplasms of the pancreas**. Disponível em: <http://www.scielo.br/scielo.php?pid=S0100-39842005000600014&script=sci_pdf&tlng=pt> Acesso em: fev. 2007.

GREENLEAF, W. Medical Applications of Virtual Reality. **Overview February 2004**. Disponível em: <<http://www.greenleafmed.com/publications/VR%20Med%20overview.pdf>> Acesso em : novembro de 2005.

GROSS, M. H. ; SPRENGER, T. C. , FINGER, J. Visualizing Information on a Sphere. In: IEEE Information Visualization, 1997. **Proceedings...** p. 11-16.

HERMOSILLA, L. G. **Sistema de Realidade Virtual para geração dinâmica de estruturas de feto**, 2004. 120 f. Grau: Dissertação (Mestrado em Ciência da Computação) – Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2004.

HERMOSILLA, L. G., NUNES, F. L. S. Proposta para construção de biótipo brasileiro para representar fetos virtuais. In: V WORKSHOP DE INFORMÁTICA MÉDICA, 2005, Porto Alegre. **Anais...** CD-ROM.

HINKENJANN, A.; MANNUB, F. *basho* – A Virtual Environment *Framework*. In: SVR 2004 - VII Symposium on Virtual Reality, 2004, São Paulo. **Proceedings...** São Paulo: SBC, p. 344 – 346.

Immersion (2005a), Immersion Medical. **Endovascular AccuTouch® Simulator**. Disponível em: <http://www.immersion.com/medical/products/endovascular/?m=nav_2_3>. Acesso em: 1 nov. 2005.

Immersion (2005b), Immersion Medical. **Endoscopy AccuTouch® System**. Disponível em: <http://www.immersion.com/medical/products/endoscopy/?m=nav_2_2>. Acesso em: 1 nov. 2005.

Immersion (2005c), Immersion Medical. **Laparoscopic Surgical Workstation**. Disponível em: <http://www.immersion.com/medical/products/laparoscopy/?m=nav_2_5>. Acesso em: 1 nov. 2005.

JACOBSON, L. **Realidade virtual em casa**. Rio de Janeiro: Berkeley, 1994.

JOHNSON, R. E. **How to Design Frameworks**. OOPSLA 1993

JOHNSON, R. E.; RUSSO, V. **Reusing object-oriented designs**. Relatório Técnico UIUCDCS 91 – 6196, University of Illinois, 1991.

JOHNSON, R. E. Componentes, *Frameworks*, Patterns. In: SYMPOSIUM ON SYMPOSIUM ON SOFTWARE REUSABILITY (SST'97), 1997. **Proceedings...**

JOHNSON, R. E.; FOOTE, B. **Designing reusable classes**. Journal of Object Oriented Programming, v.1, n.2, p.22-35, 1988.

KERA, M. **Detecção de colisão utilizando hierarquias em ferramentas de realidade virtual para treinamento médico**. 2005 92 f. Grau: Monografia (Bacharelado em Computação) Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2005. Disponível em: < <http://lost.fundanet.br/~kera>>. Acesso em 20 dez. 2005

KIRNER, C. Apostila do ciclo de palestras de realidade virtual, Atividade do Projeto AVVIC-CNPq (Protem - CC - fase III) - DC/UFSCar, São Carlos, p. 1-10, Out., 1996.

KIRNER, T.G.; MARTINS, V.F. **A Model of Software Development Process for Virtual Environments**. In: 2ND IEEE INT. SYMPOSIUM ON APPLICATION-SPECIFIC SYSTEMS AND SOFTWARE ENGINEERING AND TECHNOLOGY, 1999. **Proceedings...** p. 155-161.

LIMA, L. de. **Protótipo de Ferramenta de Realidade Virtual para Simulação do Exame de Punção da Mama**. 2004. 81 f. Grau: Dissertação (Mestrado em Ciência da Computação) - Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2004.

LINEBARGER, J. M.; JANNECK, C. D.; KESSLER, G. D. Shared Simple Virtual Environment: An Object-Oriented *Framework* for Highly-Interactive Group Collaboration. In: 7th IEEE DS-RT Conference, 2003. **Proceedings...** p. 170-180. Disponível em: < <http://www.cse.lehigh.edu/~dkessler/Publications/LinebargerDS-RT2003.pdf> >. Acesso em: 9 set. 2005

LIU, A.; TENDICK, F.; CLEARY, K.; KAUFMANN, C. .A Survey of Surgical Simulation: Application, Technology and Education. In: MIT Press, vol. 12, Dezembro, 2003.

LOGITHEC. **3D Motion Controllers from 3Dconnexion**. Disponível em: <<http://www.logitech.com/index.cfm/products/3rdparty/US/EN,crid=1720,categoryid=298>>. Acesso em: 15 jan. 2005.

LUZ, R. W.; HERMOSILLA, L. G.; NUNES, F. L. S.; DELAMARO, M. E.; SEMENTILLE, A. C.; BREGA, J. R. F.; RODELLO, I. A. Aplicação de técnicas de realidade virtual para representação de fetos a partir de imagens bidimensionais de ultra-som. In: IV Workshop de Informática Médica, 2004, Brasília. **Anais...** CD-ROM.

MACHADO, L. S. **A Realidade Virtual no Modelamento e Simulação de Procedimentos Invasivos em Oncologia Pediátrica: Um Estudo de Caso no Transplante de Medula Óssea**. 2003, 130 f. Grau: Tese (Doutorado em Engenharia) Departamento de Engenharia de Sistema Eletrônicos da Escola Politécnica da Universidade de São Paulo, 2003. Disponível em: <<http://www.teses.usp.br/teses/disponiveis/3/3142/tde-07052003-123257/>> . Acesso em: 10 ago. 2005.

MACHADO, L. S.; CAMPOS, S. F; CUNHA, Í. L; MORAES, R. M. de. **Cybermed: Realidade Virtual Para Ensino Médico**. In: III CONGRESSO LATINO-AMERICANO DE ENGENHARIA BIOMÉDICA, 2004, João Pessoa. **Proceedings...** p. 573-576.

MATTSSON, M. **Object-Oriented Frameworks – A survey of methodological issues**. 1996, 128 f. Grau: Tese (Doutorado em Ciência da Computação) Department of Computer Science and Business Administration, UCK, Ronneby, 1996. Disponível em: <<http://www.ipd.bth.se/michaelm/paper/Mattsson.Lic.thesis.pdf>>. Acesso em: 15 ago. 2005.

MONTEIRO, B. S. (2006a). **Foto do VirtWall** [mensagem pessoal]. Mensagem recebida por <anatilgol@uol.com.br>em 4 de mai. 2006.

MONTEIRO, B. S.; VALDEK, M. C. O.; CUNHA, Í. L., MORAES; R. M., MACHADO, L. S. AnatomI: 3D: Atlas Digital Baseado em Realidade Virtual para Ensino de Medicina. In: SVR 2006 – VII Symposium on Virtual Reality, 2006, Belém. **Proceedings...** Belém: CESUPA, 2006. p. 13-14.

MONTEIRO, B. (2006b). **Implementação de Módulo de Avaliação da Aprendizagem em uma Ferramenta Virtual para Treinamento Médico**. 2006. 99 f. Grau: Monografia (Bacharelado em Ciência da Computação) Centro Universitário Eurípides de Marília – Fundação de Ensino Eurípides Soares da Rocha, Marília, 2006.

MONTGOMERY, K.; HEINRICH, L.; BRUYNS, C.; WILDERMUTH, S.; HASSER, C.; OZENNE, S.; BAILEY, D. Surgical Simulator for Hysteroscopic: A Case Study of Visualization in Surgical Training. In: 12th IEEE VISUALIZATION CONFERENCE, 2001 San Diego-CA. **Proceedings...**

MORAES, R. M.; MACHADO, L. S.; SOUZA, A. C. M. VirtWall: A Concept of Low-Cost Virtual Wall for Immersion in Virtual Reality. In: SVR'2003 – VII SYMPOSIUM ON VIRTUAL REALITY, 2006, Ribeirão Preto. **Proceedings...** p. 383-385.

NETTO, A. V.; MACHADO, L.; OLIVEIRA, M.C.F. **Realidade Virtual – Definições, Dispositivos e Aplicações.** Revista Eletrônica de Iniciação Científica. Publicação da Sociedade Brasileira de Computação, Vol 2, No. 1, 2002.

NINISS, H.; INOUE, T. Electric Wheelchair Simulator for Rehabilitation of Persons with Motor Disability. In: SVR 2006 – VIII SYMPOSIUM ON VIRTUAL REALITY, 2006, Belém. **Proceedings...** Belém: CESUPA, 2006, p.51-62.

NLM (1990) - NATIONAL LIBRARY of MEDICINE . **Projeto Visible Human.** Disponível em: < http://www.nlm.nih.gov/research/visible/visible_human.html >. Acesso em : 2 nov. 2005.

OLIVEIRA, A. C. M. T. G.; NUNES, F. L. S. Concepção e Implementação de um *Framework* para simulação de exames de punção usando Realidade Virtual. In: IX SVR 2007 – Symposium on Virtual and Augmented Reality, 2007, Petrópolis. **Proceedings...** (aceito para publicação)

OLIVEIRA, A. C. M. T. G.; NUNES, F. L. S.; GAGNIN, M. I.(2006c) Processo de construção de *frameworks* de realidade virtual: uma experiência a partir de ferramentas para treinamento médico. In: II WORKSHOP DE APLICAÇÕES DE RV, 2006, Recife. **Anais...** CD-ROM.

OLIVEIRA, A. C. M. T. G.; PAVARINI, L.; NUNES, F. L. S.; BOTEAGA, L. C.; JUSTO, D. R.; BEZERRA, A. (2006b) Virtual Reality *Framework* for Medical Training: Implementation of a deformation class using Java. In: *ACM SIGGRAPH INTERNATIONAL CONFERENCE ON VIRTUAL-REALITY CONTINUUM AND ITS APPLICATIONS IN INDUSTRY*, 2006, Hong Kong. **Proceedings...** Nova York: ACM Press, 2006. p. 347-351.

OLIVEIRA, A. C. M. T. G.; NUNES, F. L. S.; PAVARINI, L.; BOTEAGA; L. C. (2006a) Desenvolvimento e Implementação de *Framework* de Realidade Virtual para treinamento médico. In: VI Workshop de Informática Média, 2006, Vilha Velha. **Anais...** CD-ROM.

OLIVEIRA, A. C. M. T. G.; NUNES, F. L. S.; PAVARINI, L.; BOTEAGA; L. C. ; KERA, M. Um *framework* para desenvolvimento de aplicações de realidade virtual para treinamento médico. In: II Workshop de Aplicações de RV, 2005, Uberlândia. **Anais...** CD-ROM.

OSFIELD, R.; BURNS, D. **Open Scene Graph.** Disponível em: < <http://www.openscenegraph.org/>>. Acesso em: 12 ago. 2005.

PAIVA, J. G. de S.; CARDOSO, A.; LAMOUNIER, E. Interface for Virtual Automotive route creation in driving phobia treatment. In: SVR 2006 – VIII SYMPOSIUM ON VIRTUAL REALITY, 2006, Belém. **Proceedings...**Belém: CESUPA, 2006. p. 27-38.

PAVARINI, L.; NUNES, F. L. S.; BOTEAGA, L. C.; RAMOS, F. M.; BEZERRA, A. Proposta de implementação de Deformação em Ferramentas Virtuais de Treinamento Médico. In: II Simpósio de Instrumentação e Imagens Médicas, 2005, São Pedro. **Anais...** .

PAVARINI, L. **Estudo e Implementação do Método massa-mola para Deformação em Ambientes Virtuais de Treinamento Médico usando a API Java 3D.** 2006. 147f. Dissertação (Mestrado em Ciência da Computação) - Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2006.

PAVARINI, L.; NUNES, F. L. S.; OLIVEIRA, A. C. M. T. G.; BOTEAGA; L. C.; BEZERRA, A. DefApliMed – Sistema de Deformação para Aplicações Médicas, com base no método Massa-Mola, utilizando a API Java 3D . In: SVR 2006 – VIII SYMPOSIUM ON VIRTUAL REALITY, 2006, Belém. **Proceedings...**Belém: CESUPA, 2006. CD-ROM.

PREE, W. Hot-spot driven development. In: FAYAD, M.; JOHNSON, R.; SCHMIDT D. **Building Application Frameworks: Object-Oriented Foundation of Frameworks Design.** Nova Iorque: John Wiley & Sons, 1999, p. 379-393.

PREE, W.; POMBERGER, G.; SCHAPPERT, A.; SOMMERLAD, P. Active Guidance of *Framework* Development. In: *Software-Concepts and Tools.* Springer-Verlag, 1995, p. 94 - 103.

RAMOS, F. M.. **Aplicação de Realidade Virtual para construção de Atlas de Anatomia e Fisiopatologia do Câncer de Mama.** 2005. 83 f. Grau: Dissertação (Mestrado em Ciências da Computação) – Centro Universitário Eurípides de Marília, Marília, 2005.

RIQUELME, F. **Estudo comparativo de tecnologias de *software* para Realidade Virtual.** 2005. Grau: Dissertação (Mestrado em Ciência da Computação) - Fundação de Ensino Eurípides Soares da Rocha, Marília, 2005.

ROBERTS, D.; JOHNSON R. Evolving *Frameworks*: A Pattern Language for Developing Object-Oriented *Frameworks*. In: *Pattern Languages of Programs*, 1996, Illinois. **Proceedings...** Disponível em: <<http://citeseer.ist.psu.edu/roberts96evolving.html>>. Acesso em: 3 nov. 2004.

RODRIGUES, M. A. F.; SILVA, W. B.; NETO, M. E. B. Um Sistema de Realidade Virtual para tratamento ortodôntico. In: SVR 2006 – VIII SYMPOSIUM ON VIRTUAL REALITY, 2006, Belém. **Proceedings...**Belém: CESUPA, 2006. p. 433-444.

ROSSATO, D. J. (2006). **Banco de Dados e Wizard para um *Framework* de Realidade Virtual para Treinamento Médico**. 2006. 106 f. Grau: Monografia (Bacharelado em Ciência da Computação) Centro Universitário Eurípides de Marília – Fundação de Ensino Eurípides Soares da Rocha, Marília, 2006.

SANTOS, E.T. Uma Proposta para uso de Sistemas Estereoscópicos Modernos no Ensino de Geometria Descritiva e Desenho Técnico. In: *Graphica* 2000, Ouro Preto. p. 1-8. <Disponível em: http://docentes.pcc.usp.br/toledo/pdf/graphica2000_estereo.pdf>. Acesso em: 12 ago. 2005.

SCHMID, H. A. *Framework* design by systematic generalization. In: FAYAD, M.; JOHNSON, R.; SCHMIDT D. **Building Application Frameworks: Object-Oriented Foundation of Frameworks Design**. Nova Iorque :John Wiley & Sons, 1999, p. 353-378.

SCHMID, H. A. Systematic *framework* design by generalization. In: Communications of the ACM, v. 40, n.10, p.48-51, 1997.

SENSABLE TECHNOLOGIES. Disponível em: < <http://www.sensible.com/haptic-phantom-omni.htm>>. Acesso em: 5 fev. 2007.

SENSE8. **Now in its 9th generation, WorldToolkit is the leading cross- platform real-time 3D development tool**. Disponível em: < <http://sense8.sierraweb.net/products/wtk.html>>. Acesso em: 12 nov. 2005.

SILVA, R. P. **Suporte ao desenvolvimento e uso de *frameworks* e componentes**. 2000. 262 f. Grau: (Doutorado em Ciência da Computação). Universidade Federal do Rio Grande do Sul, Porto Alegre, 2002. Disponível em: <<http://www.inf.ufsc.br/~ricardo/download/tese.pdf>>. Acesso em: 12 jun. 2005.

SPRENGER, T. C.; GROSS, M.; BIELSER, D.; STRASSER, T.**IVORY - An Object-Oriented *Framework* for Physics-Based Information Visualization in Java**. In: IEEE Symposium on Information Visualization (InfoViz'98), 1998. **Proceedings...** IEEE CS Press, p. 79-86.

SUN. (2006a). **The Java™ Tutorial**. Disponível em: < <http://java.sun.com/docs/books/tutorial>>. Acesso em: jun. 2006.

SUN. (2006b). **Java 3D API Tutorial**. Disponível em: < <http://java.sun.com/developer/onlineTraining/java3d/>> . Acesso em: dez. 2006.

TALIGENT. **Building object-oriented framework**. Taligent Inc. Write paper, 1994, Disponível em : <<http://www.taligent.com>>. Acesso em : 16 ago. 2005.

TORI, R.; KIRNER, C. Fundamentos de Realidade Virtual e . In . **Fundamentos e Tecnologia de Realidade Virtual e Aumentada**. Livro do Pré-Simpósio SVR 2006. Belém : CESUPA, 2006, Cap. 1, p.2-21.

TRAMBEREND, H. Avango: A Distributed Virtual Reality *Framework*. In: Afrigraph '01, ACM, 2001. **Proceedings....**

TROLLTECH. **Qt Signals and Slots**. Disponível em: <<http://doc.trolltech.com/3.3/signalsandslots.htm>> .Acesso em: 10 set. 2005.

VINCE, J. Introduction to Virtual Reality. Springer-Verlag, Germany, 2004.

Virtual Realities (2006a). **Data Gloves**. Disponível em: < <http://www.vrealities.com/5dtglove5.html> > . Acesso em: 15 jan. 2006.

Virtual Realities (2006b). **HMD**. Disponível em: < <http://www.vrealities.com/5dt.html>> . Acesso em: 15 jan. 2006

WAGNER, C.; SCHILL, M. A.; MÄNNER, R. Collision Detection and Tissue Modeling in a VR-Simulator for Eye Surgery. In: EIGHTH EUROGRAPHICS WORKSHOP ON VIRTUAL ENVIRONMENTS, 2002, Barcelona. **Proceedings....**

WEBSTER, R. ; SASSANI, J. ; SHENK, R.; HARRIS, M., GERBER, J.; BENSON, A. , BLUMENSTOCK, J.; BILLMAN, C. ; HALUCK, R. Simulating the Continuous Curvilinear Capsulorhexis Procedure During Cataract Surgery on the EYESI™ System In: Medicine Meets Virtual Reality, 2005, Long Beach. IOS Press, p. 592- 595.

WEBSTER, R.; ZIMMERMAN, D.; MOHLER, B.; MELKONIAN, M.; HALUCK, R. A Prototype Haptic Suturing Simulator. In: Medicine Meets Virtual Reality - Studies in Health Technology and Informatics,2001. IOS Press, n. 81, p. 567-569.

WELFER, D.; d'ORNELAS, M. C. Cooperação entre Padrões de Projetos na Resolução de Problemas de Processamento de Imagens Baseados em Filtros de Convolução. In: SugarLoafPLoP'2005, 5th Latin American Conference on Pattern Languages of Programing, Campos do Jordão, 2005.

GLOSSÁRIO

Termos Médicos

Angioplastia coronária transluminal percutânea: consiste na inserção de um cateter com um balão na ponta na artéria coronária (vaso sanguíneo que leva sangue para o coração). Quando o balão é expandido, causa o alargamento dos vasos sanguíneos estreitados.

Colonoscopia: estudo endoscópico do intestino grosso, no qual, o colonoscópio é introduzido pelo ânus.

Endoscopia: exame que permite a visualização das cavidades internas do organismo por meio de tubos flexíveis de fibra ótica denominados de broncoscópios, colonoscópios e endoscópios. Tem finalidade diagnóstica e terapêutica, pois permite visualizar as estruturas anormais no interior das cavidades, colher materiais para exames e retirar corpos estranhos ou tumores pequenos.

Extração Extracapsular do Cristalino: retirada do cristalino preservando sua cápsula posterior e implante de lente intra-ocular de câmara posterior.

Fisiopatologia: comprometimento funcional observado nas doenças; a alteração na função diferente dos defeitos estruturais.

Gastrosopia: inspeção da superfície interna do estômago por meio de um endoscópio.

Hepatectomia: operação que retira o fígado ou parte dele.

Histeroscopia diagnóstica e cirúrgica: exame endoscópico da cavidade uterina, mediante o uso de equipamento especial que permite a obtenção de imagens televisionadas, além de possibilitar a realização de biópsia.

Laparoscopia: procedimento cirúrgico mediante o qual se introduz através de uma pequena incisão na parede abdominal, torácica ou pélvica, um instrumento de fibra ótica que permite realizar procedimentos diagnósticos e terapêuticos.

Ortodontia: especialidade odontológica que se ocupa das correções morfológicas posicionais e funcionais dos dentes.

Sigmoidoscopia: bisualização direta do cólon sigmóide através de endoscopia.

Sutura: operação que consiste em coser os lábios de uma ferida.

APÊNDICES

APÊNDICE A – Cookbook

Este documento pode ser visto no seguinte endereço:
(http://galileu.fundanet.br/bcc_bsi/bcc/lapis/projetos/ana/cookbook.php).

COOKBOOK MANUAL DE INSTANCIÇÃO DO *FRAMEWORK ViMeT*

VERSÃO 1.0

Ana Cláudia Melo Tiessi Gomes de Oliveira

Fevereiro 2007

UNIVEM – Marília – SP – Brasil

Sumário

1 Introdução.....	3
1.2 O <i>framework</i> ViMeT.....	3
2. Processo de instanciação Manual.....	4
2.1. Exemplo de uma instanciação manual	4
3. Processo de Instanciação automático.....	10
3.1 Instalação do Sistema Gerenciados de Banco de Dados Derby	10
3.2 Exemplo de instanciação automática	11
4. Considerações Finais.....	15
Referências	16

1 Introdução

3

Frameworks orientados a objetos permitem o reúso de grandes estruturas em um domínio particular e são personalizados para atender aos requisitos de aplicações específicas desse domínio. Famílias de aplicações similares, mas não idênticas, podem ser derivadas a partir de um único *framework* (Schmid, 1999).

Este documento tem como objetivo auxiliar a instanciação do *ViMeT* (*Virtual Medical Training*), um *framework* orientado a objetos, que utiliza técnicas de RV (Realidade Virtual). Para a completa compreensão deste *framework* é necessário saber sobre:

- o domínio ao qual o *framework* está inserido;
- a estrutura interna do *framework*;
- a utilização do *framework*.

Esta documentação é descrita de forma abrangente para auxiliar usuários que pretendem utilizar o *ViMeT* para o desenvolvimento de novas aplicações, manutenção de aplicações existentes, ou integração de novas funcionalidades e novos dispositivos de entrada e saída. Na Seção 2.1 é apresentado um exemplo de instanciação manual e na Seção 2.2 disponibiliza-se um exemplo de instanciação automática.

1.2 O *framework* *ViMeT*

O *ViMeT* faz parte de um projeto que prevê a construção de aplicações, de código aberto, para treinamento médico, inicialmente de exames de punção. Para que essas aplicações forneçam resultados satisfatórios deverão ser integradas diversas técnicas de deformação, detecção de colisão, estereoscopia e incluídos dispositivos convencionais e não convencionais.

O *ViMeT* possui uma estrutura de classes relativamente simples e teve como objetivo inicial integrar três outros módulos, previamente implementados como aplicações isoladas, que representam uma técnica de deformação (PAVARINI, 2006), uma de detecção de colisão (KERA, 2006) e outra de estereoscopia (BOTEGA, 2006). Além disso, teve como objetivo implementar classes abstratas e subclasses para proporcionar uma estrutura básica projetada para facilitar a integração de novos módulos e dispositivos não convencionais.

Existem duas formas de instanciação do *ViMeT*: uma diretamente do pacote *ViMeT* e outra por meio da *Wizard* (ferramenta de instanciação automática). Na Figura 1 é mostrado o projeto arquitetural do *ViMeT*

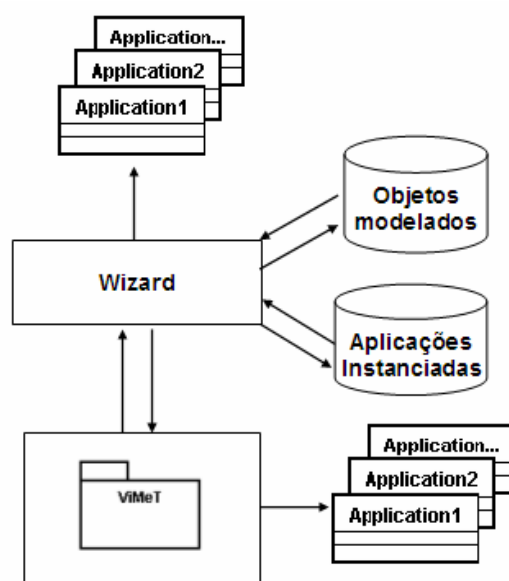


Figura 1 – Projeto arquitetural do *ViMeT*

2 Processo de instanciação Manual

4

Para a instanciação manual do *ViMeT* é necessário, primeiramente, conhecer a hierarquia de classes do *framework* e os principais métodos de cada classe. Na Figura 1 é apresentado o diagrama de classes.

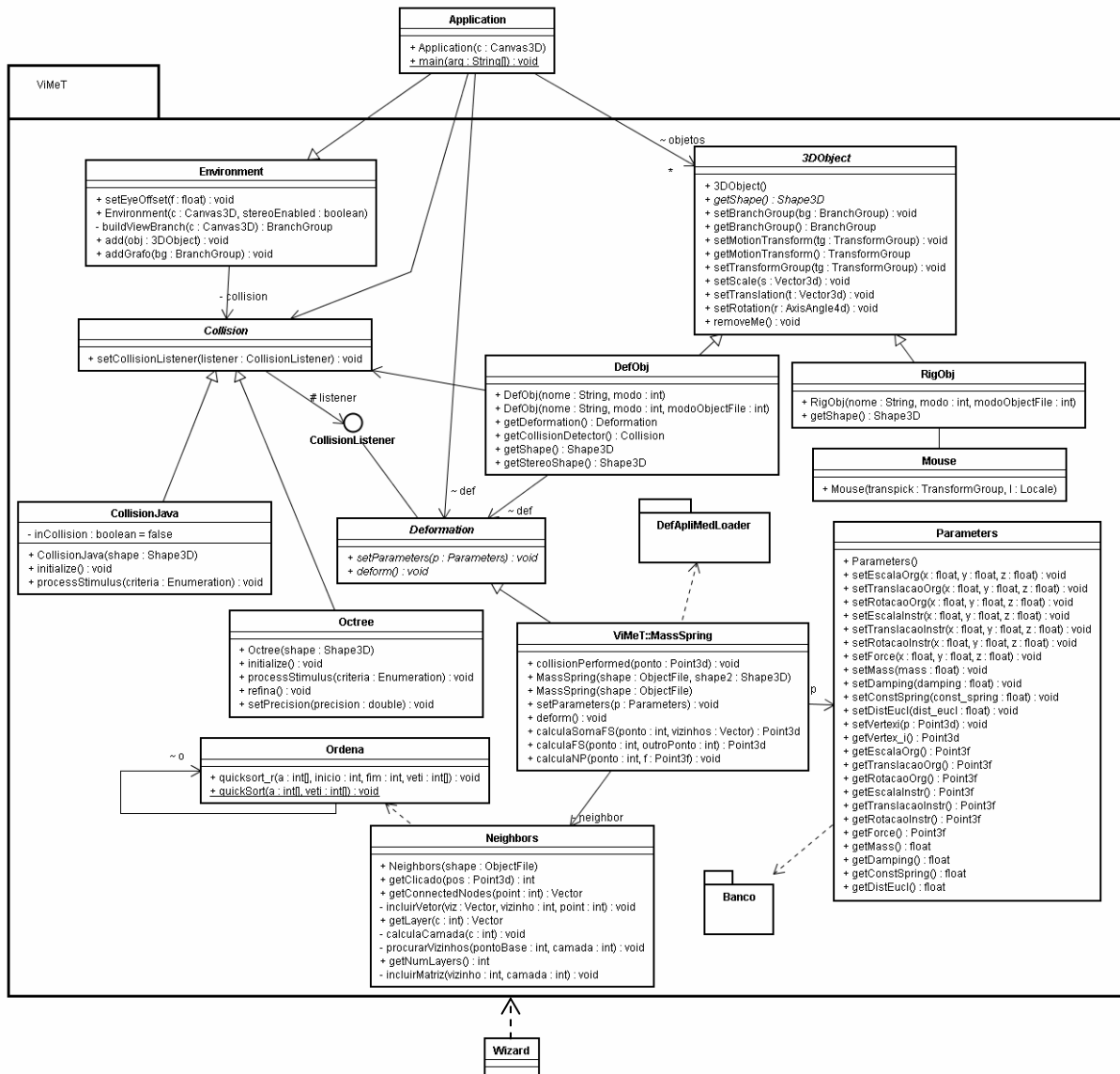


Figura 2 – Diagrama de classes do *ViMeT*

Para auxiliar a compreensão das classes e métodos do *ViMeT* foi feita a documentação no formato *javadoc* (SUN, 2007), disponível em: http://galileu.fundanet.br:80/bcc_bsi/bcc/lapis/projetos/ana/javadoc.php. A seguir são fornecidos exemplos dos dois tipos de instanciação - manual e automática.

2.1 Exemplo de uma instanciação manual

Para facilitar a instanciação do *ViMeT* foi criada uma classe *Application* (Figura 3), que tem como função ser um *template* (gabarito) de instanciação. Nesta classe os *hots spots* (partes variáveis) e os *frozen spots* (partes fixas) permaneceram visíveis. Os *frozen spots* do *ViMeT* são representados pela criação do Ambiente Virtual (AV), *background*, luzes, interação por meio do mouse e interface da aplicação desenvolvida. Esta classe serve como um padrão a ser seguido para o desenvolvimento de aplicações para exames de punção, onde é variável a seleção de técnicas de deformação, detecção de colisão, estereoscopia e seus parâmetros. Porém, se o desenvolvedor desejar, poderá utilizar o *ViMeT* para a construção de aplicações dentro de outro domínio que tenha características semelhantes ao domínio coberto por ele.

Para o desenvolvimento de uma aplicação utilizando diretamente o *ViMeT* é necessário seguir quatro passos:

1. Criação do AV;
2. Carregamento dos objetos e adição das funcionalidades;
3. Alteração dos parâmetros das funcionalidades e objetos modelados;
4. Compilação e execução da aplicação.

A seguir é apresentado um exemplo de instanciação do tipo caixa branca, utilizando o *template*, ou seja, a classe *Application*. A nova aplicação é denominada *Teste* e o código-fonte da classe *Application* é mostrado na Figura 3.

```

0001 import ViMeT.*;
0002 import javax.swing.JFrame;
0003 import java.awt.Container;
0004 import java.awt.event.*;
0005 import javax.media.j3d.*;
0006 import javax.vecmath.*;
0007 import com.sun.j3d.loaders.objectfile.ObjectFile;
0008 public class Application extends Environment{
0009 //atributos
0010 // - Lista de objetos no universo
0011     Object3D objetos[];
0012 // - detecção de colisão
0013     Collision cd;
0014 // - deformação
0015     Deformation def;
0016 public Application(Canvas3D c) {
0017     super(c, true);
0018 //Instanciação dos atributos
0019 // - Instanciação dos objetos
0020     objetos = new Object3D[2];
0021     objetos[0] = new ObjDef("orgao.obj",
0022         Object3D.OCTREE + Object3D.DEFORMATION + Object3D.STEREOCOPY, ObjectFile.RESIZE);
0023     objetos[1] = new ObjRig("instrumento.obj",
0024         Object3D.STEREOCOPY, ObjectFile.RESIZE);
0025 //Adição dos objetos no universo
0026     this.add(objetos[0]);
0027     this.add(objetos[1]);
0028 // - Instanciação da deformação
0029     Parameters p = new Parameters();
0030     def = ((ObjDef)objetos[0]).getDeformation();
0031     p.setForce(0.0f, 0.0f, 0.0f);
0032     p.setMass(0.0f);
0033     p.setDamping(0.0f);
0034     p.setConstSpring(0.0f);
0035     def.setParameters(p);
0036 // - Instanciação da detecção de colisão
0037     cd = ((ObjDef)objetos[0]).getCollisionDetector();
0038     ((Octree)cd).setPrecision(0);
0039     cd.setCollisionListener(def);
0040     BranchGroup bgTemp = new BranchGroup();
0041     bgTemp.addChild(cd);
0042     super.myLocale.addBranchGraph(bgTemp);
0043 // - Estereoscopia
0044     super.setEyeOffset(0f);
0045 // - Dispositivo
0046     Mouse m = new Mouse(objetos[1].getMotionTransform(), super.myLocale);
0047     objetos[0].setScale(new Vector3d (1.0f,1.0f,1.0f));
0048     objetos[0].setTranslation(new Vector3d (0.0f,0.0f,0.0f));
0049     objetos[0].setRotation(new AxisAngle4d (0.0f,0.0f,0.0f,0.0f));
0050     objetos[1].setScale(new Vector3d (1.0f,1.0f,1.0f));
0051     objetos[1].setTranslation(new Vector3d (0.0f,0.0f,0.0f));
0052     objetos[1].setRotation(new AxisAngle4d (0.0f,0.0f,0.0f,0.0f));
0053 }
0054 public static void main (String arg[])
0055 {
0056     Canvas3D c = new Canvas3D(null);
0057     Application n = new Application(c);
0058     JFrame frm = new JFrame("Application");
0059     Container ct = frm.getContentPane();
0060     ct.add(c);
0061     frm.setSize(600,400);
0062     frm.setVisible(true);
0063     frm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
0064 }
0065 }

```

Figura 3 – Código-fonte de classe *Application*

a. Criação do AV

6

Para a criação do AV basta criar uma subclasse da classe *Environment* que herda os atributos e métodos implementados, fazendo com que o AV, *background* e iluminação fiquem disponíveis. As linhas 1 a 7 representam os pacotes que são importados da API Java3D. Na linha 8 é alterado o nome da classe *Application* para *Teste*. Na Figura 4 mostra-se o trecho da nova classe.

```
0001 import ViMeT.*;
0002 import javax.swing.JFrame;
0003 import java.awt.Container;
0004 import java.awt.event.*;
0005 import javax.media.j3d.*;
0006 import javax.vecmath.*;
0007 import com.sun.j3d.loaders.objectfile.ObjectFile;
0008 public class Teste extends Environment{
```

Figura 4 - Trecho da nova Aplicação denominada *Teste*

b. Carregamento dos objetos e adição das funcionalidades

Na Figura 5 são apresentadas três objetos instanciados do *ViMeT*. Para a adição dos objetos modelados no AV é necessário criar um objeto da classe *3DObject*, que é a classe responsável pela importação dos objetos e da associação destes com suas funcionalidades. O vetor que irá conter os objetos modelados é criado na linha 11.

Os objetos das classes *Collision* e *Deformation* são criados nas linhas 13 e 15, respectivamente. A classe abstrata *Collision* contém os atributos e métodos similares da técnica de deformação *Octrees* e dos métodos nativos da API Java3D. A classe abstrata *Deformation* é responsável por gerenciar os atributos e métodos similares das várias técnicas de deformação. No estágio atual do *ViMeT* só existe uma técnica implementada (Massa-Mola). As linhas 9 a 15 não se modificam na classe *Teste*.

```
0009 //atributos
0010 // - Lista de objetos no universo
0011     Object3D objetos[];
0012 // - detecção de colisão
0013     Collision cd;
0014 // - deformação
0015     Deformation def;
```

Figura 5 - Trecho mantido da classe *Application*

Na Figura 7 é apresentado um trecho de código da classe *Teste*. Na linha 16 é criado o método construtor que possui como parâmetro o objeto *Canvas3D*. O método *super* faz referência aos parâmetros da superclasse *Environment*. Com isso, a nova aplicação irá herdar o AV, a iluminação, o *background* e, ainda, a técnica de estereoscopia Anaglifos. Esta técnica foi implementada na classe *Environment* e caso o valor do parâmetro *stereoEnabled* for *false*, a aplicação não terá a funcionalidade estereoscopia e quando for *true* esta funcionalidade ficará disponível na nova aplicação.

A instância *objetos* da classe *3DObject* recebe o vetor com o número de objetos carregados no AV, na linha 20. Na linha 21 a instância da subclasse *ObjDef* (classe que contém os atributos e métodos relacionados aos objetos modelados que simulam a deformação), representada por *objetos[0]* recebe os parâmetros do objeto que representa o órgão humano: o arquivo *.obj*, os atributos *OCTREE*, *DEFORMATION* e *STEREOSCOPY* (todos provenientes da classe *3DObject*) e, ainda, o método *Resize* da classe *ObjectFile*. Este último é mais um *frozen spot* do *ViMeT*. Este método foi implementado, mas na fase atual do *ViMeT* não ficou flexível para o usuário alterá-lo.

Na linha 23 é feita a instanciação da subclasse *RigObj* (classe que contém os atributos e métodos relacionados aos objetos que simulam um instrumento médico), representada por *objetos [1]*. Nas linhas 26 e 27, os *objetos[0]* e *objetos[1]* são adicionados no AV.

Na Figura 7, as linhas 21 e 23 recebem diretamente o nome do arquivo com extensão *.obj* e também todos os atributos referentes às funcionalidades que o *ViMeT* possui. Os atributos referentes a detecção de colisão e deformação sempre devem estar definidos, o da estereoscopia pode ser variável.

```

0016 public Teste(Canvas3D c) {
0017     super(c, true);
0018     //Instanciação dos atributos
0019     // - Instanciação dos objetos
0020     objetos = new Object3D[2];
0021     objetos[0] = new ObjDef("E:\\ObjetosModelados\\mama_ac.obj",
0022 Object3D.OCTREE + Object3D.DEFORMATION + Object3D.STEREOCOPY, ObjectFile.RESIZE);
0023     objetos[1] = new ObjRig("E:\\ObjetosModelados\\seringa pronta.obj",
0024 Object3D.STEREOCOPY, ObjectFile.RESIZE);
0025     //Adição dos objetos no universo
0026     this.add(objetos[0]);
0027     this.add(objetos[1]);

```

Figura 8 - Trecho classe *Teste* para definição carregamento e adição das funcionalidades

Na linha 29 foi criada uma instância (*p*) da classe *Parameters*. O objeto *def*, que tem sua instância criada na linha 15 e na linha 30, recebe o método *getDeformation* (método que representa a técnica de deformação que será empregada) da classe *DefObj*. Nas linhas 31 a 34 o objeto *p* recebe os valores dos parâmetros força, massa, *damping* e constante da mola. Na linha 35 o objeto *def* recebe como parâmetro o *p*. Na Figura 9 verifica-se que os parâmetros possuem os valores definidos para cada um deles.

```

0028 // - Instanciação da deformação
0029     Parameters p = new Parameters();
0030     def = ((ObjDef)objetos[0]).getDeformation();
0031     p.setForce(3.0f, 0.0f, 0.0f);
0032     p.setMass(300.0f);
0033     p.setDamping(0.7f);
0034     p.setConstSpring(0.3f);
0035     def.setParameters(p);

```

Figura 9 - Trecho classe *Teste* para a instanciação da deformação

Na Figura 10 é apresentada a instanciação da detecção de colisão. Na linha 37 o método *getCollisionDetector* da classe *DefObj* é passado como parâmetro para o objeto *cd* que tem sua instância da classe *Octree*, criada na linha 13. Na linha 38 o objeto *cd* recebe como parâmetro o método *setPrecision* e seu valor default é 0.0010 e na linha 39 é definido o método *setCollisionListener* com o parâmetro *def*. Na linha 40 é criada um objeto *bgTemp* que é uma instância da classe *BranchGroup* da API Java3D e na linha 41 o *bgTemp* tem adicionado como filho o parâmetro *cd*. Na linha 42 o *bgTemp* é adicionado ao *myLocale*. Na Figura 11 é apresentado um trecho de código quando a técnica escolhida para a detecção de colisão é a CJAVA, definida na Figura 9, linha 22.

```

0036 // - Instanciação da detecção de colisão
0037     cd = ((ObjDef)objetos[0]).getCollisionDetector();
0038     ((Octree)cd).setPrecision(0.0010);
0039     cd.setCollisionListener(def);
0040     BranchGroup bgTemp = new BranchGroup();
0041     bgTemp.addChild(cd);
0042     super.myLocale.addBranchGraph(bgTemp);

```

Figura 10 - Trecho da classe *Teste* para instanciação a colisão (*Octrees*)

```

0036 // - Instanciação da detecção de colisão
0037     cd = ((ObjDef)objetos[0]).getCollisionDetector();
0038     cd.setCollisionListener(def);
0039     BranchGroup bgTemp = new BranchGroup();
0040     bgTemp.addChild(cd);
0041     super.myLocale.addBranchGraph(bgTemp);

```

Figura 11- Trecho alterado da classe *Teste* para instanciação a colisão (*CJava*)

Na Figura 12 pode se observado o último trecho da classe *Teste* onde na linha 44 o método *super* faz referência ao método *setEyeOffset* da superclasse *Enviroment*, para definir o valor da paralaxe (distância interocular).

```
0043 // - Estereoscopia
0044     super.setEyeOffset(0.017f);
```

Figura 12 - Trecho mantido da classe *Application*

Na Figura 13 a instanciação da classe *Mouse*, que é responsável por todo comportamento adicionado no mouse, é feita na linha 46. Nas linhas 47 até 52 as instâncias *objetos[0]* e *objetos[1]* recebem como parâmetros os métodos que armazenam os valores das transformações (escala, translação e rotação).

```
0045 // - Dispositivo
0046     Mouse m = new Mouse(objetos[1].getMotionTransform(), super.myLocale);
0047     objetos[0].setScale(new Vector3d (0.9f,0.9f,0.9f));
0048     objetos[0].setTranslation(new Vector3d (0.0f,0.0f,0.0f));
0049     objetos[0].setRotation(new AxisAngle4d (0.0f,0.0f,0.0f,0.0f));
0050     objetos[1].setScale(new Vector3d (0.5f,0.5f,0.5f));
0051     objetos[1].setTranslation(new Vector3d (0.5f,0.5f,0.2f));
0052     objetos[1].setRotation(new AxisAngle4d (0.0f,0.0f,1.0f,0.5f));
0053 }
```

Figura 13 - Trecho da classe *Teste* para a definição da utilização do mouse

Finalizando, as linhas 54 até 65 fazem parte do método *main* que tem como finalidade abstrair os outros trechos da classe. A aplicação gerada será mostrada ao desenvolvedor por meio de uma interface que é criada na linha 58 e adicionado em um *Container* na linha 75 e neste é adicionado o objeto (*c*). A Figura 14 mostra um trecho da classe *Application* e a Figura 15 da classe *Teste*.

```
0054     public static void main (String arg[])
0055     {
0056         Canvas3D c = new Canvas3D(null);
0057         Teste n = new Teste(c);
0058         JFrame frm = new JFrame("Teste");
0059         Container ct = frm.getContentPane();
0060         ct.add(c);
0061         frm.setSize(600,400);
0062         frm.setVisible(true);
0063         frm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
0064     }
0065 }
```

Figura 14 - Trecho classe *Teste* método *main*

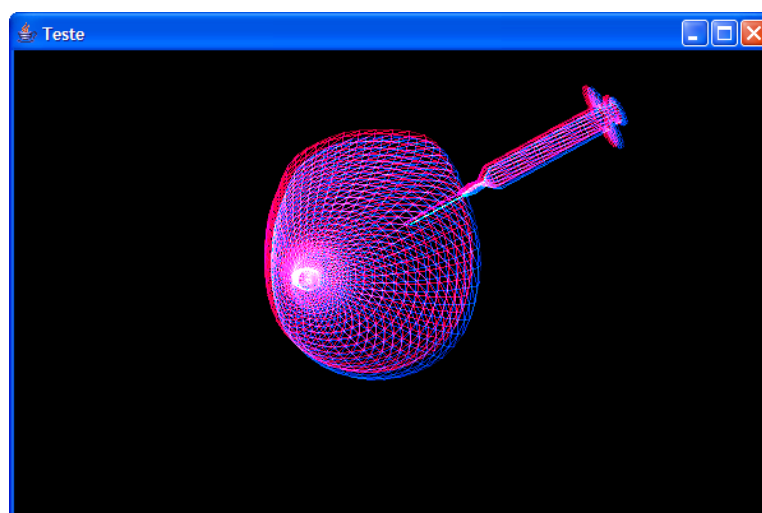


Figura 15 – Resultado da aplicação *Teste*

3 Processo de Instanciação automático

Para a utilização da ferramenta de instanciação automática *Wizard* do *ViMeT* é necessária a instalação do SGBD Derby (APACHE, 2006), descrita a seguir.

3.1 Instalação do Sistema Gerenciados de Banco de Dados Derby

O primeiro passo para a instalação do Derby é fazer o *download* disponível na página da Apache, empresa responsável pelo desenvolvimento do Derby (APACHE, 2007). Nesta página existe uma lista de arquivos do tipo *zip* e *tar*. Existem distribuições do tipo *bin*, *lib* e *src*, sendo que é a *bin* a recomendada para *download*. Esta distribuição possui cinco subdiretórios a saber:

- *demo*: contém programas de demonstração;
- *frameworks*: contém scripts para a execução de utilitários e configuração do ambiente;
- *javadoc*: contém a documentação do Derby;
- *doc*: contém a documentação do Derby.
- *lib*: contém os arquivos com a extensão *.jar* do Derby.

Depois de feito o *download*, deve-se descompactar o arquivo na pasta onde está instalada a linguagem Java e, em seguida, configurar as variáveis de ambiente do Derby. A configuração é feita da seguinte maneira:

1. Criar a variável `DERBY_HOME` de acordo com a Figura 16.
2. Definir a variável `CLASSPATH` de acordo com a Figura 17 (a) e (b)
3. Definir a variável *Path* desta maneira: `%DERBY_HOME%\frameworks\embedded\bin`.

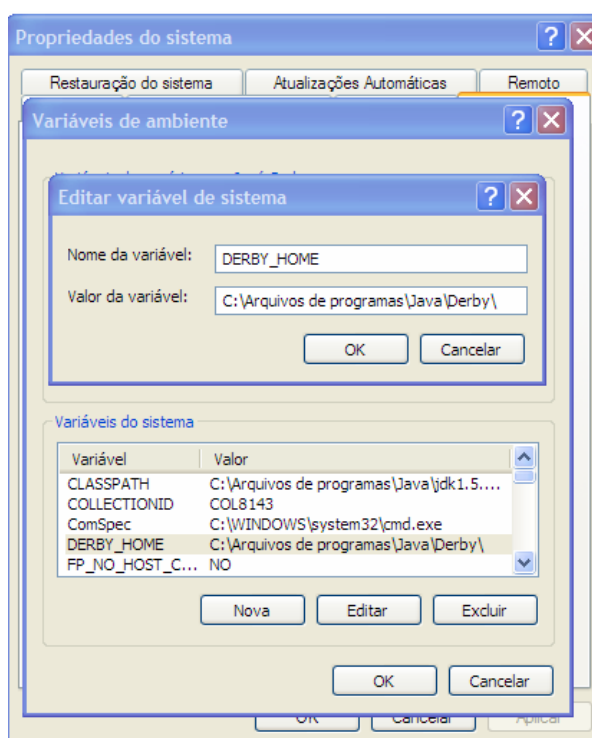
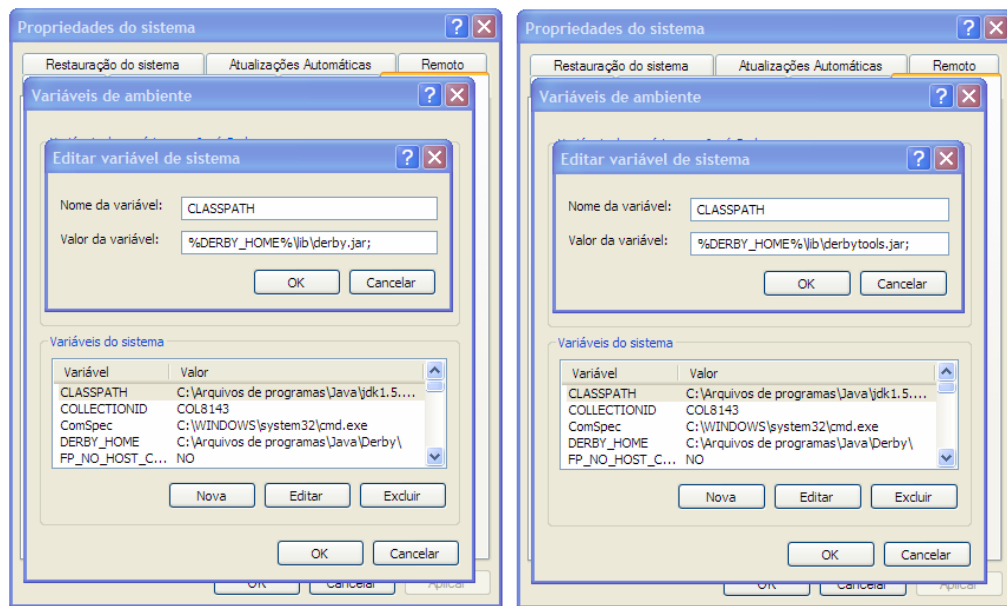


Figura 16 - Configuração da variável `DERBY_HOME`



(a) (b)
Figura 17 - Configuração da variável CLASSPATH

Com esta configuração a ferramenta *ij*, responsável pela conexão com o BD Derby fica habilitada. Para executar manualmente o aplicativo *ij*, deve-se digitar: `java org.apache.derby.tools.ij`, em uma janela *MS-DOS*, conforme Figura 18. Em seguida, podem ser executadas quaisquer instruções SQL.

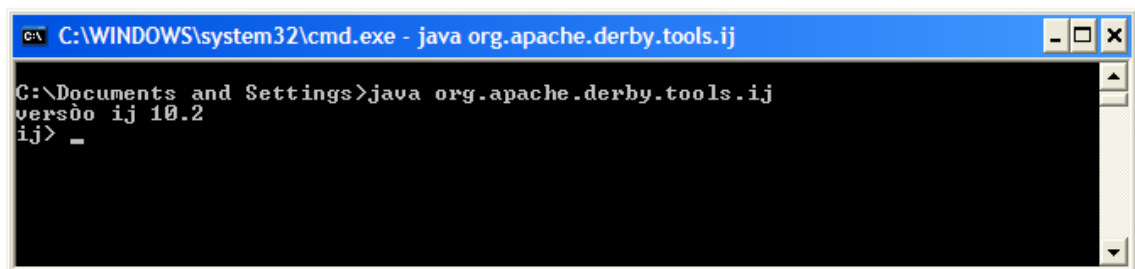


Figura 18 - Execução do aplicativo *ij*

3.2 Exemplo de instanciação automática

Nesta seção é mostrado um exemplo de instanciação automática realizado por meio da *Wizard*. Na Figura 19 é mostrada a interface da *Wizard*. Após a instalação do BD basta executar o arquivo *Wizard.java* e em seguida a interface fica disponibilizada.

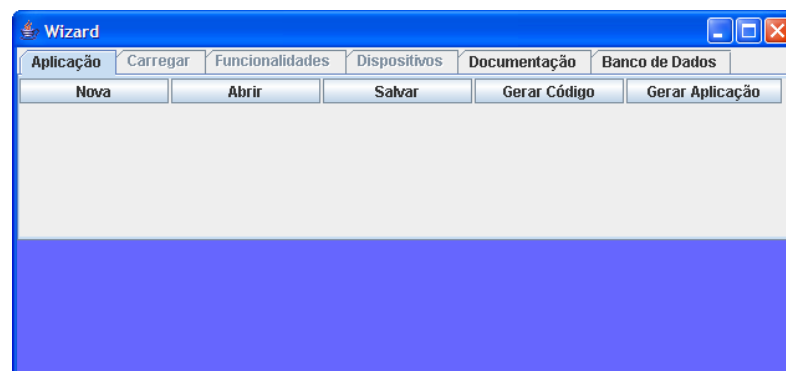


Figura 19 - Interface da *Wizard*

A *Wizard* também garante o acesso ao BD, sendo possível salvar novos objetos modelados, removê-los e alterá-los, assim como é possível salvar, alterar e excluir os parâmetros das aplicações armazenadas. Para utilizar a *Wizard*, sugere-se uma seqüência de atividades:

1. Consultar a guia *Documentação*, onde se encontra toda a documentação do *ViMeT*, que consiste neste documento, no javadoc e no diagrama de classes;
2. Incluir os objetos modelados no BD por meio da guia *Banco de Dados*;
3. Criar a nova aplicação com a utilização da guia *Aplicações*;
4. Carregar objetos modelados no AV e definir os parâmetros das transformações na guia *Carregar*;
5. Escolher das funcionalidades e seus parâmetros na guia *Funcionalidades*;
6. Na guia *Aplicações*, *Salvar* todos os valores definidos para a aplicação Com o botão *Gerar Código*, o código-fonte da aplicação é aberto em uma janela. E, por último, com o botão *Gerar Aplicação* é aberta uma janela com a nova aplicação.

Para facilitar a utilização da *Wizard*, as seis atividades descritas são detalhadas a seguir com ilustrações. Na Figura 20 pode ser observada a gravação de um objeto que simula um órgão. Verifica-se que é necessário definir o tipo de objeto que será gravado no BD. Isto acontece porque existe apenas uma tabela para ambos os objetos, sendo que cada um deles possui características diferentes, quanto ao método de carregamento, aparências e funcionalidades. Este código determinado no momento da gravação no BD, é utilizado na criação e carregamento das aplicações.

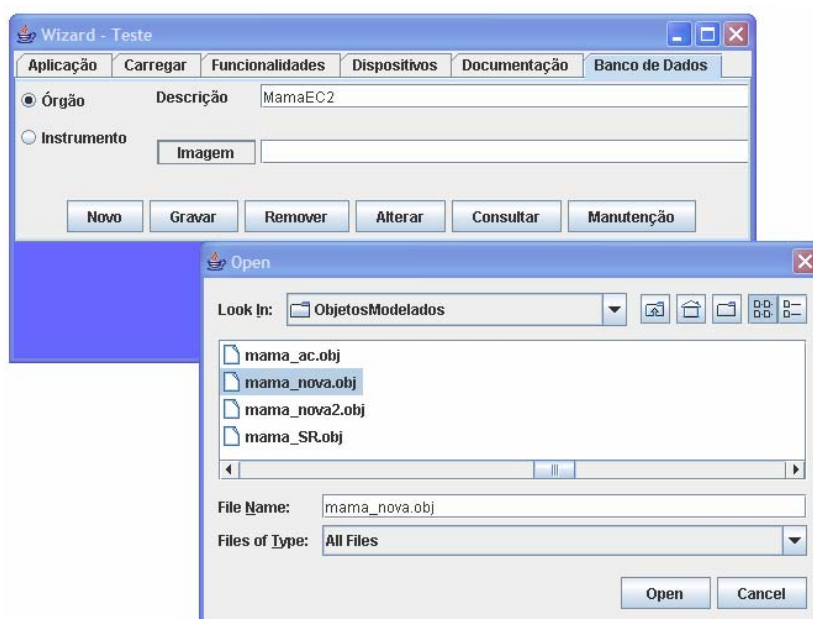


Figura 20- Gravação no Banco de Dados

Após a gravação dos objetos modelados que podem ser utilizados nas futuras aplicações pode se dar início ao desenvolvimento de uma nova aplicação. A criação das novas aplicações é realizada por meio da guia *Aplicações*. Será criada uma aplicação como exemplo para mostrar passo-a-passo esta criação. Neste exemplo foi dado o nome de *Teste*, conforme Figura 21.

Quando o botão *Nova* é acionado, dá-se início à instanciação do *ViMeT*. A aplicação criada irá conter todas características da classe *Environment*, ou seja, o AV, o *background* e as luzes. Estas três características representam os *frozen spots* do *ViMeT*.

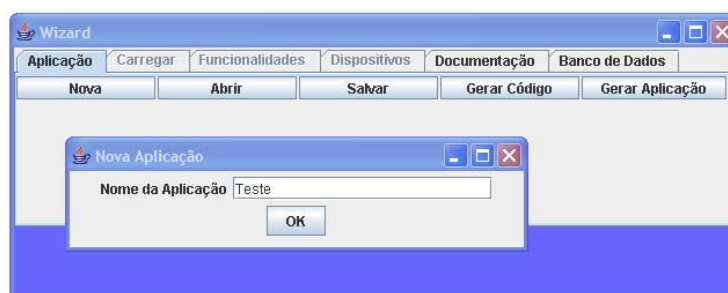


Figura 21 - Criação de uma nova aplicação

Após a criação da nova aplicação as guias *Carregar*, *Funcionalidades* e *Dispositivos* ficam habilitadas. A próxima guia a ser utilizada é a guia *Carregar*. Nela estão as opções para carregar os objetos modelados no AV e também os campos para a definição das transformações dos objetos.

Na Figura 22 podem ser observados um objeto modelado que simula uma mama e outro objeto que simula uma seringa, além dos valores empregados na escala, translação e rotação da seringa. Todos estes parâmetros são armazenados no BD, podendo ser consultados e alterados futuramente.

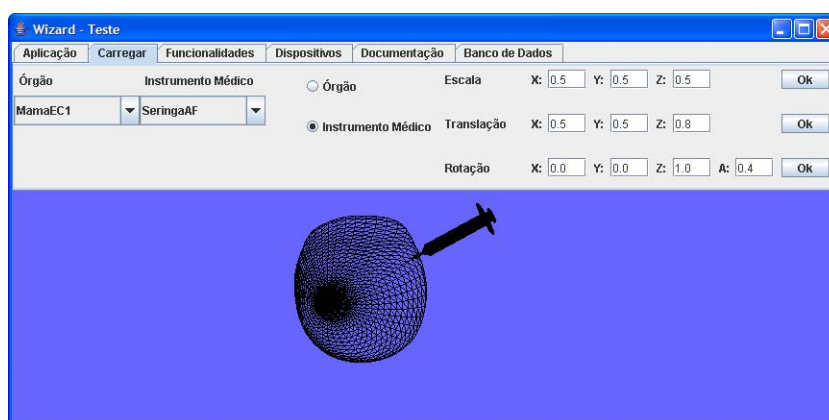


Figura 22 - Carregando os objetos no AV

Na guia *Funcionalidades* estão as opções das funcionalidades e de seus parâmetros. Estes valores também são armazenados no BD, conforme Figura 23.

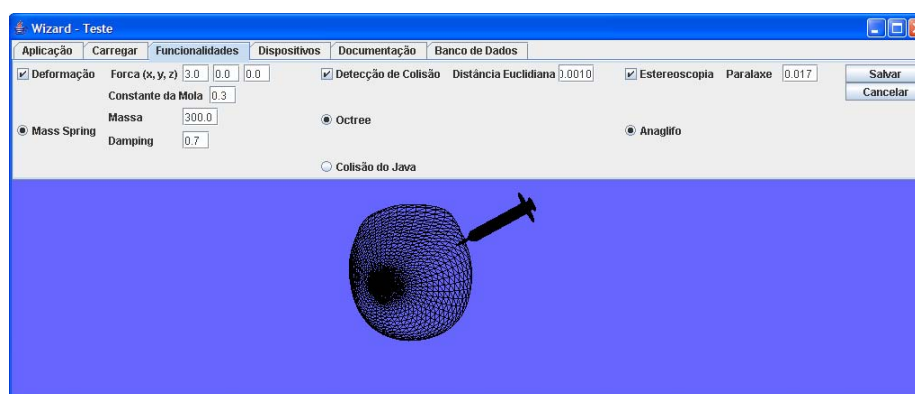


Figura 23- Escolha das funcionalidades e seus parâmetros

Voltando à guia Aplicações, é possível gravar os parâmetros definidos acionando-se o botão *Salvar*. Em seguida, pode-se gerar o código (botão *Gerar Código*) e, então gerar a aplicação acionando-se o botão com esta funcionalidade. A Figura 24 mostra o resultado destas três ações.

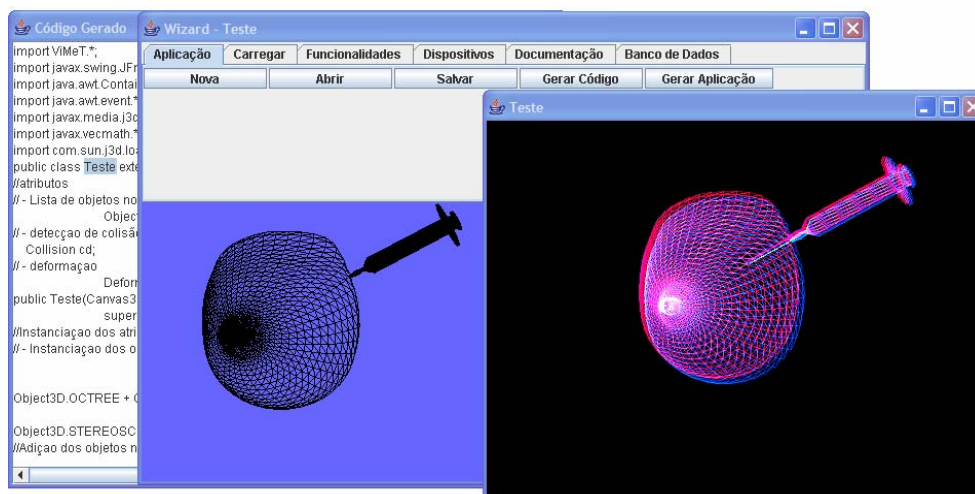


Figura 24 - Código-fonte e aplicação gerada

A guia *Aplicações* também possibilita abrir uma aplicação já desenvolvida e gravada no BD. Na Figura 25 pode ser observado como isso acontece. Caso seja necessário, podem ser alterados e uma nova aplicação pode ser criada.

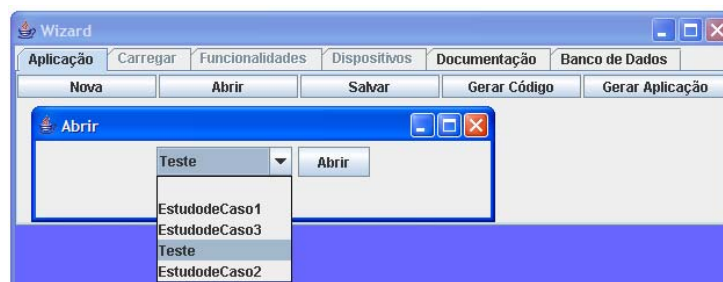


Figura 25 - Abrir uma aplicação salva no BD.

Uma outra forma de alterar a aplicação e por meio da guia Banco de Dados, nela existe o botão *Manutenção*, que ao ser clicado abre uma janela, conforme Figura 26. Nesta janela é possível alterar os parâmetros da aplicação ou até mesmo removê-la. Somente o nome da aplicação não é possível alterar, caso seja necessário recomenda-se que uma nova aplicação seja criada.

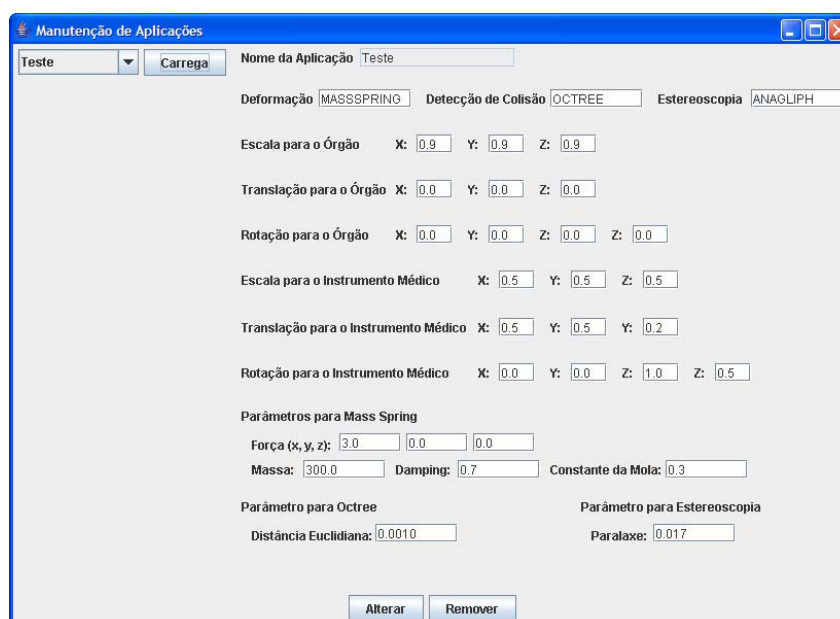


Figura 26 - Abrir uma aplicação existente

4 Considerações Finais

14

Este documento mostrou, detalhadamente, como desenvolver uma aplicação utilizando o *ViMeT*. Existem dois tipos de instanciação possíveis: manual e automática.

A instanciação feita manualmente deve ser realizada por um desenvolvedor com conhecimentos da linguagem Java.

A instanciação realizada por meio da ferramenta *Wizard* automatiza o processo de instanciação, geração de código e armazenamento no Banco de Dados. Pretende-se que esse manual sirva como base para a utilização do *ViMeT*.

Referências

15

APACHE. Distribuições. Disponível em: < <http://db.apache.org/derby/releases/release-10.2.1.6.cgi#Distributions>> Acesso em: 05 fev. 2007.

BOTEGA, L. C.. **Implementação de Estereoscopia de Baixo Custo para Aplicações em Ferramentas de Realidade Virtual para Treinamento Médico**. 2005. 105 f. Grau: Monografia (Bacharelado em Ciência da Computação) Centro Universitário Eurípides de Marília – Fundação de Ensino Eurípides Soares da Rocha, Marília, 2005.

BRAGA, R. T. V.; MASIERO, P. C. *GREN-Wizard*, Disponível em: <<http://www.icmc.usp.br/~rtvb/GRENWizard.htm>> Acesso em: 12 fev. 2007.

KERA, M. **Detecção de colisão utilizando hierarquias em ferramentas de realidade virtual para treinamento médico**. 2005 92 f. Grau: Monografia (Bacharelado em Computação) Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2005. Disponível em: < <http://lost.fundanet.br/~kera>>. Acesso em 20 dez. 2005

PAVARINI, L. **Estudo e Implementação do Método massa-mola para Deformação em Ambientes Virtuais de Treinamento Médico usando a API Java 3D**. 2006. 147f. Dissertação (Mestrado em Ciência da Computação) - Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2006.

SCHMID, H. A. *Framework design by systematic generalization*. In: FAYAD, M. JOHNSON, R., SCHMIDT D. **Building Application Frameworks: Object-Oriented Foundation of Frameworks Design**. John Wiley & Sons, Nova Iorque, p. 353-378, 1999.

SUN.(2006). **Java 3D API Tutorial**. Disponível em:
< <http://java.sun.com/developer/onlineTraining/java3d/>> . Acesso em: dez. 2006.

APÊNDICE B – Javadoc

Neste apêndice é apresentado o sumário com todas as classes do pacote *ViMeT* (Figura 1). Esta documentação pode ser obtida, na íntegra no site do LApIS: (http://galileu.fundanet.br/bcc_bsi/bcc/lapis/projetos/ana/javadoc.php).

Package *ViMeT*

Interface Summary	
CollisionListener	Interface que contém o método collisionPerfomerd

Class Summary	
Collision	Classe abstrata que contém a interface CollisionListener.
CollisionJava	Subclasse que possui os métodos para a detecção de colisão default da API Java 3D
Deformation	Classe Abstrata que contém os métodos similares entre as técnicas de deformação.
Environment	Classe responsável pela criação do Ambiente Virtual.
Layer	Possui o método para calcular o número de camadas que deformam em relação à Força aplicada.
MassSpring	Subclasse que possui os métodos para reposicionamento de vértices e demais estruturas de objetos 3D na cena.
Mouse	Classe pela movimentação do mouse na cena.
Neighbors	Constrói a estrutura de camadas e armazená-la e encontra os vértices vizinhos conectados de um determinado ponto selecionado do objeto.
ObjDef	Possui os métodos de carregamento, funcionalidades do objeto modelado que simula um órgão (objeto deformável).
Object3D	Possui os atributos referentes as funcionalidades existentes, o método para adicionar os objetos no MyLocale e os métodos responsáveis pelas transformações (escala, translação e rotação) nos objetos modelados que fazem parte do AV.
ObjRig	Possui os métodos de carregamento, funcionalidades do objeto modelado que simula um instrumento médico (objeto rígido)
Octrees	Subclasse que contém a técnica <i>Octrees</i> implementada.
Ordena	Organiza os vértices vizinhos ao vértice clicado.
Parameters	Possui métodos responsáveis por armazenar e recuperar valores nos parâmetros no BD.

Figura 1 - Sumário do pacote *ViMeT*