FUNDAÇÃO DE ENSINO “EURÍPIDES SOARES DA ROCHA”
CENTRO UNIVERSITÁRIO “EURÍPIDES DE MARÍLIA” – UNIVEM
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

DANILO JUSTO ROSSATTO

BANCO DE DADOS E WIZARD PARA FRAMEWORK DE REALIDADE VIRTUAL PARA TREINAMENTO MÉDICO

MARÍLIA
2006

DANILO JUSTO ROSSATTO

BANCO DE DADOS E WIZARD PARA FRAMEWORK DE REALIDADE VIRTUAL PARA TREINAMENTO MÉDICO

Monografia apresentada ao curso de Bacharelado em Ciência da Computação do Centro Universitário Eurípides de Marília (UNIVEM), mantido pela Fundação de Ensino Eurípides Soares da Rocha, como requisito parcial para obtenção do Título de Bacharel em Ciência da Computação.

Orientador (a):
Profª. Drª. Fátima L. S. Nunes Marques

MARÍLIA
2006

ROSSATTO, Danilo Justo
Banco de Dados e Wizard para Framework de Realidade Virtual para Treinamento Médico / Danilo Justo Rossatto; orientadora: Fátima L. S. Nunes. Marília, SP: [s.n.], 2006.
105 f.

Monografia (Bacharelado em Ciência da Computação) – Centro Universitário Eurípides de Marília – UNIVEM – Fundação de Ensino Eurípides Soares da Rocha

1.Banco de Dados 2. RealidadeVirtual 3.*Framework*

CDD: 005.74

À Deus pela iluminação e proteção em toda minha vida, e de meus queridos amados;

À minha mãe e meu pai que me educaram e me deram a oportunidade de estar concluindo este trabalho para minha formação acadêmica;

Às minhas irmãs Fernanda e Franceli, que apesar da distância, pelo todo carinho;

Aos amigos da faculdade, em especial ao Adriano, Ana Paula, Bárbara, Mariana e Rodrigo, pela força dada durante todo nosso período de graduação.

AGRADECIMENTOS

Agradeço à minha professora e orientadora Profª. Drª. Fátima de Lourdes dos Santos Nunes Marques, pela orientação, incentivo e apoio para que o projeto fosse concluído com relevância.

A todos os professores que ajudaram com idéias, estímulo e companheirismo ao longo desta jornada.

Não poderia deixar de lado, aos amigos Fernando Ferreira, Kelson Ferreira, Guilherme Scombatti, José Roberto (Zé), Larissa Andreolli, Ana Cláudia Oliveira, Larissa Pavarini e Elaine pelo incentivo, estímulo e força para que terminasse o projeto.

A todos os colegas da Fundação de Ensino “Eurípides Soares da Rocha” – UNIVEM – os quais convivi durante os últimos 4 anos.

"Experiência é o nome que nós damos aos nossos próprios erros"

Oscar Wilde

ROSSATTO, Danilo Justo. **Banco de Dados e Wizard para Framework de Realidade Virtual para Treinamento Médico.** 2006 105f. Monografia (Bacharelado em Ciência da Computação) – Centro Universitário Eurípides de Marília – Fundação de Ensino Eurípides Soares da Rocha, Marilia, 2006.

RESUMO

Visto que um Banco de Dados (BD) é essencial para realizar testes de um *Framework* orientado a objetos utilizando-se técnicas de Realidade Virtual para procedimentos minimamente invasivos, com foco em punção de mama, foi desenvolvido um BD com um Sistema Gerenciador de Banco de Dados (SGBD) gratuito, escrito inteiramente em Java, o Derby, juntamente com uma ferramenta *Wizard*, que realiza a instanciação automática do *Framework* a partir de parâmetros fornecidos pelo usuário. O principal objetivo do BD é armazenar dados a respeito de objetos tridimensionais que representam órgãos humanos e instrumentos médicos. Para isto, as aplicações foram desenvolvidas na linguagem de programação Java, podendo assim obter a reusabilidade de código e prover uma ferramenta rápida e prática para os estudos e diagnósticos da área médica. As aplicações são geradas pela ferramenta *Wizard*, e armazenadas no BD, com o propósito de posteriomente serem consultadas e alteradas de acordo com os objetivos desejados.

Palavras-chave: Banco de Dados, Realidade Virtual, *Framework, Wizard*.

ROSSATTO, Danilo Justo. **Banco de Dados e Wizard para Framework de Realidade Virtual para Treinamento Médico.** 2006 105f. Monografia (Bacharelado em Ciência da Computação) – Centro Universitário Eurípides de Marília – Fundação de Ensino Eurípides Soares da Rocha, Marilia, 2006.

ABSTRACT

To accomplish tests of a Framework oriented to objects using techniques of Virtual Reality (VR) for medical procedures, with focus in breast biopsy, a Database (DB) was developed with a free Database Management System (DBMS), written entirely in Java, the Derby, together with a tool called Wizard, that realize a automatic instance of Framework starting from parameters supplied by the user. The main objective of DB is to store data regarding three-dimensional objects that represent human organs and medical instruments. For this, the applications were developed in the programming language Java, reusing the code and to provide a fast and practical tool for the studies and diagnoses of the medical area. The applications are generated by a Wizard, a automatic tool of instance, and stored in the DB, with the purpose to be consulted and altered in agreement with the wanted objectives.

Keyworlds: Database, Virtual Reality, Framework, Wizard.

ROSSATTO, Danilo Justo. **Banco de Dados e Wizard para Framework de Realidade Virtual para Treinamento Médico.** 2006 105f. Monografia (Bacharelado em Ciência da Computação) – Centro Universitário Eurípides de Marília – Fundação de Ensino Eurípides Soares da Rocha, Marilia, 2006.

RESUMEN

Ya que una Base de Datos (BD) es esencial para lograr las pruebas de un Framework orientado a objetos que utilizan técnicas de Realidad Virtual (RV) para los procedimientos médicos, con el enfoque en la punción de mamma, una BD fue desarrollado con un Sistema Gerenciador de Base de Datos (SGBD) libre, escrito completamente en Java, el Derby, juntamente con herramienta Wizard, que realiza la instanciación automática del Framework a partir de parámetros proporcionados por el usuario. El objetivo principal del BD es armazenar datos de objetos tridimensionales que representan órganos humanos y los instrumentos médicos. Para esto, las aplicaciones se desarrollaron en el idioma de programación Java, podendo reutilizaren el código y prover una herramienta lista e práctica para los estudios y diagnósticos del área médica. Las aplicaciones se generan por un Wizard, una herramienta automática de instanciación, y armazenadas en la BD, con el propósito de ser consultado y alterado de acuerdo con sus objetivos.

Palabras-Clave: Banco de Dados, Realidad Virtual, Framework, Wizard.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API: (*Application Programming Interface*) Interface de Programação de Aplicação

AV: Ambiente Virtual

AVET: Ambiente de Videoconferência para o Ensino Tecnológico

AWT: (*Abstract Window Toolkit)* Ferramenta de Janela Abstrata

BD: Banco de Dados

BDR: Banco de Dados Relacional

CAD: (*Computer-Aided Diagnosis*) Diagnóstico Auxiliado por Computador

ETD: Ensino Tecnológico a Distância

HSQLDB: (*Hypersonic SQL Database*) Banco de Dados SQL Hipersonico

HTTP: (*HyperText Transfer Protocol*) Protocolo de Transferência HiperTexto

JDBC: (*Java Database Connectivity*) Conectividade Banco de Dados Java

JVM: (Java Virtual Machine) Máquina Virtual Java

RDBMS: (*Relational Database Management System*) Sistema de Gerenciamento de Banco de Dados Relacional

RV: Realidade Virtual

SAM: Servidor de Armazenamento de Mídias

SGBD: Sistema Gerenciador de Banco de Dados

SIG-3D: Sistemas de Informações Geográficas Tridimensionais

SQL: (*Structured Query Language*) Linguagem de Consulta Estruturada

LApIS: Laboratório de Aplicações de Informática em Saúde

MVC: (*Model-View-Controller*) Controlador de Modelo de Visão

UML: (*Unified Modeling Language*) Linguagem de Modelagem Unificada

SUMÁRIO

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.

As pesquisas de RV focalizando aplicações para treinamento médico crescem a cada dia, apesar de ainda serem observados poucos trabalhos nacionais sendo aplicados na prática médica. Simuladores de procedimentos para Medicina utilizam a RV para oferecer sistemas de tempo-real interativos e com estímulo visual, em sua maioria (BOTEGA, 2005).

O avanço na área médica permitiu uma nova modalidade de cirurgia, o procedimento minimamente invasivo, que permite a inserção e manipulação de instrumentos médicos no corpo do paciente através de pequenas incisões (SORID, 2000).

As aplicações de RV são de grande importância para a Medicina, proporcionando a um estudante/cirurgião realizar diversos treinamentos em um ambiente virtual (AV) ou um paciente virtual, tendo de volta o *feedback* da sua atuação. Assim, o nível de aprendizado abrange vários detalhes que não poderiam ser perceptíveis em uma prática convencional.

Objetivo do trabalho

Este trabalho tem como objetivo a construção de um BD e uma interface de manutenção do mesmo para teste do ViMet, um *framework* orientado a objetos para aplicações de treinamento médico utilizando técnicas de RV, tendo como foco os exames de punção a partir de um *Wizard*. O *framework* está em fase de desenvolvimento e tem as seguintes funcionalidades já implementadas: detecção de colisão com precisão, deformação e estereoscopia.

Justificativa do trabalho

Para que uma aplicação de Realidade Virtual para treinamento médico seja efetiva, é necessário que uma Base de Dados armazene os casos disponíveis, contendo os objetos e imagens 3Ds, assim como as aplicações, para que seja possível estabelecer uma forma de avaliar seu desempenho.

O BD do *framework* de RV estará interligado a uma interface de manutenção para testes e validações, possibilitando um treinamento médico mais efetivo.

Disposição do trabalho

Além desta introdução, a monografia apresenta mais cinco capítulos, a saber:

O Capítulo 1 primeiramente conceitua o termo Realidade Virtual, apresentando algumas aplicações de RV para treinamento médico e, em seguida, apresenta o conceito de *framework* destacando alguns que foram implementados na área da RV.

O Capítulo 2 apresenta o conceito de Banco de Dados abordando aplicações que utilizam SGBDs na área da RV.

O Capítulo 3 aborda as características de alguns SGBDs gratuitos desenvolvidos em Java, bem como os passos de instalação e configuração do SGBD que será utilizado, o Derby.

O Capítulo 4 apresenta a implementação do projeto Banco de Dados, o ambiente de programação utilizado e o funcionamento do sistema.

O Capítulo 5 abrange dois estudos de casos realizados para o teste do *framework* ViMet e do Banco de Dados.

E, por último, é apresentada a conclusão do trabalho e alguns trabalhos futuros que podem estar complementando esta monografia, e também as referências utilizadas para a fundamentação teórica.

CAPÍTULO 1 - APLICAÇÕES DE REALIDADE VIRTUAL NA MEDICINA

1.1 – Realidade Virtual

O termo Realidade Virtual (RV) é creditado a Jaron Lanier, fundador da VPL Research Inc., que o cunhou, no início dos anos 80, para diferenciar as simulações tradicionais feitas por computador de simulações envolvendo múltiplos usuários em um ambiente compartilhado (ARAÚJO, 1996).

O tema RV é bastante abrangente. Acadêmicos, desenvolvedores de softwares e pesquisadores tendem a defini-lo com base em suas próprias experiências, gerando diversas definições na literatura. Conforme Hancock (1995) pode-se dizer, de uma maneira simplificada, que RV é a forma mais avançada de interface entre o usuário e o computador. Trata-se de uma interface que simula um ambiente real e permite aos participantes interagirem com o mesmo (LATTA, 1994), permitindo às pessoas visualizarem, manipularem e interagirem com representações extremamente complexas (AUKSTAKALNIS, 1992).

1.2 – Aplicações de Realidade Virtual para Treinamento Médico

A Realidade Virtual está proporcionando, sem dúvidas, algumas mudanças no processo de educação médica. Sabbatini (1999), afirma que nos últimos anos, esta área apresentou um salto muito grande nas possibilidades oferecidas para interação homem-máquina, permitindo o desenvolvimento de várias aplicações, sendo a maioria delas ainda se encontram em estágio de teste. Algumas áreas que estão sendo beneficiadas pela RV são: laparoscopia, ortopedia, endoscopia, oftalmologia e oncologia pediátrica.

Sabbatini (1999), descreve a simulação de endoscopia[1], simulação de cirurgia videoendoscópica, simulação de microcirurgias e superposição de imagens médicas.

A simulação de endoscopia consiste na utilização de um computador que gera um modelo artificial, mas altamente realista, do trato gastrointestinal, por exemplo. Manipuladores semelhantes aos de um endoscópio[2] enviam informações para o computador, o qual gera a imagem em consonância com os movimentos realizados com o tubo e a cabeça do endoscópio pelo aluno.

Imagens de endoscopias reais também podem ser acopladas, dando mais realismo ao sistema. Algumas vezes o endoscópio é inserido em um manequim de plástico como mostrado na Figura 1.

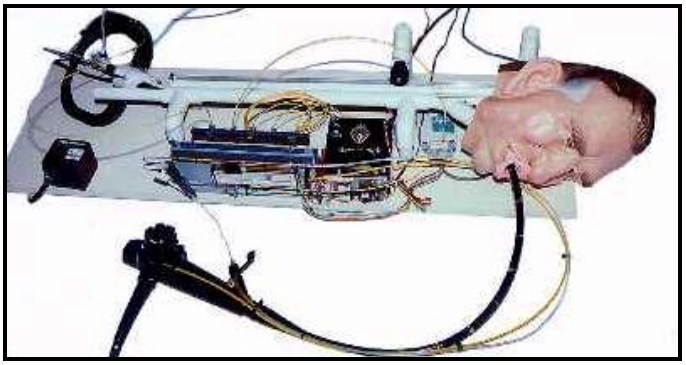


Figura 1– Manequim para Simulação de Endoscopia (SABBATINI, 1999)

Na simulação de microcirurgias, um equipamento especial gera a imagem de vasos e nervos simulando a visualização por um estereomicroscópio e permite sua manipulação com instrumentos cirúrgicos dotados de sensores de posição e atuação. Um dos melhores exemplos

[1] Endoscopia é um exame visual do interior de certos órgãos ou cavidades do corpo humano com um endoscópio. (Michaelis, 2006)

[2] Um endoscópio é como um tubo ótico, rígido ou leve, provido de sistemas de iluminação, para realizar endoscopia**.** (Houaiss, 2006)

é de um simulador de microcirurgias do ouvido médio, que tem a capacidade de mostrar ao aluno qual é o movimento correto dos instrumentos no campo cirúrgico, ilustrado na Figura 2.

Figura 2 – Simulador de microcirurgias do ouvido médio (SABBATINI, 1999)

Conforme Sabbatini (1999), a RV ainda parece ficção científica para o nosso meio, mas o fato é que já existem centenas de aplicações em Medicina e Biologia, particularmente para a área de ensino, que é uma vocação natural da RV em todo o mundo. A escassez de cadáveres e de pacientes para apoiar o ensino médico de anatomia, exames imagenológicos e invasivos, intervenções diagnósticas ou cirúrgicas, tem servido de incentivo para um maior desenvolvimento da RV na pratica médica.

No futuro, com a queda dos preços dos equipamentos causada pela entrada maciça da RV no mercado de entretenimento, a educação médica poderá se beneficiar disso grandemente.

1.3 – *Frameworks* de Realidade Virtual

O reuso de código é uma das principais metas para os desenvolvedores de software. O problema do reuso é que apenas pequenas partes do código podem normalmente ser reaproveitadas. Uma proposta para que uma parcela maior de código seja reutilizada, diminuindo, assim, o tempo de programação, são os *framework*s orientados a objetos.

Conforme Mattsson (2005) cita, *framework* consiste em uma técnica de construção de software que possibilita a reutilização da análise, do design, do código e dos testes. Um *framework* orientado a objetos é definido como um conjunto de classes abstratas e/ou concretas usadas para o desenvolvimento de uma aplicação com domínio específico, isto é, a construção de um *framework* visa a atender uma determinada classe de aplicações.

Há *framework*s desenvolvidos para domínios específicos nas áreas de negócio, medicina, educação, engenharia, entre outras. Um *framework* descreve como será a interação entre as classes e objetos dos sistema, e como será a decomposição do sistema em objetos.

A quantidade de *frameworks* de RV existentes é grande. Nesta monografia serão apresentados alguns exemplos, como o ViRAL (*Virtual Reality Abstraction Layer*), o VPat (*Virtual Patients*), o Avango, e o ViMet, que é utilizado aqui como base para projetar o BD.

Segundo Bastos *et. al.* (2004), o ViRAL surgiu pela carência de um *framework* que fosse pequeno e simples, adequado para sistemas de RV de baixo custo, e ainda, que pudesse ser utilizado por pessoas não especializadas e funcionar em computadores pessoais. O ViRAL está sendo utilizado para construir desde pequenos jogos até complexos visualizadores de modelos CAD.

O ViRAL pode ser visto como uma camada interposta entre as aplicações e os sistemas de RV, como é mostrado na Figura 3. Essa camada é capaz de abstrair particularidades dos sistemas, tais como a forma de capturar entradas ou renderizar gráficos. É um *framework* baseado em componentes, que é um artefato de software, com interfaces internas e externas bem definidas, independente de outros artefatos de software. Este é um conceito central no design do ViRAL: dispositivos de entrada, observadores e ambientes virtuais são todos componentes, que podem ser carregados de bibliotecas e integrados em GUIs (*Graphical User Interface*).

O *framework* é portável 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.

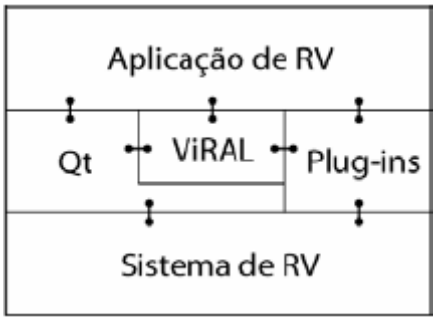


Figura 3 – Camada ViRAL (BASTOS, 2005)

O VPat (FREITAS, 2003) foi projetado com o propósito de suportar o desenvolvimento de aplicações de computação gráfica na área médica. É um *framework* orientado a objetos, que contém classes básicas, com finalidades que podem ser compartilhadas ou estendidas, permitindo o desenvolvimento de classes mais especializadas que implementem, por exemplo, algoritmos complexos de visualização ou simulação de movimento.

O seu modelo conceitual foi construído de maneira a permitir que sistemas de visualização e exploração de dados médicos pudessem ser rápidos e facilmente projetados e desenvolvidos. A utilização mais recente do VPat foi uma extensão para uma aplicação de laparoscopia virtual.

A Figura 4 apresenta o conjunto de classes fundamentais do VPat, na forma de um diagrama UML (*Unified Modeling Language*). As classes foram concebidas e implementadas (em C++) de forma independente de plataforma, assumindo o modelo MVC (*Model-View-Controller*) como base para o desenvolvimento de aplicações.

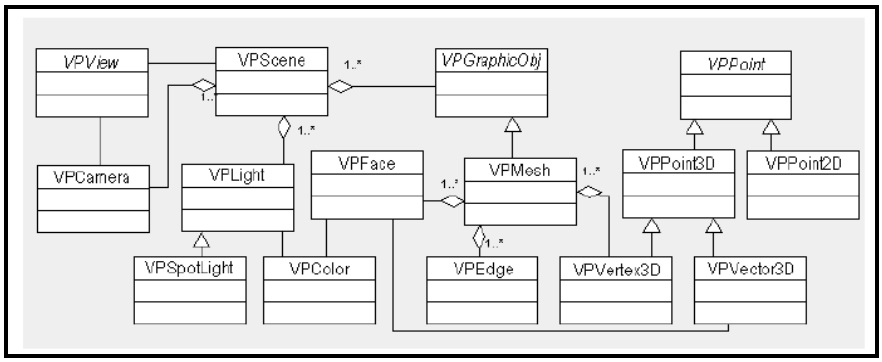


Figura 4 – Diagrama de classes básicas do *framework* VPAT (FREITAS, 2003)

Outro *framework* de RV é o Avango (TRAMBEREND, 1999), criado pelo Instituto IMK/Fraunhofer, um *framework* orientado a objetos para o desenvolvimento de ambientes virtuais distribuídos com foco em sistemas de RV *high-end*.

Tramberend (1999) afirma que o *framework* Avango funciona como um grafo de cena distribuído de forma semitransparente, e provê formas básicas de interação com o mundo virtual (clicar, mover objetos, etc). Ele depende do grafo de cena *Performer*, e é relativamente limitado quanto às aplicações que podem ser criadas.

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 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.

A estrutura permite a criação das classes específicas da aplicação, que herdam estas propriedades da distribuição. Além disso, o grafo de cena compartilhado é aumentado com um grafo distribuído do fluxo de dados. Isso fornece as mesmas características da avaliação em aplicações distribuídas que em aplicações autônomas, e suporta com eficácia o desenvolvimento de aplicações interativas distribuídas.

O programador é protegido dos detalhes de tratar dos gráficos e do sistema de baixo nível, podendo concentrar-se no desenvolvimento da própria aplicação.

Com o Avango, é fornecida uma estrutura que combina o modelo de programação familiar de ferramentas autônomas existentes, com a sustentação interna para a distribuição dos dados que é quase transparente ao colaborador da aplicação.

E por último, o ViMet (OLIVEIRA, 2005), um *framework* em Java para aplicações de treinamento médico usando RV, é um projeto de mestrado que está em fase de desenvolvimento, sendo implementado a partir de aplicações já construídas, disponibilizando um conjunto de classes que facilitem a construção de ferramentas de RV que permita a simulação de procedimentos médicos.

O projeto é realizado no LApIS (Laboratório de Aplicações de Informática em Saúde), constituído por alunos de graduação, com projetos de iniciação científica e trabalhos de conclusão de curso, e por alunos do Programa de Mestrado em Ciência da Computação do UNIVEM, tendo como foco das pesquisas em RV, o desenvolvimento de ferramentas de baixo custo para o treinamento médico.

O ViMet tem o objetivo de facilitar a construção de aplicações de exames de punção, que são procedimentos médicos com o objetivo de coletar material para confirmação ou composição de um diagnóstico médico.

Segundo Oliveira (2005), o *framework* conterá um conjunto de classes básicas que serão responsáveis pela construção do AV, pelo carregamento de objetos (inicialmente um órgão humano sintético e um equipamento sintético), pela renderização da cena, pelos dispositivos de entrada e saída e pela iluminação do ambiente.

Atualmente, a interação com o framework é realizada por dispositivos convencionais, mouse e teclado, sendo previsto a inserção de dispositivos não-convencionais, chamados também de dispositivos hápticos, podendo retornar um *feedback* ao usuário.

Neste capítulo foram apresentados os conceitos básicos de RV, algumas aplicações de RV para treinamento médico e alguns *frameworks* de RV existentes.

No próximo capítulo é apresentado o conceito de Banco de Dados, bem como alguns sistemas que foram projetados ou implementados para aplicações de RV.

CAPÍTULO 2 - BANCO DE DADOS EM APLICAÇÕES DE RV

Segundo Siberschatz, Korth e Sudarshan (1999) um SGBD é constituído por um conjunto de dados associados a um conjunto de programas para acesso a esses dados. O conjunto de dados, comumente chamado Banco de Dados, contém informações sobre uma entidade em particular. O principal objetivo de um SGBD é proporcionar um ambiente tanto conveniente quanto eficiente para a recuperação e armazenamento das informações do Banco de Dados.

Um SGBD é uma coleção de arquivos e programas inter-relacionados que permitem ao usuário o acesso para consultas e alterações desses dados. O maior benefício de um Banco de Dados é proporcionar ao usuário uma visão abstrata dos dados, isto é, o sistema acaba por ocultar determinados detalhes sobre a forma de armazenamento e manutenção desses dados.

Conforme afirmam Elmasri e Navathe (2005), um Banco de Dados é uma coleção de dados relacionados. Os dados são fatos que podem ser gravados e que possuem um significado implícito.

Para que uma aplicação de Realidade Virtual para treinamento médico seja efetiva, é necessário que uma base de dados armazene os casos disponíveis, contendo os objetos e imagens 3Ds, assim como as aplicações geradas, para que seja possível estabelecer uma forma de avaliar seu desempenho.

Coutinho, Porto e Oliveira (2003) apresentam um sistema para armazenar e recuperar objetos tridimensionais (O3D), tais como edificações, porções da superfície terrestre, acidentes naturais e artificiais e todos os demais dados geográficos existentes no mundo real, em um Ambiente Virtual Colaborativo (AVC), para que possam ser feitas consultas num ambiente de Sistemas de Informações Geográficas tridimensionais (SIG-3D). Para tal, faz-se o uso de um SGBD Objeto-Relacional, utilizado para armazenar os O3D, que realiza uma

busca dos que estão visíveis para o observador, à medida que ele se move. A busca é feita através de uma árvore de indexação, recuperando os O3D e possibilitando consultas sobre eles.

Segundo os autores, o estudo de sistemas de SIG-3D é de grande interesse para as Forças Armadas, pois a visualização tridimensional do terreno permite o treinamento de comandantes e tropas, evitando deslocamentos (diminuição do custo operacional) e simulando diversas situações impraticáveis num treinamento real.

Outro sistema que envolve BD na RV é o AVET (Ambiente de Videoconferência para o Ensino Tecnológico), apresentado por Vasconcelos (2002) que trata da concepção e implementação de um ambiente distribuído de videoconferência destinado à comunicação utilizando várias mídias, de forma síncrona e dentro de um mesmo contexto (Aplicação Multimídia Distribuída). É um projeto em desenvolvimento no Laboratório Multiinstitucional de Redes de Computadores e Sistemas Distribuídos (LAR), como parte do projeto Gênesis, tendo como foco em caso particular, o Ensino Tecnológico a Distância (ETD).

O AVET é um ambiente propício à disseminação de conhecimento e troca de informações, que como também permite a execução de aulas práticas utilizando o artifício da RV. Todavia, para sua concepção é necessário contextualizar seus diversos serviços em um ambiente de ETD, levando em consideração alguns pontos críticos: flexibilidade do ambiente, adaptação cultural e exploração dos sentidos.

Para esta aplicação é proposta uma arquitetura cliente/servidor dividida em três camadas: Exibição, onde são agrupados elementos responsáveis pela interação com o usuário (*Player*, *Chat*, etc); Aplicação, que contém os objetos definidos para gerir os recursos e o ambiente; e a camada de Dados, responsável pelo armazenamento das várias mídias distribuídas. Na Figura 5 é mostrado como é dividida a arquitetura do AVET.

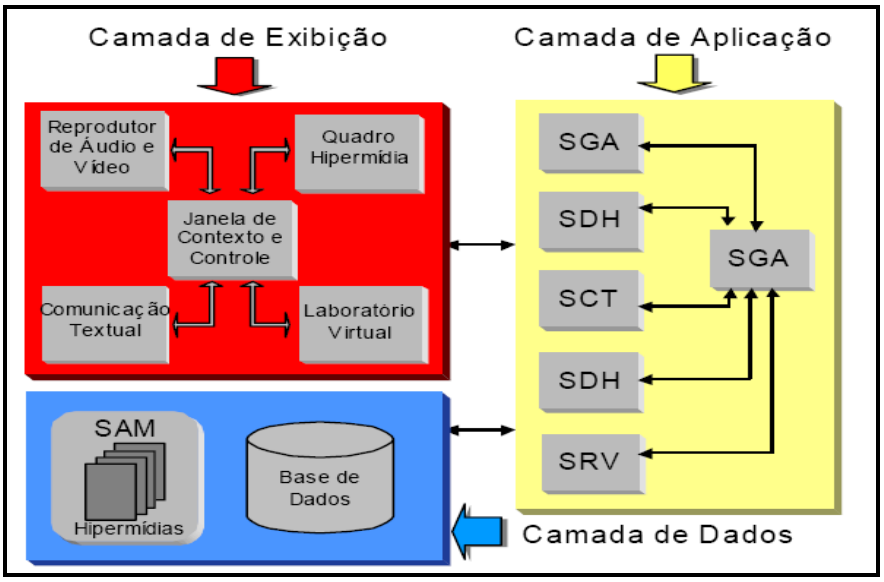


Figura 5 – Arquitetura do AVET (VASCONCELOS, 2002)

A camada de Exibição responde pela formatação dos dados a serem apresentados ao usuário e pela interação com o mesmo. É composta por objetos que realizam a interface de intermédio entre usuários e objetos servidores.

A camada de Aplicação é responsável pela provisão de serviços de gestão de recursos disponíveis no ambiente, pelo gerenciamento do ambiente e pela recepção e transmissão de mídias.

Já a camada de Dados tem como função primordial o armazenamento das mídias e arquivos de *log* que trafegam no sistema. Os arquivos de armazenamento das sessões de videoconferência preservam uma estrutura de diretórios baseada no contexto da sessão. O BD do AVET guarda informações sobre as sessões agendadas no ambiente, os serviços disponíveis durante a sessão de videoconferência, mecanismos que compõem o sistema dos serviços e o agrupamento dos recursos em salas virtuais.

O sistema de BD do AVET armazena as URLs das mídias transmitidas em uma sessão de videoconferência, em um Servidor de Armazenamento de Mídias (SAM).

Neste capítulo foram apresentados os conceitos de BD, e alguns sistemas que foram projetados ou implementados para aplicações de RV, tais como para um Ambiente Virtual Colaborativo, e para o Ensino Tecnológico a Distância.

CAPÍTULO 3 – SGBDs GRATUITOS

Este capítulo tem como objetivo apresentar diversos SGBDs desenvolvidos em Java que podem ser utilizados para a construção do BD. São apresentados dados como autores, utilização, principais características, forma de funcionamento, vantagens e desvantagens, e instalação.

Os SGBDs descritos são: Derby[3], HSQDB[4] e DB4O[5], podendo, assim, fazer-se uma escolha de qual é mais eficiente e objetivo para a construção do BD para um *framework* de RV para treinamento médico com foco em exames de punção.

3.1 Derby

O SGBD *Apache Derby*, um subprojeto do *Apache BD*, é um Banco de Dados relacional implementado completamente em Java e disponível sob a Licença *Apache*, versão 2.0. O *Apache BD* é um projeto do *Apache Software Foundation*, encarregado na criação e manutenção de qualidade-comercial *open-source*, e em soluções de Banco de Dados dos softwares licenciados a *Foundation*.

O *Apache Software Foundation* fornece suporte à comunidade *Apache* em relação a projetos de software *open-source*. Os projetos *Apache* são caracterizados por uma colaboração, consensos baseados em processo de desenvolvimento, uma licença de software pragmática e aberta. O projeto *Apache BD* funciona na meritocracia, ou seja, quanto mais faz, mais responsabilidade obterá.

[3] Referenciado com base no site disponível em: http://db.apache.org/
[4] Referenciado com base no site disponível em: http://hsqldb.org/
[5] Referenciado com base no site disponível em: http://www.db4o.com/

O SGBD *Apache Derby* será referido apenas como Derby no restante do trabalho. O Derby é um Sistema Gerencial de Banco de Dados Relacional (SGBDR) baseado em Java e SQL que pode ser executado em qualquer *Java Virtual Machine* (JVM) certificada. Ele pode ser executado em dois ambientes diferentes: incorporado ou cliente/servidor.

Um ambiente incorporado é um ambiente no qual somente um único aplicativo pode acessar o Banco de Dados de cada vez, sem que ocorra nenhum acesso pela rede. Quando um aplicativo inicia uma instância do Derby na sua JVM, o aplicativo é executado em um ambiente incorporado. A carga do *driver* incorporado inicia o Derby. Assim, ele fica praticamente invisível ao usuário final porque não requer administração e executa na mesma máquina virtual Java (JVM) que o aplicativo.

Um ambiente cliente/servidor é um ambiente no qual vários aplicativos se conectam ao Derby por meio da rede. Ele é executado incorporado a uma estrutura de servidor que permite várias conexões de rede, sendo que a própria estrutura inicia uma instância do Derby e é executada em um ambiente incorporado. Entretanto, os aplicativos clientes não executam no ambiente incorporado.

Em (Anexo A) é apresentada a instalação do SGBD Derby, a configuração do ambiente Java, a utilização das ferramentas e dos utilitários de inicialização, e a configuração manual das variáveis de ambiente.

3.2 HSQLDB

O *Hypersonic SQL Database* (HSQLDB) é um projeto de Banco de Dados livre, escrito em Java, que permite a manipulação de Banco de Dados em uma arquitetura cliente-servidor, ou *standalone*.

Uma grande vantagem de utilização do HSQLDB é a possibilidade de agregar o Banco de Dados ao pacote das aplicações. Ele é multi-plataforma e ocupa um pequeno espaço

em disco. Outra característica é a possibilidade de manipular Bancos de Dados em disco, memória, ou em formato texto. Trata-se de uma tecnologia flexível e muito útil na construção de aplicações que manipulam Banco de Dados.

No núcleo do pacote do HSQLDB estão o SGBD e o *driver* JDBC para conexão por meio de aplicações Java, que disponibilizam as principais funcionalidades.

Além disso, o pacote contém um conjunto de componentes e ferramentas para execução do SGBD. Por meio das ferramentas é possível criar estruturas de um BD, acessar o BD pelas ferramentas de consulta, exportar e importar esquemas entre BD distintos, além de outras facilidades disponibilizadas para o desenvolvedor.

A descrição de cada um dos componentes do HSQLDB é apresentada a seguir:

- HSQLDB JDBC *Driver*: disponibiliza um *driver* padrão JDBC para conexão de aplicações Java com o SGBD. A conexão com o BD segue um modelo de protocolo proprietário, mas também é possível realizar uma conexão via rede, por meio de protocolos Internet.

- Gerenciador de BD (*Database Manager*): duas versões de ferramentas para o gerenciamento de BD são disponibilizadas. Uma ferramenta usando *AWT*[6] (*Abstract Window Toolkit*), e uma outra versão usando *Swing*[7]. Trata-se de uma ferramenta gráfica para visualização do esquema do BD, conjunto de tabelas e submissão de instruções SQL. A versão AWT pode ser executada como um Applet dentro de um navegador.

- Ferramenta de Transferência (*Transfer Tool*): é uma ferramenta utilizada para transferências de esquemas SQL ou dados de uma fonte JDBC para outra. Trata-se de uma ferramenta bastante útil quando se deseja realizar uma migração de BD, transferindo esquemas e conjunto de dados entre duas tecnologias distintas.

---

[6] *AWT* é um pacote que contém as classes Java e interfaces exigidas para criar e manipular interfaces gráficas com o usuário. (Deitel, 2003).

[7] *SWING* contém classes e interfaces para os componentes GUI Swing de Java que fornecem suporte para GUIs portáveis. (Deitel, 2003)

- Ferramenta de Consulta (*Query Tool*): a finalidade dessa ferramenta é prover ao desenvolvedor um software para interação com o SGBD por meio de envio de instruções SQL a partir de uma linha de comando, ou por um arquivo texto contendo um conjunto de instruções. A ferramenta apresenta um *shell* interativo ao usuário.
- SQL *Tool*: é uma outra ferramenta do pacote para construção e submissão de instruções SQL ao BD.

O HSQLDB pode ser executado como um servidor de BD, ou como um processo em aplicações *standalone*. Para cada modo de execução há uma forma específica de conexão com o BD. A forma de conexão é dada por um protocolo, informado na string da URL de conexão com o BD.

Para executar o gerenciador no modo servidor, deve-se invocar o programa *Server*. O programa recebe alguns argumentos para que se possa iniciar ou criar um novo BD. É possível passar alguns argumentos por linha de comando, um deles é o argumento *"-?"* que apresenta uma ajuda na tela.

Uma forma de chamada ao servidor, iniciando o BD denominado *DBEmpresa* apenas como exemplo, é apresentada a seguir:.

java org.hsqldb.Server -database.0 DBEmpresa -dbname.0 empresa

O primeiro argumento *-database.0* informa o nome do BD, enquanto que o segundo, *-dbname.0*, informa um alias para o BD.

Caso o servidor seja carregado sem informar algum BD, o mesmo gera uma base padrão denominado *test*.

A execução no modo servidor Web é uma modalidade, a qual permite a conexão de uma aplicação Java via protocolo HTTP. Essa característica é útil quando se tenta disponibilizar o BD em máquinas protegidas por firewalls.

A forma de execução no modo servidor Web é a seguinte:

java org.hsqldb.WebServer

Aplicações clientes realizam uma conexão a um servidor em execução via JDBC, conforme mostrado como exemplo na Figura 6.

A conexão do exemplo utiliza a porta *"9001"*, que é a porta padrão para conectar a aplicação Java, via JDBC, com o BD *DBEmpresa*. O usuário padrão do HSQLDB é o *"sa"*, cuja senha é uma string vazia.

```
try {
        Class.forName ("org.hsqldb.jdbcDriver");
}
catch (SQLException e) {
        System.out.println ("Erro ao carregar o driver JDBC.");
}
Connection con = DriverManager.getConnection ("jdbc:hsqldb:hsql:/localhost/DBEmpresa",
"sa", "");
```

Figura 6 – Exemplo de conexão via JDBC

Na conexão de uma aplicação no modo *standalone*, o BD faz parte da própria aplicação. Ou seja, ambos são empacotados juntos e ambos rodam em uma mesma JVM. Para pequenas aplicações que são executadas em um desktop, sem o uso da rede, é um conceito muito interessante. A Figura 7 mostra um exemplo de como realizar uma conexão com um BD *standalone*.

```
Connection con = DriverManager.getConnection ("jdbc:hsqldb:file:/banco/DBEmpresa",
"sa", "");
```

Figura 7 – Exemplo de conexão modo *standalone*

O HSQLDB também permite executar o SGBD de forma que os dados sejam mantidos em memória. Essa é uma técnica que mantém todo o conjunto de dados de uma tabela na memória do computador, permitindo uma melhor performance do SGBD.

Um exemplo de conexão utilizando o protocolo de memória, é demonstrado na Figura 8.

```
Connection con = DriverManager.getConnection (“jdbc:hsqldb:mem:empresa”, “sa”, “”);
```

Figura 8 – Exemplo de conexão utilizando o protocolo de memória

Finalizando, o SGBD HSQLDB é uma ferramenta muito flexível para manipulação de um BD. Trata-se de uma ferramenta para trabalhar-se com Java e para construção de pequenas aplicações *standalone*. Para a área voltada ao desenvolvimento de software é uma escolha, pois o software cabe em um disquete, tornando-se muito prático. Na área de educação, trata-se de uma excelente opção para o ensino de conectividade de BD com JDBC.

3.3 Db4o

O Db4o é um BD *open source* que possibilita aos desenvolvedores Java e .Net reduzir o tempo e o custo de desenvolvimento e alcançar níveis altos de performance.

O design do mecanismo nativo da base de objetos faz com que ele seja ideal para aplicações embarcadas em equipamentos e dispositivos, executado tanto em desktop como dispositivos móveis, ou em sistemas de controle de tempo real, ou seja, nos ambientes Java e .Net.

Todos os desenvolvedores de programas orientados a objetos sabem das dificuldades encontradas para passar um modelo orientado a objetos para uma persistência relacional. São forçados a escolher entre velocidade e orientação a objetos: o acesso nativo ao SQL é rápido, mas trabalhoso, requerendo um enorme esforço extra para codificação.

Mapeadores objeto-relacional oferecem uma ponte conveniente, no entanto eles degradam a performance. O Db4o elimina a troca orientada a objetos por performance, permitindo armazenar complexas estruturas de objetos atingindo altos níveis de performance.

Avaliações de BDs mostram que o Db4o pode ser até 44 vezes mais rápido que o Hibernate (HIBERNATE, 2006) e o MySQL (MYSQL, 2006), uma combinação popular de

mapeador objeto-relacional e BD. A maior razão para utilizar BDR hoje em dia é o legado, como por exemplo, o armazenamento de antigas informações da empresa e o conjunto de aplicações existentes para eles.

Mas além da persistência centrada em servidor, existem milhares de dispositivos, celulares e aplicações desktop onde a tecnologia convencional de BD não alcança. A tecnologia do Db4o assegura novos níveis de performance, funcionalidade e eficácia de custo.

Neste capítulo foram apresentados alguns SGBD gratuitos desenvolvidos em Java, com o objetivo de fornecer uma idéia de qual SGBD utilizar no projeto, discutido no capítulo seguinte.

CAPÍTULO 4 – IMPLEMENTAÇÃO DO BANCO DE DADOS

Neste capítulo será apresentado o ambiente de programação utilizado na construção do BD e o sistema construído para um *framework* orientado a objetos utilizando técnicas de RV na área médica, com foco em exames de punção.

4.1 Apresentação do ViMet

O ViMet, conforme apresentado no capítulo 2, um *framework* em Java para aplicações de treinamento médico usando RV, é um projeto de mestrado que está em fase de desenvolvimento, sendo implementado a partir de aplicações já construídas, disponibilizando um conjunto de classes que facilitem a construção de ferramentas de RV que permita a simulação de procedimentos médicos.

O ViMet tem o objetivo de facilitar a construção de aplicações de exames de punção, que são procedimentos médicos com o objetivo de coletar material para confirmação ou composição de um diagnóstico médico.

O *framework* é do tipo de extensão caixa cinza, e utiliza a linguagem de programação Java e API Java 3D, sendo multiplataforma. A construção de uma ferramenta de instanciação automática, o Wizard, facilita a manipulação do *framework*.

A partir do modelo de análise de domínio criou-se o projeto arquitetural do ViMeT. Foram previstas a utilização do SGBD Derby, a criação da camada de persistência que irá garantir a flexibilidade na manutenção ou migração do BD. O projeto arquitetural do ViMeT é representado na Figura 9 e permite obter uma visão geral do *framework.* Também é representada as duas formas possíveis de instanciação do ViMeT, sendo uma delas utilizando diretamente as classes e a outra utilizando a ferramenta de instanciação automática, o *Wizard*.

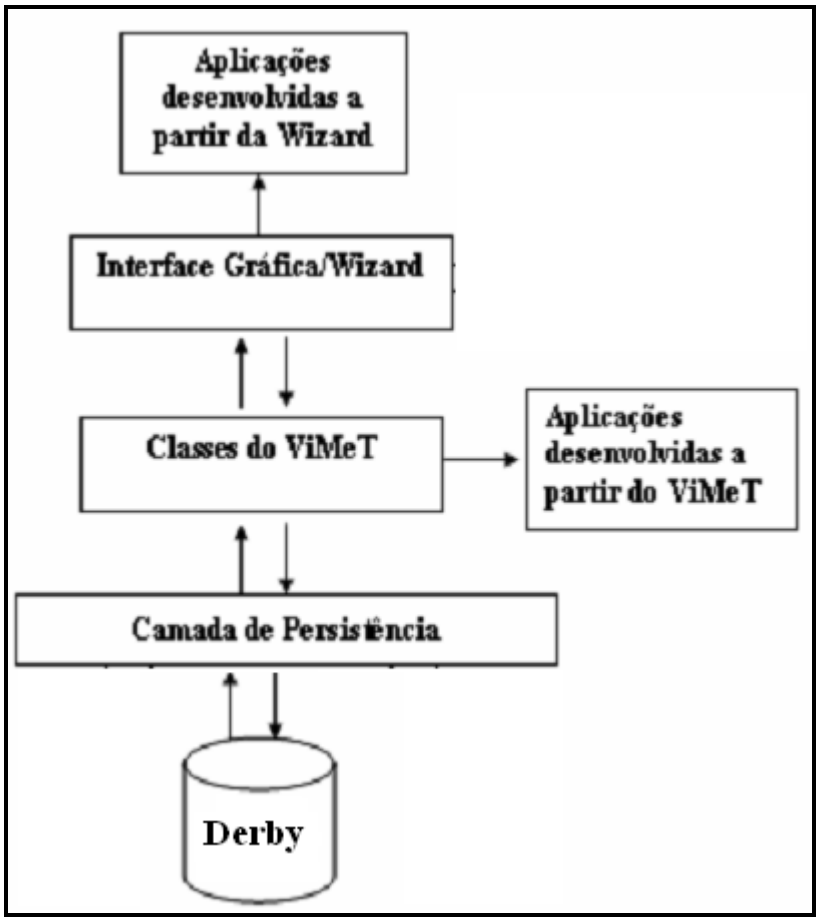


Figura 9 – Projeto arquitetural do ViMet

4.2 Ambiente de Programação

Para o desenvolvimento desse projeto foi estudada a linguagem de programação Java, que pode ser caracterizada por ser uma linguagem simples, orientada a objetos, robusta, segura, independente de arquitetura, portável, de alto desempenho, dinâmica, interpretada, *multithreaded* e distribuída. São por estas características, por permitir a reusabilidade de código e por ser uma linguagem de tecnologia gratuita, é que foi escolhida para o desenvolvimento do projeto.

A partir dos estudos realizados sobre Banco de Dados baseados em Java, foi escolhido o SGBD Derby por vários motivos. Primeiramente pelo fato de ser um software de tecnologia gratuita e *open-source,* que permite o desenvolvimento de uma aplicação com baixo custo e com acesso ao código fonte. Além disto, oferece as opções de implementações,

conforme citado anteriormente, incorporado ou cliente/servidor, que se pode estar escolhendo o melhor método a desenvolver. Outro fator importante é que por ser escrito inteiramente em Java, pode ser executado em qualquer JVM certificada, e também por ser um BD leve, levando em consideração, que os sistemas de RV para treinamento na área da saúde exigem um tempo de resposta curto.

4.3 Funcionamento do Sistema

O BD desenvolvido apresenta o esquema geral ilustrado na Figura 10.

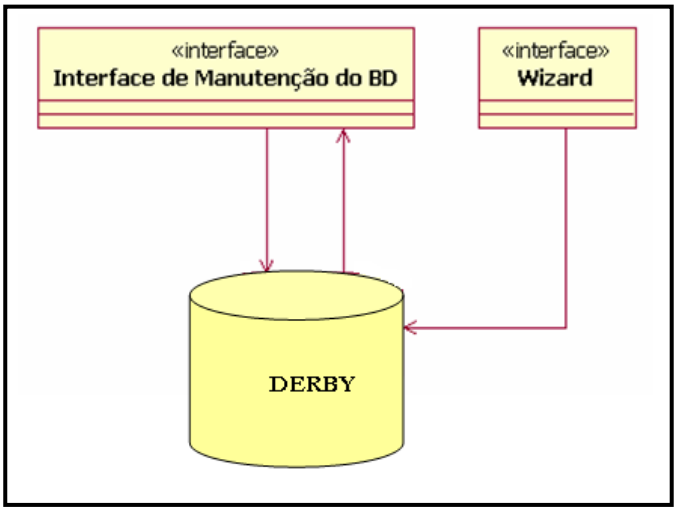


Figura 10 – Esquema Geral do BD

A interface de Manutenção do BD grava e consulta a Base, enquanto o *Wizard*, por enquanto, somente grava no Banco, sendo proposta para continuação deste trabalho a recuperação das aplicações geradas pelo Wizard salvos no BD.

Em algumas partes desta monografia, os intrumentos médicos e os órgãos 3Ds serão chamados apenas de objetos, para uma simplificação.

Na fase inicial do projeto, são criadas duas tabelas de armazenamento dos objetos e das aplicações realizadas pelos usuários durante o treinamento. A primeira tabela, OBJETOS, mostrada na Figura 11, armazena os órgãos humanos 3Ds, podendo ser uma mama, perna, glúteo, braço, entre outros, e os intrumentos médicos utilizados nos treinamentos, podendo ser

uma agulha, seringa, bisturi, entre outros. A tabela contém o código do objeto como chave primária, o tipo, (se é um órgão ou um instrumento médico), uma descrição detalhada do mesmo, e um último campo chamado de imagem, onde é guardado o caminho do objeto inserido.

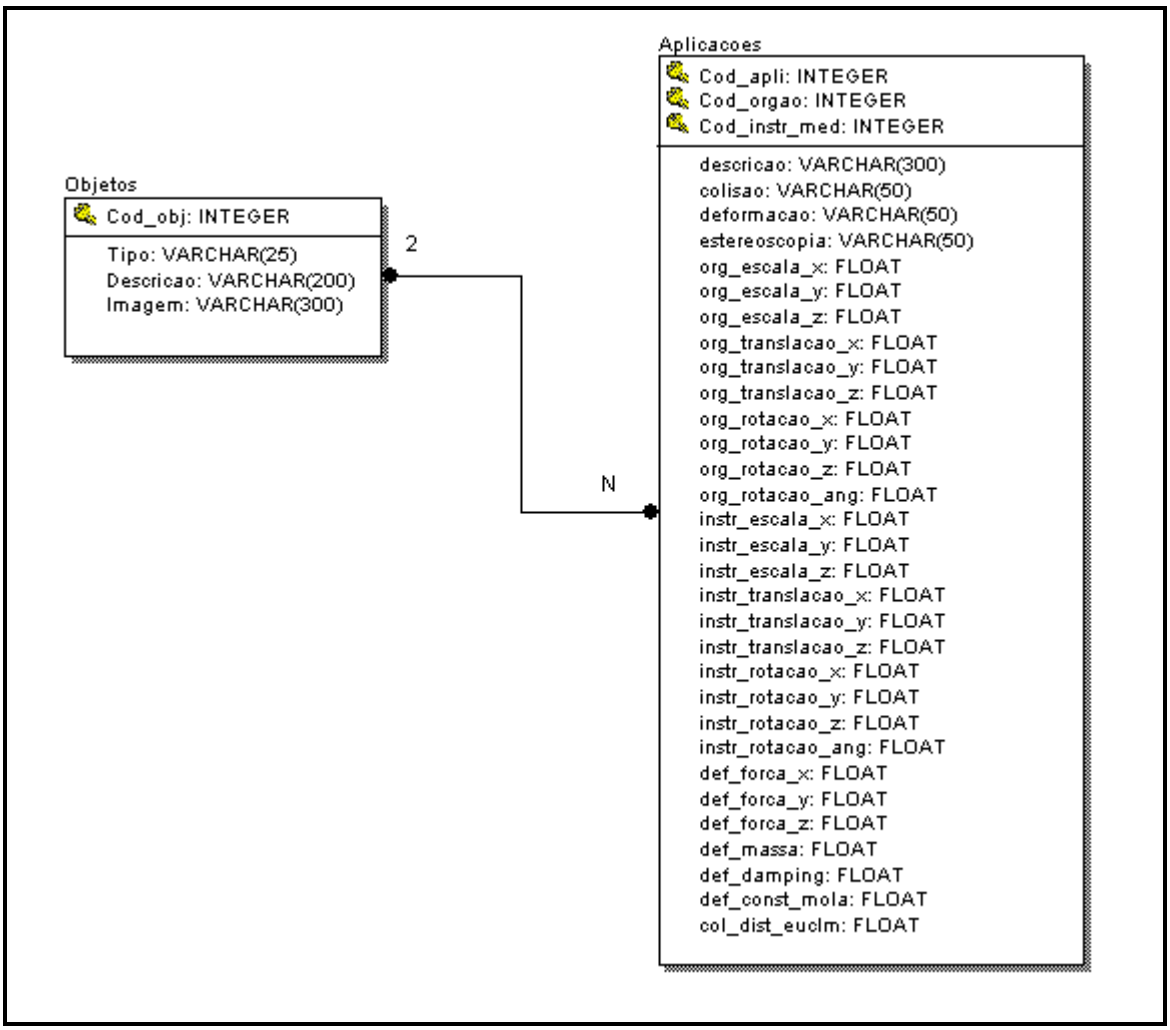


Figura 11 – Projeto do BD

O relacionamento das duas tabelas é N (muitos) para 2, ou seja, uma aplicação tem exatamente sempre dois objetos, e um objeto pode estar em muitas aplicações.

A segunda tabela, APLICAÇÕES, é utilizada para gravar as aplicações para uma posterior recuperação. A tabela contém um campo para o código da aplicação como chave primária, código do órgão inserido na tabela OBJETOS, como chave estrangeira, código do instrumento médico inserido na tabela de OBJETOS, como chave estrangeira, descrição da

aplicação realizada, os métodos de colisão, deformação e estereoscopia utilizados, as escalas (x, y, z), translações (x, y, z) e rotações (x, y, z, ângulo) dos objetos, a força (x, y, z), massa, *damping* e constante da mola do método da deformação e a distância euclidiana no método de colisão. O método de estereoscopia, ainda está em fase de desenvolvimento no ViMet.

Em seguida, foi traçado um plano do projeto, com o objetivo de identificar os passos a seguir na implementação. Assim sendo, foram compostas classes independentes a partir de uma classe principal, *Wizard*, com o objetivo de facilitar a manipulação e o entendimento de cada parte do sistema, conforme mostra o diagrama de classes da Figura 12.

A seguir é apresentado na Tabela 1, um resumo das finalidades de cada classe do sistema, para uma melhor síntese do leitor.

Tabela 1 – Descrição das finalidades das classes do sistema

Classes	Finalidade
Wizard	Instancia a classe *Application*, *FrameworkDatabaseApplication*, *BuildCode*, *Parameters*, *ParamOrgan*, *ParamMedInstr*, *ParamCollision*, *ParamDeformation* e *ParamEst*.
Application	Responsável pelo *framework* ViMet
Environment	Responsável pela criação do ambiente 3D
FrameworkDatabaseApplication	Responsável por tratar o BD
Interface	Interface de manutenção do BD
BuildCode	Responsável por gerar o código Java da aplicação
Parameters	Responsável por tratar os parâmetros dos objetos e das técnicas de colisão, deformação e estereoscopia
ParamOrgan	Responsável por ler os parâmetros dos órgãos
ParamMedInstr	Responsável por ler os parâmetros dos instrumentos médicos
ParamCollision	Responsável por ler os parâmetros da técnica *Octree*
ParamDeformation	Responsável por ler os parâmetros da técnica *MassSpring*
ParamEst	Responsável por ler os parâmetros da técnica *Anaglyph*

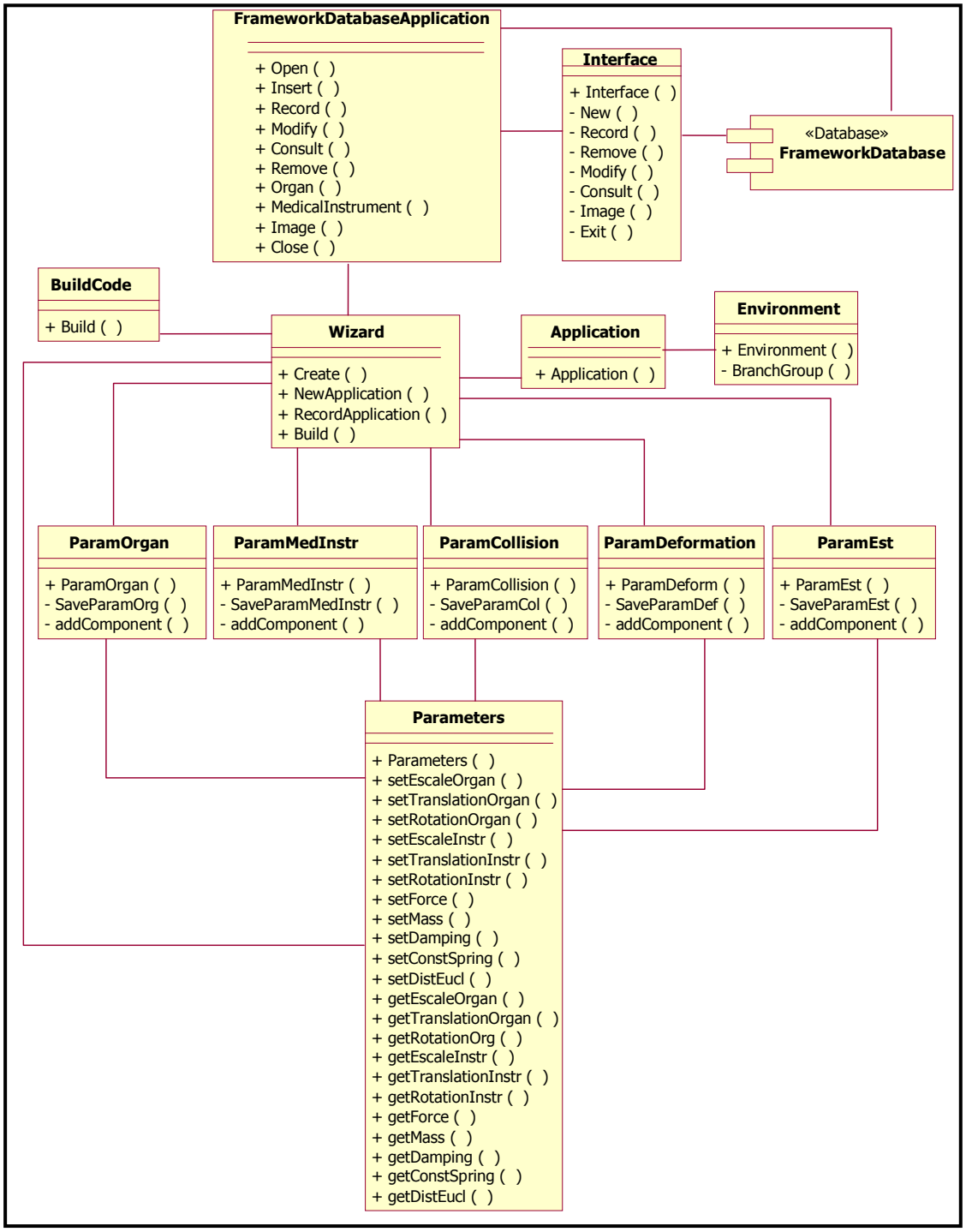


Figura 12 – Diagrama de Classes

4.3.1 Manutenção do Banco de Dados

A classe "*Interface*" é a responsável pela manutenção dos objetos armazenados no BD, utilizados durante o treinamento médico. Esta classe é construída sob um *JFrame* e

contém um método construtor chamado "*Interface*", além de outros métodos explicados a seguir.

A interface permite ao usuário inserir, remover, alterar e consultar os objetos guardados na Base de Dados, instanciando a classe "*FrameworkDatabaseApplication*", onde são implementados os métodos de tratamento do BD, conforme mostra a Figura 13.

Na inserção, o usuário escolhe o tipo do objeto, se é um órgão ou um instrumento médico, fornece uma descrição detalhada do mesmo para uma futura consulta, e busca o caminho do objeto a ser gravado. Para remover um objeto da Base, o usuário seleciona o campo de descrição do qual será removido, como por exemplo: agulha, mama, seringa, etc. O usuário também pode alterar o tipo, a descrição e a imagem de um objeto já armazenado, buscando-o pela descrição. A consulta também é feita pela descrição do objeto, mostrando seu código que é de forma automática na inserção, o tipo, a descrição e a imagem, que é o caminho onde está salvo.

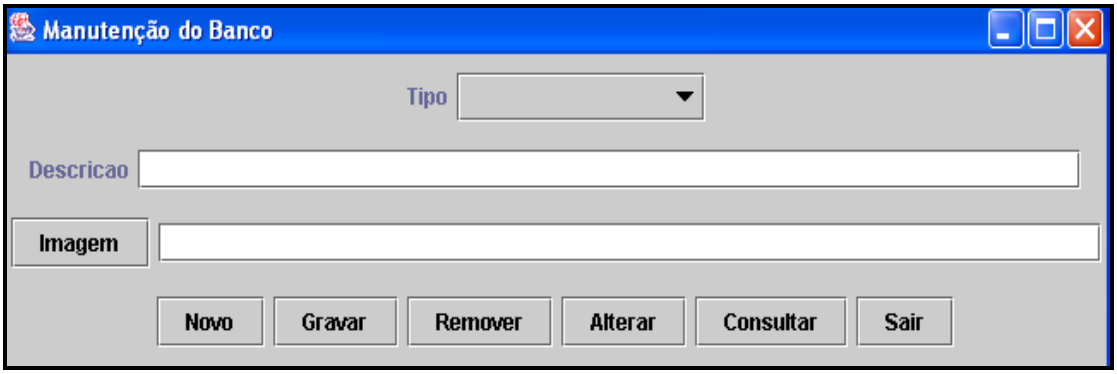


Figura 13 – Interface de Manutenção do Banco de Dados

4.3.2 *Wizard*

Um *Wizard* é uma ferramenta para automatizar o processo de instanciação do *framework* e gerar aplicações específicas.

Conforme já explicado anteriormente, o Wizard, interage com o framework por dispositivos convencionais, mouse e teclado, sendo previsto a inserção de dispositivos não-

convencionais, também chamados de dispositivos hápticos, retornando assim um *feedback* ao usuário.

A classe "*Wizard*" foi implementada para instanciar as aplicações feitas pelos usuários. Esta classe instancia as classes "*BuildCode"*, "*FrameworkDatabaseApplication"*, "*Application"*, "*Parameters"*, "*ParamOrgan"*, "*ParamMedInstr"* e dos métodos a utilizar, no caso, *Octree* na colisão, *MassSpring* na deformação e *Anagliph* na estereoscopia. A interface pode ser observada na Figura 14.

As classes "*ParamOrgan*", "*ParamMedInstr*", "*ParamCollision*" e "*ParamDeformation*" são todas construídas sob um *JFrame,* e usa como gerenciador de *layout* o *GridBagLayout*. Para utilizá-lo deve-se construir um objeto *GridBagConstraints.* Esse objeto especifica como um componente é colocado em um *GrigBagLayout*.

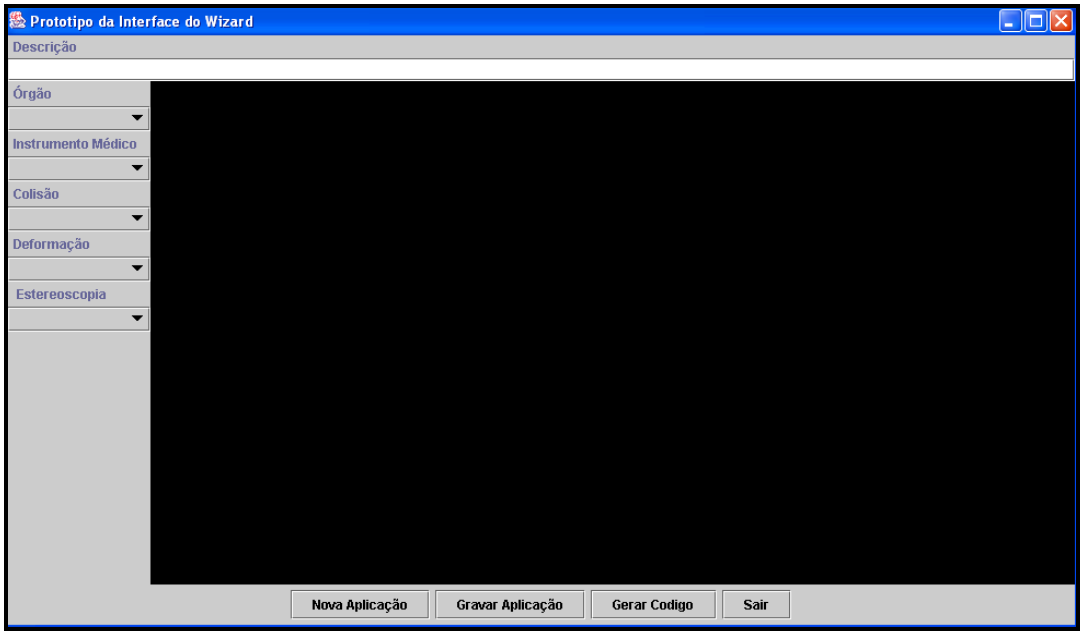


Figura 14 – Protótipo da Interface do *Wizard*

Ao escolher o órgão a carregar, a classe "*ParamOrgan*", é instanciada pela classe "*Wizard*", onde recebe por parâmetro um objeto do tipo "*Parameters*", abrindo uma nova janela, na qual o usuário pode definir os parâmetros de localização do órgão a ser carregado

no ambiente. Os parâmetros consistem em escala (x, y, z), translação (x, y, z) e rotação (x, y, z, ângulo), conforme mostrado na Figura 15.

Os parâmetros a serem gravados são passados pelo método *set* através de um objeto do tipo da classe "*Parameters*". A classe "*Wizard*" chama o método *get* da classe "*Parameters*", e passa por parâmetro para a classe "*FrameworkDatabaseApplication*", onde será armazenada na tabela chamada APLICAÇÕES do BD.

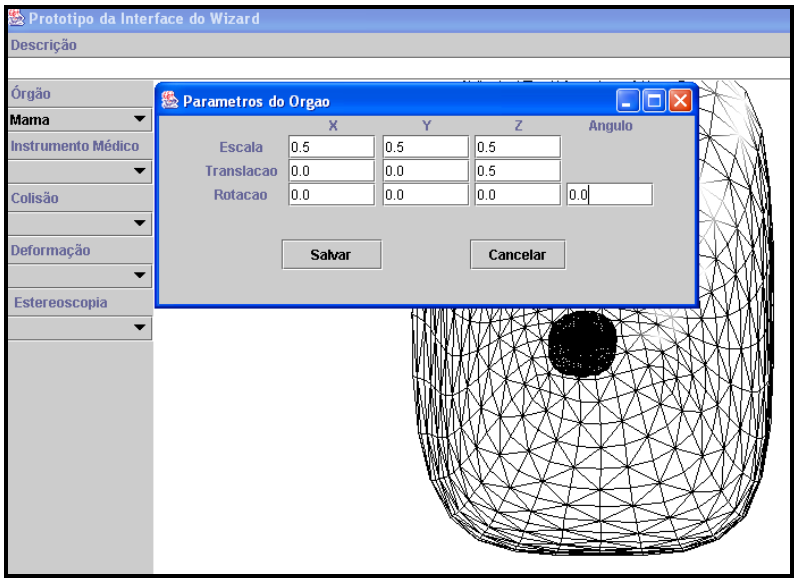


Figura 15 - Parâmetros do Órgão

Os parâmetros do instrumento médico são formados da mesma forma: a classe "*ParamMedInstr*" é instanciada pela classe "*Wizard*", recebendo por parâmetro um objeto do tipo "*Parameters*" e abrindo uma nova janela onde o usuário pode definir os parâmetros de localização do instrumento a ser carregado no ambiente. Os parâmetros consistem em escala (x, y, z), translação (x, y, z) e rotação (x, y, z e ângulo), conforme mostrado na Figura 16.

Os parâmetros são gravados da mesma forma que os parâmetros do órgão. Os parâmetros a serem gravados, são passados pelo método *set* através de um objeto do tipo de classe "*Parameters*". A classe "*Wizard*" chama o método *get* da classe "*Parameters*", e passa por parâmetro para a classe "*FrameworkDatabaseApplication*", onde armazena na tabela chamada APLICAÇÕES do BD.

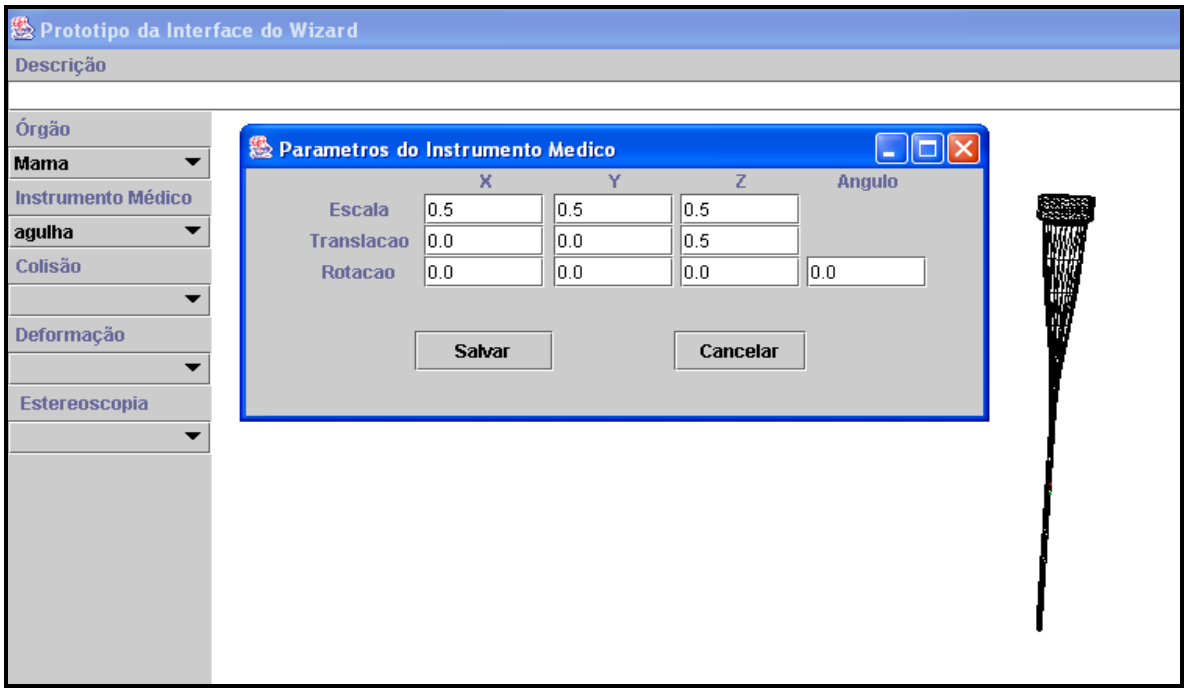


Figura 16 - Parâmetros do Instrumento Médico

A classe "*ParamCollision*" é a responsável pelos parâmetros da técnica de colisão, no caso o *Octree*, onde o único parâmetro a definir é a distância euclidiana, conforme mostrado na Figura 17.

Esta classe também é instanciada pela classe "*Wizard*", que passa por parâmetro um objeto do tipo da classe "*Parameters*". Para salvar o parâmetro, é chamado o método *set* pelo objeto da classe "*Parameters*".

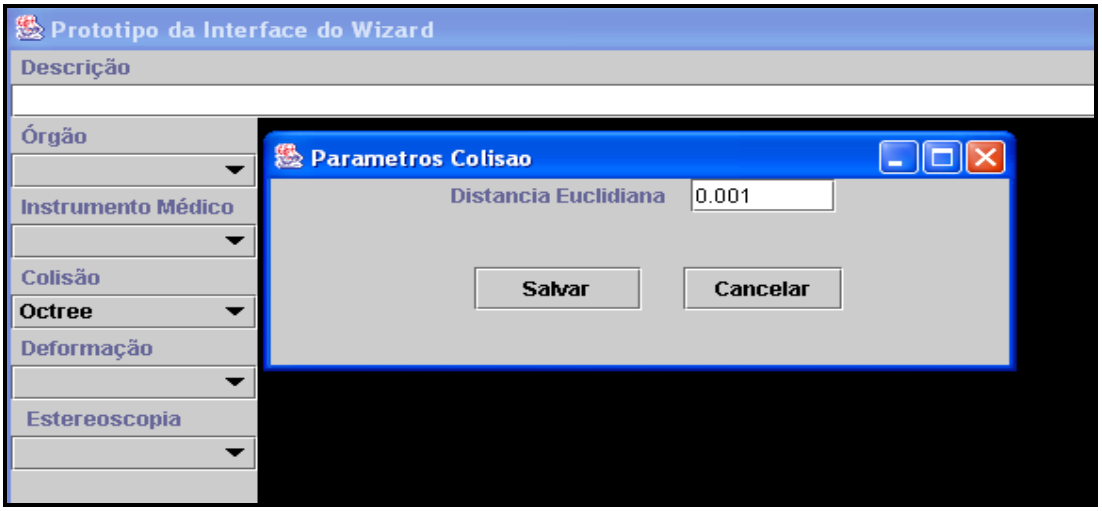


Figura 17 - Parâmetros do Método de Colisão *Octree*

A classe "*ParamDeformation*" é a responsável pelos parâmetros da técnica de deformação, que no caso, usa o método *MassSpring*, onde os parâmetros a definir são a força (x, y, z), a massa, o *damping* e constante da mola, conforme mostrado na Figura 18.

Esta classe também é instanciada pela classe "*Wizard*", que passa por parâmetro um objeto do tipo da classe "*Parameters*". Para salvar o parâmetro, é chamado o método *set* pelo objeto da classe "*Parameters*".

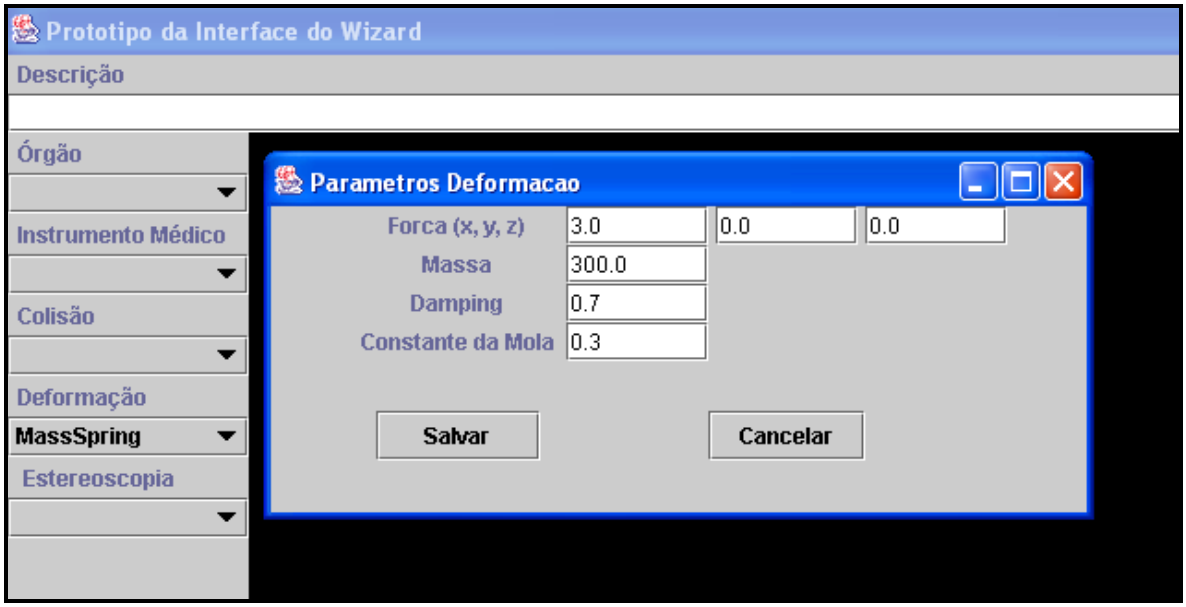


Figura 18 - Parâmetros do Método de Deformação *MassSpring*

A classe "*BuildCode*" é a responsável pela geração de um código em Java da aplicação que o usuário realiza, para que seja possível a recuperação da aplicação posteriormente. Esta classe é instanciada pelo "*Wizard"* que por sua vez instancia a classe "*Application"* que contém o código da aplicação a ser gerada. O código é gerado em uma outra classe chamada "*BuiltApplication*". Um trecho do código gerado é ilustrado na Figura 19.

```
def = ((ObjDef)objetos[0]).getDeformation();

p.setForce(3.0f,0f,0f);
p.setMass(300f);
p.setDamping(0.7f);
p.setConstSpring(0.3f);

def.setParameters(p);

cd = ((ObjDef)objetos[0]).getCollisionDetector();
cd.setPrecision(0.000001);
cd.setCollisionListener(def);
```

Figura 19 – Trecho do Código Gerado pela Classe "*BuildCode*"

No trecho de código, são mostrados os parâmetros de força, massa, *damping* e constante de mola do método de deformação sendo aplicados em um objeto deformável, e também, adicionando o método de colisão no objeto e a distância euclidiana com precisão.

4.3.3 Manutenção das Aplicações

Após gerada a aplicação no *Wizard* pelo usuário, é permitido que o mesmo possa fazer alterações nos parâmetros pela interface de manutenção das aplicações, conforme é mostrado na Figura 20. O usuário pode alterar, remover consultar e gerar o código da aplicação.

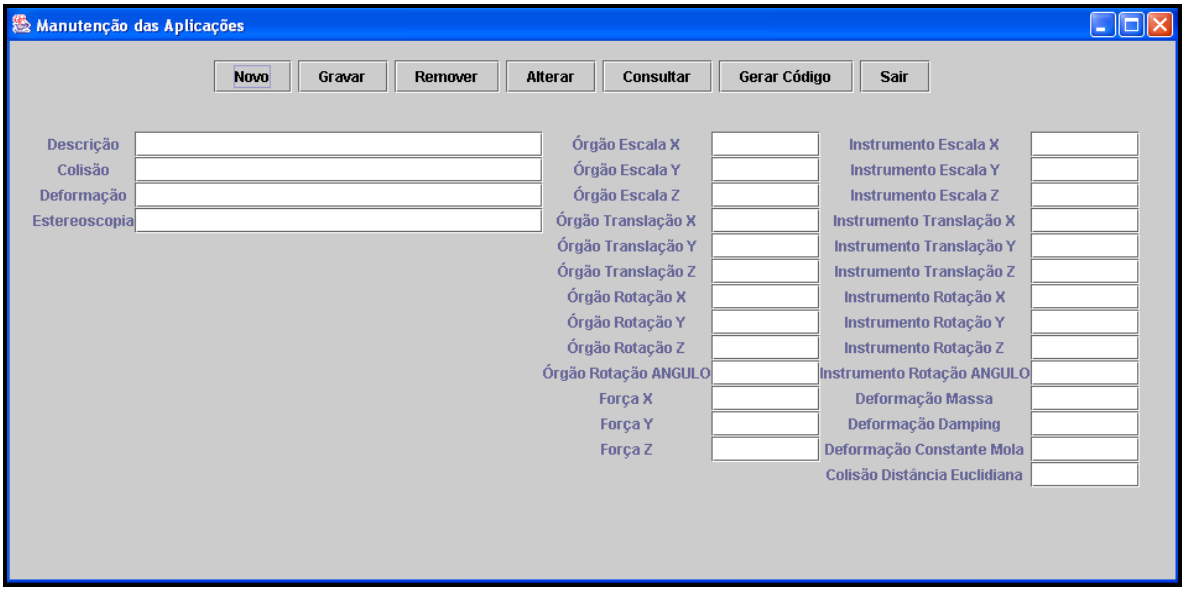


Figura 20 – Manutenção das Aplicações

Neste capítulo foi apresentado o ambiente de programação utlilizado para construir o BD, a manutenção dele, o funcionamento do projeto, o *Wizard* construído e a manutenção das aplicações. No próximo capítulo estão os resultados obtidos do sistema, realizados por dois estudos de caso.

CAPÍTULO 5 – RESULTADOS E DISCUSSÕES

Este capítulo tem como objetivo apresentar os estudos de casos realizados para testar o BD e o *Wizard* construídos. Os estudos de casos foram realizados com quatro objetos com extensão .obj, construídos em uma ferramenta de modelagem 3D chamada *Blender* (BLENDER, 2006), também gratuita, por alunos de Iniciação Científica da Fundação de Ensino "Eurípides Soares da Rocha", Centro Universitário "Eurípides de Marília" – UNIVEM.

5.1 Primeiro Estudo de Caso

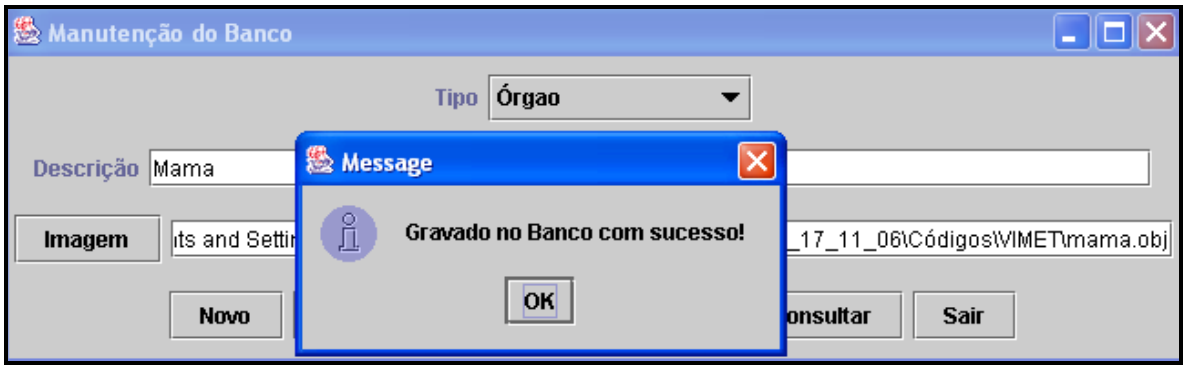
O *framework* a testar, o ViMet, conforme dito anteriormente, é um *framework* orientado a objetos que utiliza técnicas de RV para a área da saúde, com foco em procedimentos médicos minimamente invasivos, mais específico, em exames de punção.

O ViMet possui uma classe chamada "*Environment*", que é a responsável pela criação do ambiente 3D, conforme a Figura 21, onde os objetos 3Ds são carregados e, conseqüentemente, realizados os treinamentos.

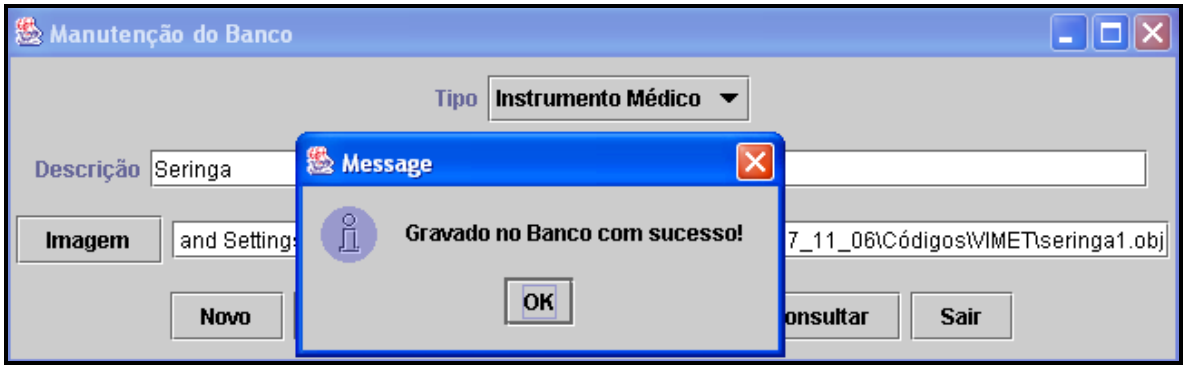
```
public Environment(Canvas3D c)
{
        myLocale = new Locale(this);
        myLocale.addBranchGraph(buildViewBranch(c));

        BranchGroup brbg = new BranchGroup();
        BoundingSphere bounds = new BoundingSphere(new Point3d(0.0, 0.0, 0.0), 100);
        Color3f bgColor = new Color3f(1.0f, 1.0f , 1.0f);
        Background bg = new Background(bgColor);

        bg.setApplicationBounds(bounds);
        brbg.addChild(bg);
        myLocale.addBranchGraph(brbg);
}
```

Figura 21 – Método construtor *Environment*

O primeiro estudo de caso foi realizado com dois objetos com extensão .obj, sendo um órgão e um instrumento médico. O órgão escolhido é uma mama e o instrumento médico uma seringa, conforme mostra a Figura 22 (a) e (b) respectivamente, sendo inseridos no BD pela interface de manutenção.



(a)

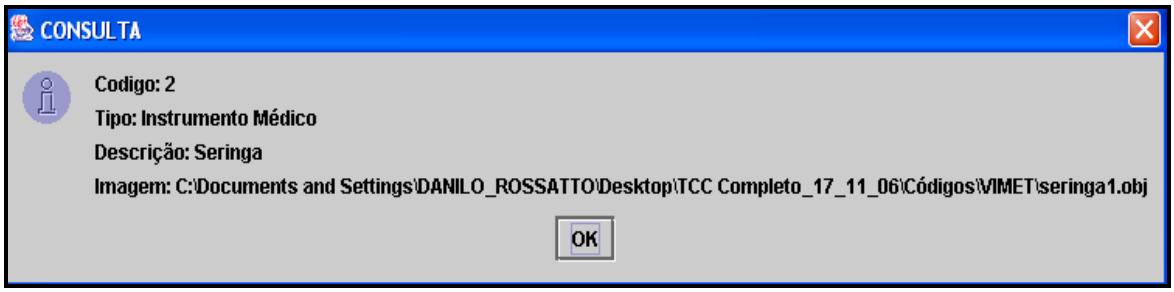


(b)

Figura 22 – Inserção no BD: (a) Mama; (b) Seringa

Após inseridos os objetos que serão utilizados no primeiro estudo de caso, a tabela OBJETOS fica confome mostrada na Figura 23 (a) e (b).

(a)



(b)

Figura 23 – Tabela OBJETOS: (a) Órgão; (b) Instrumento Médico

No *Wizard* o usuário escolhe o órgão e o instrumento médico a serem utilizados, em um *JComboBox*, que estão armazenados na tabela OBJETOS da Base de Dados. Assim, o *Wizard* instancia a classe "*Application*", que é a classe pai da classe "*Environment*", carregando os objetos no ambiente, e uma tela para definir os parâmetros que o usuário desejar.

A Figura 24, mostra a mama e a seringa carregadas no mesmo ambiente.

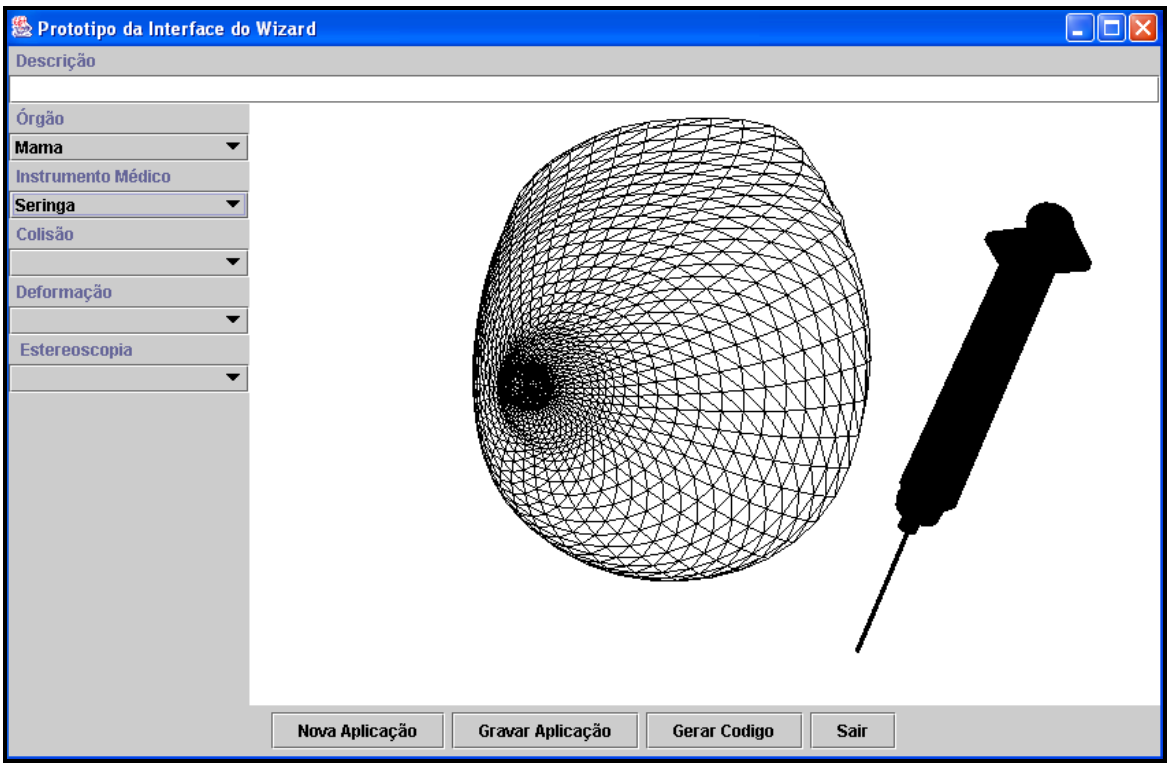
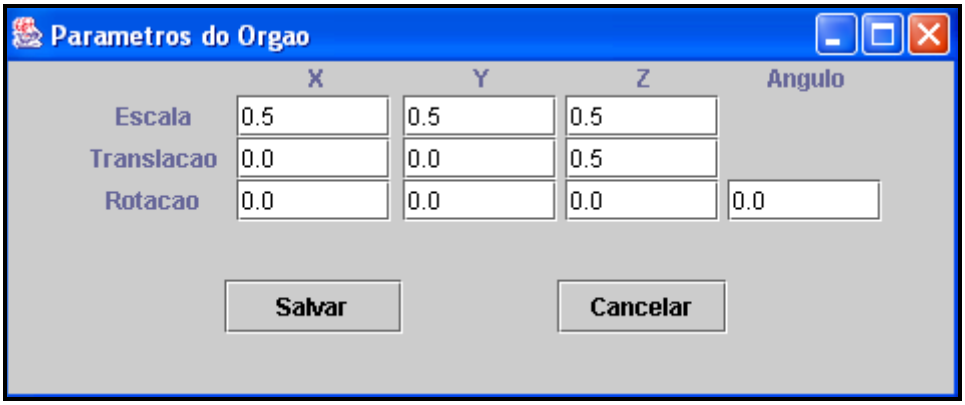
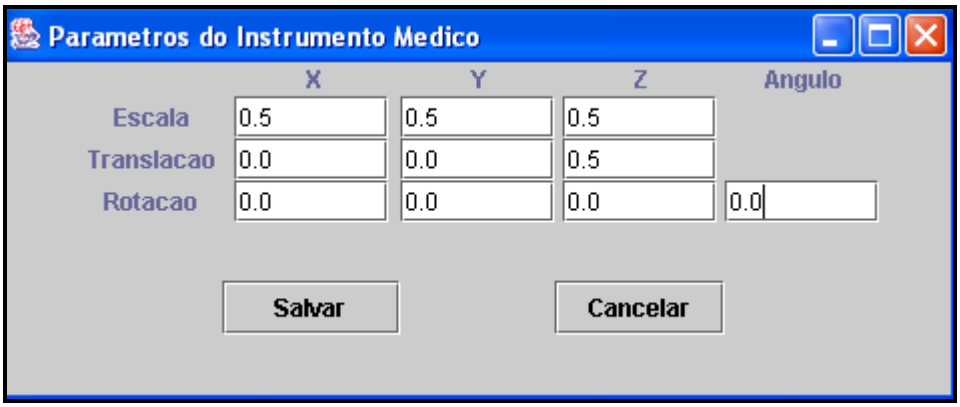


Figura 24 – Mama.obj e Seringa.obj Carregados no Ambiente do *Framework*

Os parâmetros *defaults* dos órgãos e dos instrumentos médicos são mostrados nas Figura 25 (a) e (b), respectivamente. As Figura 17 e Figura 18, mostram os parâmetros *defaults* das técnicas do método de colisão e do método de deformação, respectivamente.



(a)



(b)

Figura 25 – Parâmetros *Default* dos Objetos: (a) Órgão; (b) Instrumentos Médicos

As aplicações podem ser gravadas para uma futura recuperação. Conforme explicado anteriormente, são salvas na tabela chamada APLICAÇÕES, com todos os seus parâmetros e sua descrição detalhada. Uma demonstração da tabela APLICAÇÕES é apresentada na Figura 26.

Depois de gravada a aplicação, o usuário pode escolher gerar o código da aplicação que acabou de construir. O código é gerado em um arquivo com extensão .java, contendo os

objetos e as técnicas de colisão, deformação e estereoscopia, que o usuário utilizou durante o treinamento, juntamente com seus parâmetros.

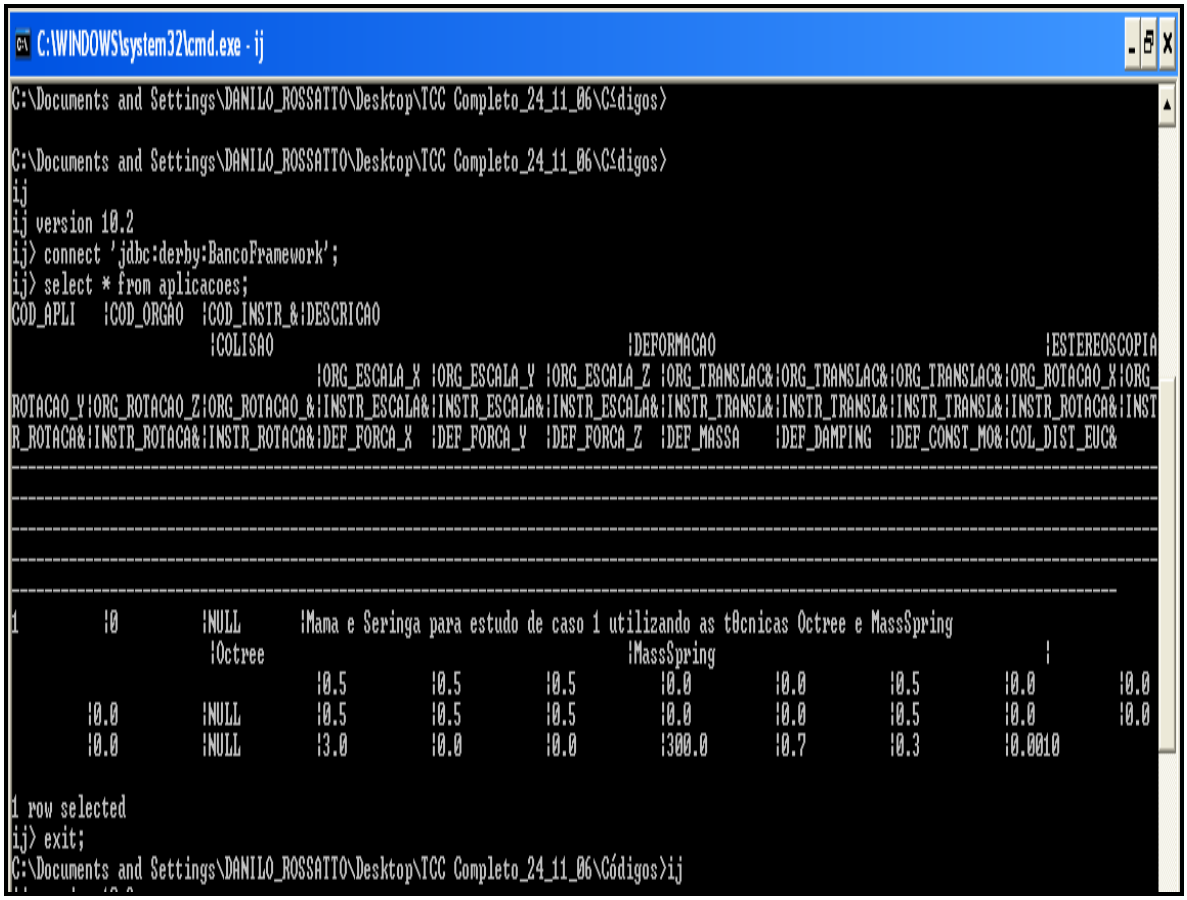
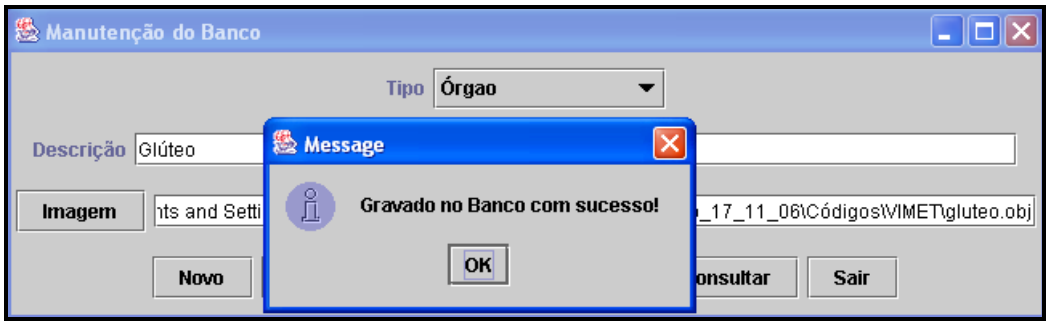


Figura 26 – Tabela APLICAÇÕES

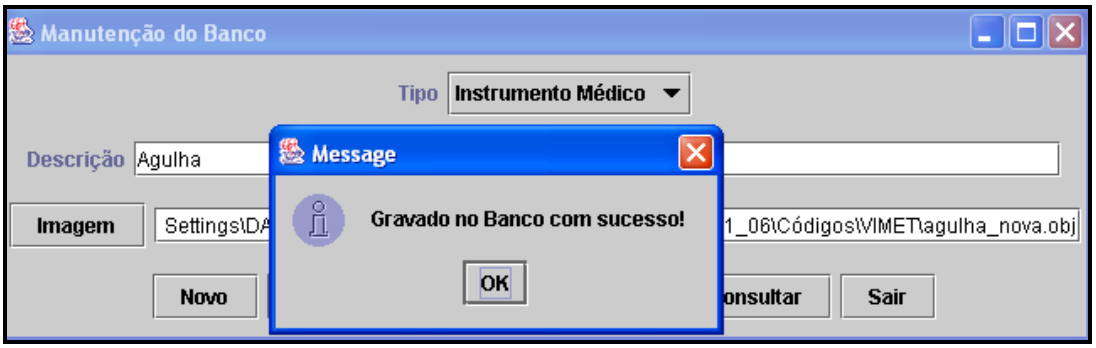
O código gerado desta aplicação é apresentado no Apêndice J.

5.2 Segundo Estudo de Caso

O segundo estudo de caso, também foi realizado com dois objetos com extensão .obj, um órgão e um instrumento médico. O órgão é um glúteo e o instrumento médico uma agulha, conforme mostra a Figura 27 (a) e (b) respectivamente, sendo inseridos no BD pela interface de manutenção.



(a)



(b)

Figura 27 – Inserção no BD: (a) Glúteo; (b) Agulha

Após inseridos os objetos que serão utilizados no estudo de caso, a tabela OBJETOS fica conforme mostrada na Figura 28.

(a)

(b)

Figura 28 – Tabela OBJETOS: (a) Órgão; (b) Instrumento Médico

No *Wizard* o usuário faz o mesmo procedimento explicado no primeiro estudo de caso: escolhe o órgão e o instrumento médico a serem utilizados no treinamento, que estão armazenados na tabela OBJETOS da Base de Dados. Assim, o *Wizard* instancia a classe "*Application*", que por sua vez carrega a classe "*Environment*", carregando os objetos no ambiente, e uma tela que permite definir os parâmetros que o usuário desejar.

A Figura 29, mostra o glúteo e a agulha carregadas no mesmo ambiente.

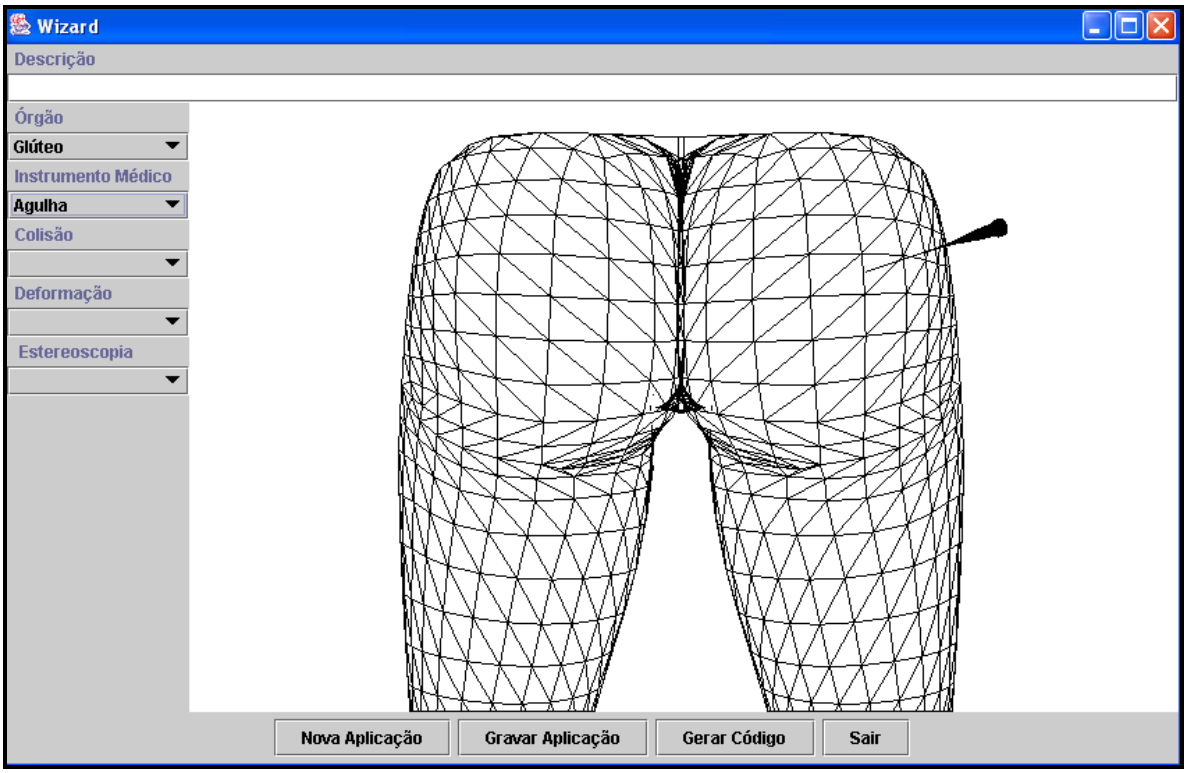


Figura 29 – Glúteo.obj e Agulha.obj Carregados no Ambiente do *Framework*

A aplicação, conforme explicado anteriormente, é gravada na tabela chamada APLICAÇÕES, com todos os seus parâmetros e sua descrição detalhada. Uma demonstração da tabela APLICAÇÕES é apresentada na Figura 26.

Depois de gravada a aplicação, o usuário pode escolher gerar o código da aplicação que acabou de fazer. O código é gerado em um arquivo com extensão .java, contendo os objetos e as técnicas de colisão, deformação e estereoscopia, que o usuário utilizou durante o treinamento, juntamente com seus parâmetros.

O código gerado desta aplicação é apresentado no Apêndice K.

Visto que um BD é essencial para que um *framework* seja efetivo, esta monografia tem como objetivo a construção de uma ferramenta de auxílio para tal *framework*, contribuindo assim, para o ensino de anatomia e fisiologia a estudantes de Medicina.

Este trabalho permite ao usuário uma facilidade de manuseio através do *Wizard*, assim, obtendo um ganho de produtividade, não sendo necessário conhecer as classes e gerando um código sem erro, obtidos a partir dos estudos de casos gerados. É possível também alterar os parâmetros e gerar novas aplicações.

A seguir, é apresentada a conclusão do trabalho e alguns trabalhos futuros que podem complementar esta monografia para melhorias do sistema.

CONCLUSÕES E TRABALHOS FUTUROS

O objetivo do trabalho aqui apresentado foi construir uma Base de Dados e uma interface de manutenção do mesmo, para teste, a partir de um *Wizard*, do ViMet, um *framework* orientado a objetos para aplicações de treinamento médico utilizando técnicas de RV, tendo como foco os exames de punção,.

Os resultados iniciais obtidos com o sistema foram satisfatórios após uma série de aplicações realizadas dentro do ambiente do *framework*. Entretanto, o sistema deverá passar por testes mais exaustivos executados a partir de novas técnicas de colisão, deformação e estereoscopia implementadas futuramente, representando um ganho de qualidade do sistema, tornando-o ainda mais abrangente e portável para a utilização em outras ferramentas de treinamento da área médica.

Assim, através deste trabalho foi possível perceber que a utilização do sistema facilita os testes do *framework* para treinamento.

Além da avaliação com o *ViMet*, o sistema futuramente pode ser avaliado com outros *frameworks* orientados a objetos utilizando-se técnicas de RV na medicina, variando-se os parâmetros aqui testados.

Como continuidade deste trabalho e com o objetivo de obter um maior grau de performance do BD, outros estudos de casos devem ser gerados, por exemplo, usar a mama juntamente com uma agulha, um glúteo com uma seringa entre outros, alterando os parâmetros aqui usados, e também, migrar o SGBD aqui usado, como por exemplo, para o HSQLDB e outros, para fins de avaliações de performance entre eles, obtendo um melhor resultado.

REFERÊNCIAS BIBLIOGRÁFICAS

ARAÚJO, R. B. **Especificação e análise de um sistema distribuído em realidade virtual**, São Paulo, Junho, 144 Pp., Tese (Doutorado), Departamento de Engenharia de Computação e Sistemas Digitais, Escola Politécnica da Universidade de São Paulo, 1996.

AUKSTAKALNIS, S., BLATNER, D. **Silicon mirage: the art and science of virtual reality**, Berkeley, CA, 1992.

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**:** *Proceedings of SVR 2004* - VII Symposium on Virtual Reality. São Paulo: SBC - Brazilian Computer Society, p. 51 – 62, 2004.

BLENDER – **Blender 3D Graphics Creation**. Disponível em: < http://www.blender.org/ > Acesso em: Outubro de 2006.

BOTEGA, L. C.; NUNES, F. L. S.; OLIVEIRA, A. C. M. T. G.; PAVARINI, L. **Implementação de Módulo de Estereoscopia Utilizando Anaglifos para Ferramentas de Realidade Virtual para Treinamento Médico**. Disponível em: < www.sbc.org.br/bibliotecadigital/download.php?paper=459 > Out, 2005. Acesso em: Novembro de 2006.

COUTINHO, E. S.; PORTO, F. A. M.; OLIVEIRA, J. C. de. **Armazenamento e Recuperação de Objetos em Ambientes Virtuais Colaborativos para SIG - 3D**, *Proceedings of VI Symposium on Virtual Reality*, pp. 453-460, Out, 2003.

DB4O – **Native Java & .NET Object Database**. Disponível em: < http://www.db4o.com/ > Acesso em: Maio de 2006.

DEITEL, H. M.; DEITEL, P. J. **Java, Como Programar**. Trad. Carlos Arthur Lang Lisbôa. – 4.ed. – Porto Alegre: Bookman, 2003. 1386p.

DERBY – **The Apache DB Project**. Disponível em: < http://db.apache.org/ > Acesso em: Maio de 2006.

ELMASRI, R.; NAVATHE, S. B.. **Sistemas de Banco de Dados**. São Paulo: Pearson Addison Wesley, 2005. 724p.

ENDOSCOPIA. In: Michaelis, Moderno Dicionário da Língua Portuguesa. Disponível em: < http://www2.uol.com.br/michaelis/indexdic.htm?busca=endoscopia&busca2=endoscopia > Acesso em: Março de 2006.

ENDOSCÓPIO. In: Houaiss, Dicionário da Língua Portuguesa. Disponível em: < http://houaiss.uol.com.br/busca.jhtm?verbete=endosc%F3pio&stype=k > Acesso em: Março de 2006.

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 of SVR 2003 - VI Symposium on Virtual Reality. Porto Alegre: SBC - Brazilian Computer Society, v. 6, p. 283-294, 2003.

HANCOCK, D. **Viewpoint: virtual reality in search of middle ground,** *IEEE Spectrum*, 32(1):68, January, 1995.

HIBERNATE – **Hibernate**. Disponível em: < http://www.hibernate.org/ > Acesso em: Maio de 2006.

HSQLDB – **HSQL Database Engine**. Disponível em: < http://hsqldb.org/ > Acesso em: Maio de 2006.

LATTA, J. N., OBERG, D. J. **A conceptual virtual reality model**, *IEEE Computer Graphics & Applications*, pp. 23-29, Jan., 1994.

MATTSSON, M. **Object-Oriented Frameworks – A survey of metholological issues,** (Tese de Doutorado) – Departament of Computer Science, UCK, 1996. Disponível em: < http://www.ipd.bth.se/michaelm/paper/ Mattsson.Lic.thesis.pdf > Acesso em: Março de 2006.

MYSQL – **MySQL Enterprise**. Disponível em: < http://www.mysql.com/ > Acesso em: Maio de 2006.

OLIVEIRA, A. C. de M. T. G. de. **VIMET – Um *Framework* em Java para Aplicações de Treinamento Médico Usando Realidade Virtual**. 2005. 103 f. Grau: (Qualificação de Mestrado em Ciência da Computação) – Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2005.

SABBATINI, R. M. E. **Aplicação – Realidade Virtual no Ensino Médico**. 1999. Disponível em: < http://www.informaticamedica.org.br/informaticamedica/n0202/sabbatini.htm > Acesso em: Março de 2006

SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. **Sistema de Banco de Dados**. São Paulo: Makron Books, 1999. 778p.

SORID, D.; MOORE, S.K. **The Virtual Surgeon**. IEEE Spectrum, v.37, n.7, p.26-31, Julho de 2000.

TRAMBEREND, H. **AVANGO: A Distributed Virtual Reality *Framework***. In: *Proceedings of IEEE Virtual Reality*, pages 14–22, Houston, TX, USA, 1999.

VASCONCELOS, M. A.; ELLERY, C. E.; SERRA, A. B.; RODRIGUES, M. A. L. **Um Ambiente Distribuído de Videoconferência para Comunicação Multimídia.** In: SBC – Sociedade Brasileira de Computação – Revista Eletrônica de Iniciação Científica, Ano II, Volume II, Número II, Junho de 2002. Disponível em: < http://www.sbc.org.br/reic/edicoes/2002e2/cientificos/UmAmbienteDistribuidoDeVideoconferenciaParaComunicacaoMultimida.pdf > Acesso em: Junho de 2006.

APÊNDICE A – CÓDIGO FONTE WIZARD.JAVA

```
//======== Bibliotecas necessárias ================
import DefApliMedLoader.ObjectFile;

import java.awt.event.*;
import java.awt.*;

import java.util.Vector;

import javax.media.j3d.*;
import javax.swing.*;
import javax.vecmath.*;

//************************ CLASSE WIZARD ****************************************
public class Wizard
{
	public JComboBox combo1, combo2, combo3, combo4, combo5;
	public JTextField jTextFieldDescricao;

	public String Colisao[]= {" ", "Octree"};
	public String Deforma[]= {" ", "MassSpring"};
	public String Estereo[]= {" ", "Anagliph"};

	String arquivo = "";

	public Point3f escala_org, trans_org, rot_org, escala_instr_med, trans_instr_med, rot_instr_med, forca;
	public float dist_eucl, massa, damp, cte ;

	Parameters p = new Parameters();
	FrameworkDatabaseApplication ap = new FrameworkDatabaseApplication();
	GerarCodigo gc = new GerarCodigo();

//============================ MÉTODO CRIAR ================================
	private void Criar()
	{
		ap.AbrirBanco();

		final Canvas3D c = new Canvas3D(null);

		Application application = new Application(c);

		final JFrame window = new JFrame();

		window.setTitle("Wizard");
      window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      window.setSize(850, 550);
		window.setLocation(100,0);

		final Container container = window.getContentPane();

		container.add(c);

	    JPanel p_menus = new JPanel(new GridLayout(2,2));
		JPanel p_combo = new JPanel(new GridLayout(20,30));
		JPanel p_button = new JPanel(new FlowLayout());

//============== Botão Nova Aplicação =========
		JButton btn_NovaApli = new JButton(" Nova Aplicação ");
      btn_NovaApli.addActionListener(new ActionListener()
```

```
                {
                  public void actionPerformed(ActionEvent evt)
                  {
                                NovaAplicacao();
        }
       });
      p_button.add(btn_NovaApli);

//============== Botão Gravar Aplicação =========
                JButton btn_GravarApli = new JButton(" Gravar Aplicação ");
      btn_GravarApli.addActionListener(new ActionListener()
                {
                  public void actionPerformed(ActionEvent evt)
                  {
                                GravarAplicacao();
        }
       });
      p_button.add(btn_GravarApli);

//============== Botão Gerar Codigo =========
                JButton btn_GerarCod = new JButton(" Gerar Código ");
      btn_GerarCod.addActionListener(new ActionListener()
                {
                  public void actionPerformed(ActionEvent evt)
                  {
                                Gerar();
        }
       });
      p_button.add(btn_GerarCod);

//========= Botão Sair ============
                JButton btn_Sair = new JButton(" Sair ");
      btn_Sair.addActionListener(new ActionListener()
                {
       public void actionPerformed(ActionEvent evt)
                  {
                                ap.FecharBanco();
                System.exit(0);
        }
      });
      p_button.add(btn_Sair);

//======= Combo p/ seleção do objeto 3D(orgão) armazenado no BD ======
                JLabel lb_def= new JLabel("  Órgão  ");
                p_combo.add(lb_def);
                Vector org = new Vector(30);
                org = ap.Orgao();
                combo1= new JComboBox(org);
                combo1.setMaximumRowCount(3);
                combo1.addItemListener(new ItemListener()
                {
                        public void itemStateChanged(ItemEvent event)
                        {
                                String des = (String)combo1.getSelectedItem();
                                arquivo = ap.CaminhoImagem(des);

                                ParamOrgan paramO = new ParamOrgan();
                                paramO.ParamOrgan(p);

                                return;
```

```
                }
            });
            p_combo.add(combo1);

//====== Combo p/ seleção do objeto 3D(instrumento cirurgico armazenado no BD ======
            JLabel lb_Estereo= new JLabel("  Instrumento Médico    ");
            p_combo.add(lb_Estereo);
            Vector instruMed = new Vector(30);
            instruMed = ap.InstrMed();
            combo2= new JComboBox(instruMed);
            combo2.setMaximumRowCount(3);
            combo2.addItemListener(new ItemListener()
            {
                public void itemStateChanged(ItemEvent event)
                {
                    String des = (String)combo2.getSelectedItem();
                    arquivo = ap.CaminhoImagem(des);

                    ParamMedInstr paramI = new ParamMedInstr();
                    paramI.ParamMedInstr(p);

                    return;
                }
            });
            p_combo.add(combo2);

//========== Combo p/ escolha da detecção de colisão =========
            JLabel lb_orgao= new JLabel("  Colisão   ");
            p_combo.add(lb_orgao);
            combo3= new JComboBox(Colisao);
            combo3.setMaximumRowCount(3);
            combo3.addItemListener(new ItemListener()
            {
                public void itemStateChanged(ItemEvent event)
                {
                    ParamCollision paramC = new ParamCollision();
                    paramC.ParamCollision(p);
                }
            });
            p_combo.add(combo3);

//============ Combo p/ escolha da deformação ============
            JLabel lb_coli= new JLabel("  Deformação  ");
            p_combo.add(lb_coli);
            combo4= new JComboBox(Deforma);
            combo4.setMaximumRowCount(3);
            combo4.addItemListener(new ItemListener()
            {
                public void itemStateChanged(ItemEvent event)
                {
                    ParamDeformation paramD = new ParamDeformation();
                    paramD.ParamDeformation(p);
                }
            });
            p_combo.add(combo4);

//================ Combo p/ escolha da Estereoscopia =====
            JLabel lb_inst= new JLabel("   Estereoscopia ");
            p_combo.add(lb_inst);
            combo5= new JComboBox(Estereo);
```

```
                combo5.setMaximumRowCount(3);
                combo5.addItemListener(new ItemListener()
                {
                        public void itemStateChanged(ItemEvent event)
                        {
                                //ParamEst paramE = new ParamESt();
                                //paramE.ParamEst(p);
                        }
                });
                p_combo.add(combo5);

//=================== DESCRICAO ======================================
                JLabel lb_desc = new JLabel("  Descrição ");
                p_menus.add(lb_desc);
                jTextFieldDescricao = new JTextField();
                p_menus.add(jTextFieldDescricao);

                container.add(p_menus,BorderLayout.NORTH);
        container.add(p_button,BorderLayout.SOUTH);
                container.add(p_combo,BorderLayout.WEST);

                window.setVisible(true);
        }

//=================== MÉTODO NOVA APLICAÇÃO ==============================
        public void NovaAplicacao()
        {
                combo1.setSelectedIndex(0);
                combo2.setSelectedIndex(0);
                combo3.setSelectedIndex(0);
                combo4.setSelectedIndex(0);
                combo5.setSelectedIndex(0);
        }

//=================== MÉTODO GRAVAR APLICAÇÃO ============================
        public void GravarAplicacao()
        {
                boolean ok = false;

                String desc = jTextFieldDescricao.getText();

                String col = (String)combo3.getSelectedItem();
                String def = (String)combo4.getSelectedItem();
                String est = (String)combo5.getSelectedItem();

                escala_org = p.getEscalaOrg();
                trans_org = p.getTranslacaoOrg();
                rot_org = p.getRotacaoOrg();

                escala_instr_med = p.getEscalaInstr();
                trans_instr_med = p.getTranslacaoInstr();
                rot_instr_med = p.getRotacaoInstr();

                dist_eucl = p.getDistEucl();

                forca = p.getForce();
                massa = p.getMass();
                damp = p.getDamping();
                cte = p.getConstSpring();
```

```
                    ok = ap.Gravar(escala_org, trans_org, rot_org, escala_instr_med, trans_instr_med,
rot_instr_med, forca, massa, damp, cte, dist_eucl, col, def, est, desc);

                    if (ok)
                JOptionPane.showMessageDialog(null, "Aplicação gravada com sucesso!");
            else
                JOptionPane.showMessageDialog(null, "Problema na gravação da aplicacao!");
        }

//================== MÉTODO GERAR CODIGO ====================================
        public void Gerar()
        {
                    boolean ok = true;

                    ok = gc.Gerar();

                    if (ok)
                JOptionPane.showMessageDialog(null, "Código gerado com sucesso!");
            else
                JOptionPane.showMessageDialog(null, "Problema na geração de código!");
        }

//===================== PRINCIPAL ===========================================
        public static void main(String[] args)
        {
      Wizard frame = new Wizard();
      frame.Criar();
   }
}
```

APÊNDICE B – CÓDIGO FONTE INTERFACE.JAVA

```
// ================== IMPORTS ====================================
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.*;

import javax.swing.*;

// ********************************* CLASSE INTERFACE ***************************
public class Interface extends JFrame
{
	private JLabel jLabelTipo, jLabelDescricao,jLabelImagem;
	private JButton jButtonNovo, jButtonRemover, jButtonSair, jButtonGravar, jButtonAlterar,
jButtonConsultar, jButtonImagem;
	private JTextField jTextFieldImagem, jTextFieldDescricao, jTextFieldTipo;
	private JComboBox combo1;

	JPanel painelT, painelD, painelI, botao;

	public String tipo[]= {" ", "Órgao" ,"Instrumento Médico"};

	FrameworkDatabaseApplication a = new FrameworkDatabaseApplication ();

// ========================== INTERFACE ====================================
	public Interface()
	{
		super("Manutenção do Banco");

		try
		{
			a.AbrirBanco();

			Container container = getContentPane();
			container.setLayout(new FlowLayout());

			painelT = new JPanel(new FlowLayout());
			painelD = new JPanel(new FlowLayout());
			painelI = new JPanel(new FlowLayout());
			botao = new JPanel(new FlowLayout());

// ****************************** CAMPO TIPO ************************
				jLabelTipo = new JLabel("Tipo");
				combo1 = new JComboBox(tipo);
				painelT.add(jLabelTipo,BorderLayout.WEST);
				painelT.add(combo1,BorderLayout.WEST);

// ****************************** CAMPO DESCRIÇÃO ************************
				jLabelDescricao = new JLabel("Descrição");
				jTextFieldDescricao = new JTextField(50);
				painelD.add(jLabelDescricao,BorderLayout.WEST);
				painelD.add(jTextFieldDescricao);

// ****************************** CAMPO IMAGEM ************************
				jButtonImagem = new JButton("Imagem");
				jButtonImagem.addMouseListener(new MouseAdapter()
				{
					public void mousePressed(MouseEvent evt)
					{
						jButtonImagemMousePressed(evt);
```

```
                    }
                });
        painelI.add(jButtonImagem,BorderLayout.WEST);
                jTextFieldImagem = new JTextField(50);
                painelI.add(jTextFieldImagem);

// ******************************** BOTÃO NOVO ************************
                jButtonNovo = new JButton();
                botao.add(jButtonNovo);
                jButtonNovo.setText("Novo");
                jButtonNovo.addMouseListener(new MouseAdapter()
                {
                        public void mousePressed(MouseEvent evt)
                        {
                                jButtonNovoMousePressed(evt);
                        }
                });
        botao.add(jButtonNovo);

// ******************************** BOTÃO GRAVAR ************************
                jButtonGravar = new JButton();
                botao.add(jButtonGravar);
                jButtonGravar.setText("Gravar");
                jButtonGravar.addMouseListener(new MouseAdapter()
                {
                        public void mousePressed(MouseEvent evt)
                        {
                                jButtonGravarMousePressed(evt);
                        }
                });
                botao.add(jButtonGravar);

// ******************************** BOTÃO REMOVER ************************
                jButtonRemover = new JButton();
                botao.add(jButtonRemover);
                jButtonRemover.setText("Remover");
                jButtonRemover.addMouseListener(new MouseAdapter()
                {
                        public void mousePressed(MouseEvent evt)
                        {
                                jButtonRemoverMousePressed(evt);
                        }
                });
                botao.add(jButtonRemover);

// ******************************** BOTÃO ALTERAR ************************
                jButtonAlterar = new JButton();
                botao.add(jButtonAlterar);
                jButtonAlterar.setText("Alterar");
                jButtonAlterar.addMouseListener(new MouseAdapter()
                {
                        public void mousePressed(MouseEvent evt)
                        {
                                jButtonAlterarMousePressed(evt);
                        }
                });
                botao.add(jButtonAlterar);

// ******************************** BOTÃO CONSULTAR ************************
                jButtonConsultar = new JButton();
```

```
                                botao.add(jButtonConsultar);
                                jButtonConsultar.setText("Consultar");
                                jButtonConsultar.addMouseListener(new MouseAdapter()
                                {
                                        public void mousePressed(MouseEvent evt)
                                        {
                                                jButtonConsultarMousePressed(evt);
                                        }
                                });
                                botao.add(jButtonConsultar);

// ******************************** BOTÃO SAIR ************************
                                jButtonSair = new JButton();
                                botao.add(jButtonSair);
                                jButtonSair.setText("Sair");
                                jButtonSair.addMouseListener(new MouseAdapter()
                                {
                                        public void mousePressed(MouseEvent evt)
                                        {
                                                jButtonSairMousePressed(evt);
                                        }
                                });
                                botao.add(jButtonSair);

                                container.add(painelT);
                                container.add(painelD);
                                container.add(painelI);
                                container.add(botao);

                                setSize(650,200);
                                setLocation(400,250);
                                setVisible(true);

                        } catch (Exception e) {
                                e.printStackTrace();
                        }
                }

// ******************************** MÉTODO NOVO ************************
                private void jButtonNovoMousePressed(MouseEvent evt)
                {
                        combo1.setSelectedIndex(0);
                        jTextFieldDescricao.setText("");
                        jTextFieldImagem.setText("");
                }

// ******************************** MÉTODO GRAVAR ************************
                private void jButtonGravarMousePressed(MouseEvent evt)
                {
                        boolean ok = false;

                        String t = (String)combo1.getSelectedItem();
                    String d = jTextFieldDescricao.getText();
                    String i = jTextFieldImagem.getText();

                        ok = a.InserirDados(t,d,i);
                    if (ok)
                        JOptionPane.showMessageDialog(null, "Gravado no Banco com sucesso!");
                    else
                        JOptionPane.showMessageDialog(null, "Problema na gravação no Banco!");
```

```
        }

// ****************************** MÉTODO REMOVER ************************
        private void jButtonRemoverMousePressed(MouseEvent evt)
        {
                boolean ok = false;

                String d = jTextFieldDescricao.getText();

        ok = a.ExcluirDados(d);
        if (ok)
      JOptionPane.showMessageDialog(null, "Removido com sucesso!");
    else
       JOptionPane.showMessageDialog(null, "Problema na remoção!");
        }

// ****************************** MÉTODO ALTERAR ************************
        private void jButtonAlterarMousePressed(MouseEvent evt)
        {
                boolean ok = false;

                String t = jTextFieldTipo.getText();
                String d = jTextFieldDescricao.getText();
                String i = jTextFieldImagem.getText();

        ok = a.AlterarDados(t,d,i);
        if (ok)
      JOptionPane.showMessageDialog(null, "Alteração com sucesso!");
     else
       JOptionPane.showMessageDialog(null, "Problema na alteração!");
        }

// ****************************** MÉTODO CONSULTAR ************************
        private void jButtonConsultarMousePressed(MouseEvent evt)
        {
                boolean ok = false;

                String d = jTextFieldDescricao.getText();

                ok = a.ConsultarDados(d);

        if (ok == false)
                   JOptionPane.showMessageDialog(null, "Problema na consulta!");
        }

// ****************************** MÉTODO IMAGEM ************************
        private void jButtonImagemMousePressed(MouseEvent evt)
        {
                boolean ok = false;

                JFileChooser chooser = new JFileChooser();
           int returnVal = chooser.showOpenDialog(null);
           String nome_arquivo = "";

                if(returnVal == JFileChooser.APPROVE_OPTION)
                {
              nome_arquivo = chooser.getSelectedFile().getPath();
                        jTextFieldImagem.setText(nome_arquivo);
                        ok = true;
           }
```

```
                if (ok)
      JOptionPane.showMessageDialog(null, "Imagem carregada com sucesso!");
    else
      JOptionPane.showMessageDialog(null, "Problema ao carregar imagem!");
        }

// ****************************** MÉTODO SAIR ************************
        private void jButtonSairMousePressed(MouseEvent evt)
        {
                a.FecharBanco();
                System.exit(0);
        }

// ============================== PRINCIPAL ============================
        public static void main(String[] args)
        {
                Interface application = new Interface();
                application.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        }
}
```

APÊNDICE C – CÓDIGO FONTE FRAMEWORKDATABASEAPPLICATION.JAVA

```
// ================== IMPORTS ======================================
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.sql.SQLException;
import java.sql.Blob;
import java.sql.PreparedStatement;

import java.util.Vector;

import javax.swing.JOptionPane;
import javax.swing.JTextField;

import javax.vecmath.*;

// ************************** CLASSE APLICAÇÃO BANCO FRAMEWORK ******************
public class FrameworkDatabaseApplication
{
  Connection conn = null;
  Statement stm;

  public String framework = "incorporado";
  public String driver = "org.apache.derby.jdbc.EmbeddedDriver";
  public String protocol = "jdbc:derby:";

  private int cod, tipo;
  private String descricao, imagem;

  private Vector orgao = new Vector(30);
  private Vector instrumento_medico = new Vector(30);
  Parameters p = new Parameters();

  private JTextField jTextFieldImagem;
  private JTextField jTextFieldDescricao;
  private JTextField jTextFieldTipo;

  ResultSet result = null;

// ============================== PRINCIPAL ====================================
  public static void main(String[] args)
  {

  }

// ================== MÉTODO ABRIR BANCO ====================================
  public void AbrirBanco()
  {
    try
    {
        System.out.println("Iniciando FrameworkDatabaseApplication em modo " + framework + ".\n");

        Class.forName(driver).newInstance(); //carrega o driver
        System.out.println("Carregando o Driver apropriado.\n");

        conn = DriverManager.getConnection(protocol + "BancoFramework"); //conecta ao banco
```

```
        System.out.println("Conectado ao BancoFramework.\n");

                        stm = conn.createStatement(); //inicializa o statement com a conexao

                        conn.setAutoCommit(true); //commit da conexao true
            }

            catch (Exception e)
            {
       System.out.println("Não foi possível Conectar ao Banco " + e.getMessage() + "\n");
     }
  } //fecha Conexao

// ======================== MÉTODO INSERIR DADOS ==============================
  public boolean InserirDados(String t, String d, String i)
  {
            int Cod = 0;
            int NovoCod = 1;
            try
            {
                  result = stm.executeQuery("select cod_obj from objetos where cod_obj = (select max(cod_obj)
from objetos)");

                  if (result.next())
                  {
                        Cod = result.getInt("cod_obj");
                        NovoCod = Cod + 1;
                  }

                  String query = "insert into objetos values(" + NovoCod + ",'" + t + "','" + d + "','" + i + "')";
             stm.executeUpdate(query);
            }

            catch (SQLException e)
            {
                  System.out.println("Erro na Inserção: " + e.getMessage() + "\n");
            }
            return true;
  } //fecha inserirDados

// ========================== MÉTODO GRAVAR ================================
  //public boolean Gravar(String desc, String arquivo)
  public boolean Gravar(Point3f escala_org, Point3f trans_org, Point3f rot_org, Point3f escala_instr_med,
Point3f trans_instr_med, Point3f rot_instr_med, Point3f forca, float massa, float damp, float cte, float dist_eucl,
String col, String def, String est, String desc)
  {
            boolean ok = true;

            int Cod = 0;
            int obj = 0;
            int NovoCod = 1;

    try
            {
                  System.out.println("Escala_Org_X = " + escala_org.x);
                  System.out.println("Escala_Org_Y = " + escala_org.y);
                  System.out.println("Escala_Org_Z = " + escala_org.z);

                  System.out.println("Trans_Org_X = " + trans_org.x);
                  System.out.println("Trans_Org_Y = " + trans_org.y);
```

```
                System.out.println("Trans_Org_Z = " + trans_org.z);

                System.out.println("Rotacao_Org_X = " + rot_org.x);
                System.out.println("Rotacao_Org_Y = " + rot_org.y);
                System.out.println("Rotacao_Org_Z = " + rot_org.z);
                //System.out.println("Rotacao_Org_ANG = " + rot_org.ang);

                System.out.println("Escala_Instr_Med_X = " + escala_instr_med.x);
                System.out.println("Escala_Instr_Med_Y = " + escala_instr_med.y);
                System.out.println("Escala_Instr_Med_Z = " + escala_instr_med.z);

                System.out.println("Trans_Instr_Med_X = " + trans_instr_med.x);
                System.out.println("Trans_Instr_Med_Y = " + trans_instr_med.y);
                System.out.println("Trans_Instr_Med_Z = " + trans_instr_med.z);

                System.out.println("Rot_Instr_Med_X = " + rot_instr_med.x);
                System.out.println("Rot_Instr_Med_Y = " + rot_instr_med.y);
                System.out.println("Rot_Instr_Med_Z = " + rot_instr_med.z);
                //System.out.println("Rot_Instr_Med_ANG = " + rot_instr_med.ang);

                System.out.println("Forca_X = " + forca.x);
                System.out.println("Forca_Y = " + forca.y);
                System.out.println("Forca_Z = " + forca.z);

                System.out.println("DistEucl = " + dist_eucl);
                System.out.println("mass = " + massa);
                System.out.println("damping = " + damp);
                System.out.println("ConstSpring = " + cte);

                result = stm.executeQuery("select cod_apli from aplicacoes where cod_apli = (select
max(cod_apli) from aplicacoes)");

                if (result.next())
                {
                        Cod = result.getInt("cod_apli");
                        NovoCod = Cod + 1;
                }

                result = stm.executeQuery("select cod_obj from objetos where descricao = " + "'" + desc + "'");

                while (result.next())
                {
                        obj = result.getInt("cod_obj");
                }

                String y = "insert into aplicacoes (cod_apli, cod_orgao, descricao, colisao, deformacao,
estereoscopia, org_escala_x, org_escala_y, org_escala_z, org_translacao_x, org_translacao_y, org_translacao_z,
org_rotacao_x, org_rotacao_y, org_rotacao_z, instr_escala_x, instr_escala_y, instr_escala_z, instr_translacao_x,
instr_translacao_y, instr_translacao_z, instr_rotacao_x, instr_rotacao_y, instr_rotacao_z, def_forca_x,
def_forca_y, def_forca_z, def_massa, def_damping, def_const_mola, col_dist_euclm) values(" + NovoCod + ","
+ obj + ",'" + desc + "','" + col + "','" + def + "','" + est + "'," + escala_org.x + "," + escala_org.y + "," +
escala_org.z + "," + trans_org.x + "," + trans_org.y + "," + trans_org.z + "," + rot_org.x + "," + rot_org.y + "," +
rot_org.z + "," + escala_instr_med.x + "," + escala_instr_med.y + "," + escala_instr_med.z + "," +
trans_instr_med.x + "," + trans_instr_med.y + "," + trans_instr_med.z + "," + rot_instr_med.x + "," +
rot_instr_med.y + "," + rot_instr_med.z + "," + forca.x + "," + forca.y + "," + forca.z + "," + massa + "," + damp
+ "," + cte + "," + dist_eucl + ")";
                stm.executeUpdate(y);

                return ok;
        }
```

```
            catch (SQLException e)
            {
                    System.out.println("Erro na Gravação: " + e.getMessage() + "\n");
                    ok = false;
            }

            return ok;
    } //fecha inserirDados

// ====================== MÉTODO CAMINHO IMAGEM ============================
    public String CaminhoImagem(String descr)
    {
            String Imagem = "";
      try
      {
                    String sql = "select imagem from objetos where descricao = '" + descr + "'";
                    result = stm.executeQuery(sql);

                    while ( result.next() )
                    {
                            Imagem = result.getString("imagem");
                    }
                    return Imagem;
      }

            catch (SQLException e)
            {
                    System.out.println("Erro: " + e.getMessage() + "\n");
                    return Imagem;
            }
    }


// ================== MÉTODO ALTERAR DADOS ==================================
    public boolean AlterarDados(String t, String d, String i)
    {
     boolean testa = false;

            try
            {
        if (t.equals(""))
                    {
                            if(d.equals(""))
                                    testa = false;
                            if(d != "")
                            {
                                    System.out.println ("ALTERANDO DESCRIÇÃO!");
                                    String query = "update objetos set descricao = '" + d +  "' where imagem = '"
+ i + "'";
                                    int linhas = stm.executeUpdate(query);
                                    testa = true;
                            }
                    }

                    if (d.equals(""))
                    {
                            if(t.equals(""))
                                    testa = false;
                            else if(t != "")
```

```
                        {
                                System.out.println ("ALTERANDO TIPO!");
                                String query = "update objetos set tipo = " + t + " where imagem = " + "'" + i
+ "'";
                                int linhas = stm.executeUpdate(query);
                                testa = true;
                        }
                 }

                testa = true;
    }
          catch (SQLException e)
          {
                System.out.println("Erro na Alteração: " + e.getMessage() + "\n");
          }

    return testa;
  }

// ==================== MÉTODO EXCLUIR DADOS ==================================
  public boolean ExcluirDados(String d)
  {
    boolean testa = false;

    try
          {
                 String query = "delete from objetos where descricao = " + "'" + d + "'";
      int linhas = stm.executeUpdate(query);

      if (linhas > 0)
        testa = true;
      else
        testa = false;
    }
          catch (SQLException e)
          {
                System.out.println("Erro na Exclusão: " + e.getMessage() + "\n");
          }

    return testa;
  }

// ==================== MÉTODO VETOR ORGAO ====================================
  public Vector Orgao()
         {
    try
    {
                result = stm.executeQuery("SELECT descricao FROM objetos where tipo = 'Órgao'");

                orgao.add("");
                while ( result.next() )
                {
                        String Descricao = result.getString("descricao");
                        orgao.add(Descricao);
                }
                return orgao;
    }

          catch (SQLException e)
          {
```

```
                System.out.println("Erro no Vetor: " + e.getMessage() + "\n");
            }
            return orgao;
    }

// ================== MÉTODO VETOR INSTRUMENTO MEDICO ========================
    public Vector InstrMed()
            {
      try
      {
                result = stm.executeQuery("SELECT descricao FROM objetos where tipo = 'Instrumento
Médico'");

                instrumento_medico.add("");
                while ( result.next() )
                {
                        String DescricaoI = result.getString("descricao");
                        instrumento_medico.add(DescricaoI);
                }
                return instrumento_medico;
      }

            catch (SQLException e)
            {
                System.out.println("Erro no Vetor: " + e.getMessage() + "\n");
            }
            return instrumento_medico;
    }


// ================== MÉTODO CONSULTAR DADOS ============================
     public boolean ConsultarDados(String d)
            {
            boolean testa = false;
            String str = "";

      try
      {
                result = stm.executeQuery("SELECT cod_obj, tipo, descricao, imagem FROM objetos
WHERE descricao = '" + d + "'");

                while ( result.next() )
                {
                        int Codigo = result.getInt("cod_obj");
                        String Tipo = result.getString("tipo");
                        String Descricao = result.getString("descricao");
          String Imagem = result.getString("imagem");

                        str = "Codigo: " + Codigo + "\n" + "Tipo: " + Tipo + "\n" + "Descrição: " + Descricao
+ "\n" + "Imagem: " + Imagem;

                        JOptionPane.showMessageDialog
                        (
                                null, str, "CONSULTA",
                                JOptionPane.INFORMATION_MESSAGE
                        );
        }
                testa = true;
      }
```

```
            catch (SQLException e)
            {
                    System.out.println("Erro na Consulta: " + e.getMessage() + "\n");
                    testa = false;
            }
            return testa;
    }

// ======================== MÉTODO FECHAR BANCO ==============================
    public void FecharBanco()
    {
      try
            {
                conn.close();
          System.out.println("Base de dados fechada!\n");
      }
      catch (Exception e)
            {
        e.printStackTrace();
        System.out.println("Problema no fechamento da Base de Dados!\n");
      }
    } //fecha FecharBanco
} //fecha classe
// ************************* FIM CLASSE APLICAÇÃO BANCO FRAMEWORK **************
```

APÊNDICE D – CÓDIGO FONTE PARAMETERS.JAVA

```
//Classe referente aos Parametros
import javax.vecmath.*;

public class Parameters
{
		//Parâmetros da fórmula
		public Point3f escala;
		public Point3f translacao;
		public Point3f rotacao;
		public Point3f force;

		public float dist_eucl;
		public float mass;
		public float damping;
		public float const_spring;

		//Construtor da Classe
		public Parameters()
		{
			force = new Point3f();
			escala = new Point3f();
			translacao = new Point3f();
			rotacao = new Point3f();
		}

//================ Set os parametros ========================
		public void setEscalaOrg(float x, float y, float z)
		{
				this.escala.x = x;
				this.escala.y = y;
				this.escala.z = z;
		}
		public void setTranslacaoOrg(float x, float y, float z)
		{
				this.translacao.x = x;
				this.translacao.y = y;
				this.translacao.z = z;
		}
		//public void setRotacaOrg(float x, float y, float z, float ang)
		public void setRotacaoOrg(float x, float y, float z)
		{
				this.rotacao.x = x;
				this.rotacao.y = y;
				this.rotacao.z = z;

				//this.rotacao.ang = ang;
		}
		public void setEscalaInstr(float x, float y, float z)
		{
				this.escala.x = x;
				this.escala.y = y;
				this.escala.z = z;
		}
		public void setTranslacaoInstr(float x, float y, float z)
		{
				this.translacao.x = x;
				this.translacao.y = y;
				this.translacao.z = z;
```

```
		}
		//public void setRotacaInstr(float x, float y, float z, float ang)
		public void setRotacaoInstr(float x, float y, float z)
		{
				this.rotacao.x = x;
				this.rotacao.y = y;
				this.rotacao.z = z;
				//this.translacao.ang = ang;

		}
		public void setForce(float x, float y, float z)
		{
				this.force.x = x;
				this.force.y = y;
				this.force.z = z;
		}
		public void setMass(float mass)
		{
				this.mass = mass;
		}
		public void setDamping(float damping)
		{
				this.damping = damping;
		}
		public void setConstSpring(float const_spring)
		{
				this.const_spring = const_spring;
		}

		public void setDistEucl(float dist_eucl)
		{
				this.dist_eucl = dist_eucl;
		}

//========== Get os parametros ========================
		public Point3f getEscalaOrg()
		{
				return this.escala;
		}
		public Point3f getTranslacaoOrg()
		{
				return translacao;
		}
		public Point3f getRotacaoOrg()
		{
				return rotacao;
		}
		public Point3f getEscalaInstr()
		{
				return escala;
		}
		public Point3f getTranslacaoInstr()
		{
				return translacao;
		}
		public Point3f getRotacaoInstr()
		{
				return rotacao;
		}
		public Point3f getForce()
```

```
		{
			return force;
		}
		public float getMass()
		{
			return mass;
		}
		public float getDamping()
		{
			return damping;
		}
		public float getConstSpring()
		{
			return const_spring;
		}
		public float getDistEucl()
		{
			return dist_eucl;
		}
}
```

APÊNDICE E – CÓDIGO FONTE PARAMORGAN.JAVA

```
//=================== Bibliotecas necessárias ========================
import java.awt.event.*;
import java.awt.*;

import javax.swing.*;

import javax.vecmath.*;

//************************************ CLASSE PARAMETROS ********************************
public class ParamOrgan extends JFrame
{
	private JLabel x, y, z, ang, lb_escala, lb_translacao, lb_rotacao;

	private JTextField jTextFieldAng;
	private JTextField jTextFieldEscalax, jTextFieldTranslacaox, jTextFieldRotacaox;
	private JTextField jTextFieldEscalay, jTextFieldTranslacaoy, jTextFieldRotacaoy;
	private JTextField jTextFieldEscalaz, jTextFieldTranslacaoz, jTextFieldRotacaoz;

	private JButton jButtonSalvar, jButtonSalvarO, jButtonSalvarI, jButtonCancelar;

	public String oex, oey, oez, otx, oty, otz, orx, ory, orz, orang;
	public float foex, foey, foez, fotx, foty, fotz, forx, fory, forz, forang;

	JFrame window = new JFrame();
	private Container container;
	private GridBagLayout gblayout = new GridBagLayout();
	private GridBagConstraints constraints = new GridBagConstraints();

	Parameters p = new Parameters();

// ================================== PRINCIPAL ================================
  public static void main(String[] args)
  {

  }

//============ METODO QUE CRIA PARAMETROS DO ORGAO ====================
	public void ParamOrgan(Parameters p)
	{
		boolean ok = true;
		try
		{
			window.setTitle("Parametros do Orgao");
	      window.setSize(480, 200);
			window.setLocation(100,0);

			container = window.getContentPane();
			container.setLayout(gblayout);

			this.p = p;

			x = new JLabel("  X ");
			addComponent(x,0,1,1,1);

			y = new JLabel("  Y ");
			addComponent(y,0,2,1,1);

			z = new JLabel("  Z ");
```

```
addComponent(z,0,3,1,1);

ang = new JLabel("  Angulo ");
addComponent(ang,0,4,1,1);

jTextFieldAng = new JTextField(7);
jTextFieldAng.setText("0.0");
addComponent(jTextFieldAng,3,4,1,1);

//========== Escala ============
lb_escala = new JLabel("  Escala ");
addComponent(lb_escala,1,0,1,1);

jTextFieldEscalax = new JTextField(7);
jTextFieldEscalax.setText("0.5");
addComponent(jTextFieldEscalax,1,1,1,1);

jTextFieldEscalay = new JTextField(7);
jTextFieldEscalay.setText("0.5");
addComponent(jTextFieldEscalay,1,2,1,1);

jTextFieldEscalaz = new JTextField(7);
jTextFieldEscalaz.setText("0.5");
addComponent(jTextFieldEscalaz,1,3,1,1);

//============== Translacao ================
lb_translacao = new JLabel("  Translacao ");
addComponent(lb_translacao,2,0,1,1);

jTextFieldTranslacaox = new JTextField(7);
jTextFieldTranslacaox.setText("0.0");
addComponent(jTextFieldTranslacaox,2,1,1,1);

jTextFieldTranslacaoy = new JTextField(7);
jTextFieldTranslacaoy.setText("0.0");
addComponent(jTextFieldTranslacaoy,2,2,1,1);

jTextFieldTranslacaoz = new JTextField(7);
jTextFieldTranslacaoz.setText("0.5");
addComponent(jTextFieldTranslacaoz,2,3,1,1);

//============== Rotacao ==================
lb_rotacao = new JLabel("  Rotacao ");
addComponent(lb_rotacao,3,0,1,1);

jTextFieldRotacaox = new JTextField(7);
jTextFieldRotacaox.setText("0.0");
addComponent(jTextFieldRotacaox,3,1,1,1);

jTextFieldRotacaoy = new JTextField(7);
jTextFieldRotacaoy.setText("0.0");
addComponent(jTextFieldRotacaoy,3,2,1,1);

jTextFieldRotacaoz = new JTextField(7);
jTextFieldRotacaoz.setText("0.0");
addComponent(jTextFieldRotacaoz,3,3,1,1);

//============== Botoes ==================
jButtonSalvarO = new JButton("   Salvar   ");
constraints.weighty = 0.1;
```

```
                    addComponent(jButtonSalvarO,5,1,1,1);
                    jButtonSalvarO.addActionListener(new ActionListener()
                    {
                        public void actionPerformed(ActionEvent evt)
                        {
                            boolean ok = true;
                            ok = SalvarParamOrg();
                            return;
                        }

    });

                    jButtonCancelar = new JButton("Cancelar");
                    addComponent(jButtonCancelar,5,3,1,1);
                    jButtonCancelar.addActionListener(new ActionListener()
                    {
                        public void actionPerformed(ActionEvent evt)
                        {
                            //System.exit(0);
        }
    });

                    window.setVisible(true);
                    //return ok;
        }

        catch (Exception ee)
        {
            ee.printStackTrace();
            //ok = false;
        }
        //return ok;
    }

//============= METODO SALVAR PARAMETROS DO ORGAO ==========================
    private boolean SalvarParamOrg()
    {
        boolean ok = true;

        try
        {
            System.out.println ("\nSalvando Parametros do Orgao");

            oex = jTextFieldEscalax.getText();
            oey = jTextFieldEscalay.getText();
            oez = jTextFieldEscalaz.getText();

            otx = jTextFieldTranslacaox.getText();
            oty = jTextFieldTranslacaoy.getText();
            otz = jTextFieldTranslacaoz.getText();

            orx = jTextFieldRotacaox.getText();
            ory = jTextFieldRotacaoy.getText();
            orz = jTextFieldRotacaoz.getText();

            orx = jTextFieldRotacaox.getText();
            ory = jTextFieldRotacaoy.getText();
            orz = jTextFieldRotacaoz.getText();
            //orang = jTextFieldAng.getText();
```

```
                        System.out.println("oex: " + oex + " oey: " + oey + " oez: " + oez);
                        System.out.println("otx: " + otx + " oty: " + oty + " otz: " + otz);
                        System.out.println("orx: " + orx + " ory: " + ory + " orz: " + orz);
                        //System.out.println("orx: " + orx + " ory: " + ory + " orz: " + orz + " orang: " +
orang);

                        foex = Float.parseFloat(oex);
                        foey = Float.parseFloat(oey);
                        foez = Float.parseFloat(oez);

                        fotx = Float.parseFloat(otx);
                        foty = Float.parseFloat(oty);
                        fotz = Float.parseFloat(otz);

                        forx = Float.parseFloat(orx);
                        fory = Float.parseFloat(ory);
                        forz = Float.parseFloat(orz);
                        //forang = Float.parseFloat(orang);

                        p.setEscalaOrg(foex,foey,foez);
                        p.setTranslacaoOrg(fotx,foty,fotz);
                        p.setRotacaoOrg(forx,fory,forz);
                        //p.setRotacaoOrg(forx,fory,forz,forang);
                }
                catch (Exception ee)
                {
                        ee.printStackTrace();
                        ok = false;
                }
                return ok;
        }

//======================== METODO ADDCOMPONENT ==========================
        private void addComponent(Component component, int row, int column, int width, int height)
        {
                constraints.gridx = column;
                constraints.gridy = row;

                constraints.gridwidth = width;
                constraints.gridheight = height;

                gblayout.setConstraints(component, constraints);
                container.add(component);
        }
}
```

APÊNDICE F – CÓDIGO FONTE PARAMMEDINSTR.JAVA

```
//=================== Bibliotecas necessárias ========================
import java.awt.event.*;
import java.awt.*;

import javax.swing.*;

//************************************ CLASSE PARAMETROS ********************************
public class ParamMedInstr extends JFrame
{
	private JLabel x, y, z, ang, lb_escala, lb_translacao, lb_rotacao;

	private JTextField jTextFieldAng;
	private JTextField jTextFieldEscalax, jTextFieldTranslacaox, jTextFieldRotacaox;
	private JTextField jTextFieldEscalay, jTextFieldTranslacaoy, jTextFieldRotacaoy;
	private JTextField jTextFieldEscalaz, jTextFieldTranslacaoz, jTextFieldRotacaoz;

	private JButton jButtonSalvar, jButtonCancelar;

	public String imex, imey, imez, imtx, imty, imtz, imrx, imry, imrz, imrang;
	public float fimex, fimey, fimez, fimtx, fimty, fimtz, fimrx, fimry, fimrz, fimrang;

	JFrame window = new JFrame();
	private Container container;
	private GridBagLayout gblayout = new GridBagLayout();
	private GridBagConstraints constraints = new GridBagConstraints();

	Parameters p = new Parameters();

// ============================== PRINCIPAL ===================================
  public static void main(String[] args)
  {

  }

//============ METODO QUE CRIA PARAMETROS DO INSTRUMENTO MEDICO =============
	public void ParamMedInstr(Parameters p)
	{
		boolean ok = true;
		try
		{
			window.setTitle("Parametros do Instrumento Medico");
	      window.setSize(480, 200);
			window.setLocation(100,0);

			container = window.getContentPane();
			container.setLayout(gblayout);

			this.p = p;

			x = new JLabel("  X ");
			addComponent(x,0,1,1,1);

			y = new JLabel("  Y ");
			addComponent(y,0,2,1,1);

			z = new JLabel("  Z ");
			addComponent(z,0,3,1,1);
```

```
ang = new JLabel("  Angulo ");
addComponent(ang,0,4,1,1);

jTextFieldAng = new JTextField(7);
jTextFieldAng.setText("0.0");
addComponent(jTextFieldAng,3,4,1,1);

//========== Escala ============
lb_escala = new JLabel("  Escala ");
addComponent(lb_escala,1,0,1,1);

jTextFieldEscalax = new JTextField(7);
jTextFieldEscalax.setText("0.5");
addComponent(jTextFieldEscalax,1,1,1,1);

jTextFieldEscalay = new JTextField(7);
jTextFieldEscalay.setText("0.5");
addComponent(jTextFieldEscalay,1,2,1,1);

jTextFieldEscalaz = new JTextField(7);
jTextFieldEscalaz.setText("0.5");
addComponent(jTextFieldEscalaz,1,3,1,1);

//============== Translacao ================
lb_translacao = new JLabel("  Translacao ");
addComponent(lb_translacao,2,0,1,1);

jTextFieldTranslacaox = new JTextField(7);
jTextFieldTranslacaox.setText("0.0");
addComponent(jTextFieldTranslacaox,2,1,1,1);

jTextFieldTranslacaoy = new JTextField(7);
jTextFieldTranslacaoy.setText("0.0");
addComponent(jTextFieldTranslacaoy,2,2,1,1);

jTextFieldTranslacaoz = new JTextField(7);
jTextFieldTranslacaoz.setText("0.5");
addComponent(jTextFieldTranslacaoz,2,3,1,1);

//============== Rotacao ==================
lb_rotacao = new JLabel("  Rotacao ");
addComponent(lb_rotacao,3,0,1,1);

jTextFieldRotacaox = new JTextField(7);
jTextFieldRotacaox.setText("0.0");
addComponent(jTextFieldRotacaox,3,1,1,1);

jTextFieldRotacaoy = new JTextField(7);
jTextFieldRotacaoy.setText("0.0");
addComponent(jTextFieldRotacaoy,3,2,1,1);

jTextFieldRotacaoz = new JTextField(7);
jTextFieldRotacaoz.setText("0.0");
addComponent(jTextFieldRotacaoz,3,3,1,1);

//============== Botoes ==================
jButtonSalvar = new JButton("   Salvar   ");
constraints.weighty = 0.1;
addComponent(jButtonSalvar,5,1,1,1);
jButtonSalvar.addActionListener(new ActionListener()
```

```
                {
                        public void actionPerformed(ActionEvent evt)
                        {
                                boolean ok = true;
                                ok = SalvarParamInstrMed();
                                return;
                        }

        });

                        jButtonCancelar = new JButton("Cancelar");
                        addComponent(jButtonCancelar,5,3,1,1);
                        jButtonCancelar.addActionListener(new ActionListener()
                        {
                                public void actionPerformed(ActionEvent evt)
                                {
                                        //System.exit(0);
                        }
        });

                                window.setVisible(true);
                                //return ok;
                        }

                        catch (Exception ee)
                        {
                                ee.printStackTrace();
                                //ok = false;
                        }
                        //return ok;
        }

//============== METODO SALVAR PARAMETROS DO INTRUMENTO MEDICO =============
        private boolean SalvarParamInstrMed()
        {
                boolean ok = true;

                try
                {
                        System.out.println ("\nSalvando Parametros do Instrumento Medico");

                        imex = jTextFieldEscalax.getText();
                        imey = jTextFieldEscalay.getText();
                        imez = jTextFieldEscalaz.getText();

                        imtx = jTextFieldTranslacaox.getText();
                        imty = jTextFieldTranslacaoy.getText();
                        imtz = jTextFieldTranslacaoz.getText();

                        imrx = jTextFieldRotacaox.getText();
                        imry = jTextFieldRotacaoy.getText();
                        imrz = jTextFieldRotacaoz.getText();
                        //imrang = jTextFieldAng.getText();

                        System.out.println("imex: " + imex + " imey: " + imey + " imez: " + imez);
                        System.out.println("imtx: " + imtx + " imty: " + imty + " imtz: " + imtz);
                        System.out.println("imrx: " + imrx + " imry: " + imry + " imrz: " + imrz);
                        //System.out.println("imrx: " + imrx + " imry: " + imry + " imrz: " + imrz + " imrang:
" + imrang);
```

```
                        fimex = Float.parseFloat(imex);
                        fimey = Float.parseFloat(imey);
                        fimez = Float.parseFloat(imez);

                        fimtx = Float.parseFloat(imtx);
                        fimty = Float.parseFloat(imty);
                        fimtz = Float.parseFloat(imtz);

                        fimrx = Float.parseFloat(imrx);
                        fimry = Float.parseFloat(imry);
                        fimrz = Float.parseFloat(imrz);
                        //fimrang = Float.parseFloat(imrang);

                        p.setEscalaInstr(fimex,fimey,fimez);
                        p.setTranslacaoInstr(fimtx,fimty,fimtz);
                        p.setRotacaoInstr(fimrx,fimry,fimrz);
                        //p.setRotacaoInstr(fimrx,fimry,fimrz,fimrang);
                }
                catch (Exception ee)
                {
                        ee.printStackTrace();
                        ok = false;
                }
                return ok;
        }

//======================== METODO ADDCOMPONENT ========================
        private void addComponent(Component component, int row, int column, int width, int height)
        {
                constraints.gridx = column;
                constraints.gridy = row;

                constraints.gridwidth = width;
                constraints.gridheight = height;

                gblayout.setConstraints(component, constraints);
                container.add(component);
        }
}
```

APÊNDICE G – CÓDIGO FONTE PARAMCOLLISION.JAVA

```
//=================== Bibliotecas necessárias ==========================
import java.awt.event.*;
import java.awt.*;

import javax.swing.*;

//************************************ CLASSE PARAMETROS ********************************
public class ParamCollision extends JFrame
{
	private JLabel dist_eucl;

	private JTextField jTextFieldDistEucl;

	private JButton jButtonSalvar, jButtonCancelar;

	public String cde;

	public float fcde;

	JFrame window = new JFrame();
	private Container container;
	private GridBagLayout gblayout = new GridBagLayout();
	private GridBagConstraints constraints = new GridBagConstraints();

	Parameters p = new Parameters();

// ============================ PRINCIPAL ====================================
  public static void main(String[] args)
  {

  }

//=============== METODO CRIAR PARAMETROS COLISAO ========================
	public boolean ParamCollision(Parameters p)
	{
		boolean ok = true;
		try
		{
			window.setTitle("Parametros Colisao");
		window.setSize(400, 150);
			window.setLocation(100,0);

			container = window.getContentPane();
			container.setLayout(gblayout);

			this.p = p;

			dist_eucl = new JLabel("   Distancia Euclidiana   ");
			addComponent(dist_eucl,0,0,1,1);

			jTextFieldDistEucl = new JTextField(7);
			jTextFieldDistEucl.setText("0.001");
			addComponent(jTextFieldDistEucl,0,1,1,1);

			jButtonSalvar = new JButton("   Salvar   ");
			constraints.weighty = 0.1;
			addComponent(jButtonSalvar,1,0,1,1);
			jButtonSalvar.addActionListener(new ActionListener()
```

```
                {
                        public void actionPerformed(ActionEvent evt)
                        {
                                boolean ok = true;
                                ok = SalvarParamCol();
                                return;
                }
        });

                        jButtonCancelar = new JButton("Cancelar");
                        addComponent(jButtonCancelar,1,1,1,1);
                        jButtonCancelar.addActionListener(new ActionListener()
                        {
                                public void actionPerformed(ActionEvent evt)
                                {
                                        //System.exit(0);
                }
        });

                        window.setVisible(true);
                        return ok;
                }

                catch (Exception ee)
                {
                        ee.printStackTrace();
                        ok = false;
                }
                return ok;
        }

//======================== METODO ADDCOMPONENT ========================
        private void addComponent(Component component, int row, int column, int width, int height)
        {
                constraints.gridx = column;
                constraints.gridy = row;

                constraints.gridwidth = width;
                constraints.gridheight = height;

                gblayout.setConstraints(component, constraints);
                container.add(component);
        }

//=============== METODO SALVAR PARAMETROS DA COLISAO ======================
        private boolean SalvarParamCol()
        {
                boolean ok = true;

                try
                {
                        System.out.println ("\nSalvando Parametros de Colisao");

                        cde = jTextFieldDistEucl.getText();
                        System.out.println("cde: " + cde);

                        fcde = Float.parseFloat(cde);
                        p.setDistEucl(fcde);
                }
                catch (Exception ee)
```

```
                {
                        ee.printStackTrace();
                        ok = false;
                }
                return ok;
        }
}
```

APÊNDICE H – CÓDIGO FONTE PARAMDEFORMATION.JAVA

```
//=================== Bibliotecas necessárias ==========================
import java.awt.event.*;
import java.awt.*;

import javax.swing.*;

//************************************ CLASSE PARAMETROS ********************************
public class ParamDeformation extends JFrame
{
	private JLabel forca, massa, damping, cte_mola;

	private JTextField jTextFieldForca_x, jTextFieldForca_y, jTextFieldForca_z;
	private JTextField jTextFieldMassa, jTextFieldDamping, jTextFieldConst_Mola;

	private JButton jButtonSalvar, jButtonCancelar;

	public String dfx, dfy, dfz, dm, dd, dcm;

	public float fdfx, fdfy, fdfz, fdm, fdd, fdcm;

	JFrame window = new JFrame();
	private Container container;
	private GridBagLayout gblayout = new GridBagLayout();
	private GridBagConstraints constraints = new GridBagConstraints();

	Parameters p = new Parameters();

// ============================== PRINCIPAL ========================================
  public static void main(String[] args)
  {

  }

//================ METODO CRIAR PARAMETROS DEFORMACAO ====================
	public void ParamDeformation(Parameters p)
	{
		boolean ok = true;
		try
		{
			window.setTitle("Parametros Deformacao");
	      window.setSize(450, 200);
			window.setLocation(100,0);

			container = window.getContentPane();
			container.setLayout(gblayout);

			this.p = p;

			forca = new JLabel(" Forca (x, y, z) ");
			addComponent(forca,0,0,1,1);

			jTextFieldForca_x = new JTextField(7);
			jTextFieldForca_x.setText("3.0");
			addComponent(jTextFieldForca_x,0,1,1,1);

			jTextFieldForca_y = new JTextField(7);
			jTextFieldForca_y.setText("0.0");
			addComponent(jTextFieldForca_y,0,2,1,1);
```

```
                jTextFieldForca_z = new JTextField(7);
                jTextFieldForca_z.setText("0.0");
                addComponent(jTextFieldForca_z,0,3,1,1);

                massa = new JLabel("Massa");
                addComponent(massa,1,0,1,1);

                jTextFieldMassa = new JTextField(7);
                jTextFieldMassa.setText("300.0");
                addComponent(jTextFieldMassa,1,1,1,1);

                damping = new JLabel("Damping");
                addComponent(damping,2,0,1,1);

                jTextFieldDamping = new JTextField(7);
                jTextFieldDamping.setText("0.7");
                addComponent(jTextFieldDamping,2,1,1,1);

                cte_mola = new JLabel("  Constante da Mola  ");
                addComponent(cte_mola,3,0,1,1);

                jTextFieldConst_Mola = new JTextField(7);
                jTextFieldConst_Mola.setText("0.3");
                addComponent(jTextFieldConst_Mola,3,1,1,1);

                jButtonSalvar = new JButton("   Salvar   ");
                constraints.weighty = 0.1;
                addComponent(jButtonSalvar,4,0,1,1);
                jButtonSalvar.addActionListener(new ActionListener()
                {
                        public void actionPerformed(ActionEvent evt)
                        {
                                boolean ok = true;
                                ok = SalvarParamDef();
                                //return;
        }
});

                jButtonCancelar = new JButton("Cancelar");
                addComponent(jButtonCancelar,4,2,1,1);
                jButtonCancelar.addActionListener(new ActionListener()
                {
                        public void actionPerformed(ActionEvent evt)
                        {
                                //System.exit(0);
        }
});

                window.setVisible(true);
                //return ok;

        }
        catch (Exception ee)
        {
                ee.printStackTrace();
                ok = false;
        }
        //return ok;
}
```

```
//======================== METODO ADDCOMPONENT ============================
	private void addComponent(Component component, int row, int column, int width, int height)
	{
		constraints.gridx = column;
		constraints.gridy = row;

		constraints.gridwidth = width;
		constraints.gridheight = height;

		gblayout.setConstraints(component, constraints);
		container.add(component);
	}

//=================== METODO SALVAR PARAMETROS DO ORGAO ==================
	private boolean SalvarParamDef()
	{
		boolean ok = true;

		try
		{
			System.out.println ("\nSalvando Parametros da Deformacao");

			dfx = jTextFieldForca_x.getText();
			dfy = jTextFieldForca_y.getText();
			dfz = jTextFieldForca_z.getText();
			dm = jTextFieldMassa.getText();
			dd = jTextFieldDamping.getText();
			dcm = jTextFieldConst_Mola.getText();

			System.out.println("dfx: " + dfx + "\ndfy: " + dfy + "\ndfz: " + dfz + "\ndm: " + dm +
"\ndd: " + dd + "\ndcm: " + dcm);

			fdfx = Float.parseFloat(dfx);
			fdfy = Float.parseFloat(dfy);
			fdfz = Float.parseFloat(dfz);
			fdm = Float.parseFloat(dm);
			fdd = Float.parseFloat(dd);
			fdcm = Float.parseFloat(dcm);

			p.setForce(fdfx,fdfy,fdfz);
			p.setMass(fdm);
			p.setDamping(fdd);
			p.setConstSpring(fdcm);
		}
		catch (Exception ee)
		{
			ee.printStackTrace();
			ok = false;
		}
		return ok;
	}
}
```

APÊNDICE I – CÓDIGO FONTE BUILDCODE.JAVA

```
import java.io.*;

//************************************ CLASSE GERAR CODIGO ******************************
public class BuilCode
{
	public static void main(String[] args)
	{

   }

//========================== METODO GERAR ==============================
	public boolean Build()
	{
		boolean ok = true;
		try
		{
			// Lendo do arquivo .java
	File arq = new File("Application.java");
		FileInputStream entrada = new FileInputStream(arq);

			File file = new File("ApplicationGerado.java");
			FileOutputStream saida = new FileOutputStream(file);

	int linha;
	while ( (linha = entrada.read()) != -1 )
			{
				saida.write((char) linha);
	}
	entrada.close();

	saida.close();
			return ok;
		}
		catch (Exception ee)
		{
	ee.printStackTrace();
			ok = false;
	}
		return ok;
	} //fecha metodo Gerar
} //fecha classe GerarCodigo
```

APÊNDICE J – CÓDIGO GERADO DA APLICAÇÃO DO PRIMEIRO ESTUDO DE CASO

```
import com.sun.j3d.loaders.objectfile.ObjectFile;

import java.awt.Container;
import java.util.Vector;

import javax.swing.JFrame;
import javax.media.j3d.*;
import javax.vecmath.*;

//***************** CLASSE APPLICATION EXTENDS ENVIRONMENT *************************
public class Application extends Environment{

	Object3D objetos[]; // - Lista de objetos no universo

   Octree cd; // - detecçao de colisão
	MassSpring def; // - deformaçao

	public Application(Canvas3D c)
	{
		super(c);

		Parameters p = new Parameters();

		objetos = new Object3D[2];

// ----------------------- Instanciaçao do objeto deformável -----------------------------
		objetos[0] = new ObjDef("mama.obj",ObjDef.COLLISION +
ObjDef.DEFORMATION,ObjectFile.RESIZE);

		def = ((ObjDef)objetos[0]).getDeformation();

		// --- Parametros da deformação (Força, Massa, Damping e Constante da mola) ---
		p.setForce(3.0f,0f,0f);
		p.setMass(300f);
		p.setDamping(0.7f);
		p.setConstSpring(0.3f);

		// passa esses parametros para o objeto def.
		def.setParameters(p);

		cd = ((ObjDef)objetos[0]).getCollisionDetector(); //adiciona o método de colisão no objeto de
colisão
		cd.setPrecision(0.000001); //parametros da distância Euclidiana
		cd.setCollisionListener(def); //tratamento do evento

// ---------------------- Instanciaçao do objeto rígido --------------------------
		objetos[1] = new ObjRig("seringa1.obj",ObjectFile.RESIZE);

		// --- Parametros de Escala, Transformação e Rotação ---
		objetos[1].setScale(new Vector3d (0.4f, 0.4f,0.4f));
	objetos[1].setTranslation(new Vector3d (0.5f, 0.0f, 0.90f));
	    objetos[1].setRotation(new AxisAngle4d (1.0f, 0.0f, 0.0f, 1.57f));

		//Adiçao dos objetos no universo
		this.add(objetos[0]);
		this.add(objetos[1]);
```

```
	}

	public static void main (String arg[])
	{
		Canvas3D c = new Canvas3D(null);

		System.out.println("stereo is available: " + c.getStereoAvailable());
		System.out.println("stereo is enabled: " + c.getStereoEnable());

		Application n = new Application(c);
		JFrame frm = new JFrame();

		Container ct = frm.getContentPane();
		ct.add(c);

		frm.setSize(600,400);
		frm.setVisible(true);
		frm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
	}
}
```

APÊNDICE K – CÓDIGO GERADO DA APLICAÇÃO DO SEGUNDO ESTUDO DE CASO

```
import com.sun.j3d.loaders.objectfile.ObjectFile;

import java.awt.Container;
import java.util.Vector;

import javax.swing.JFrame;
import javax.media.j3d.*;
import javax.vecmath.*;

//***************** CLASSE APPLICATION EXTENDS ENVIRONMENT *************************
public class Application extends Environment{

	Object3D objetos[]; // - Lista de objetos no universo

   Octree cd; // - detecçao de colisão
	MassSpring def; // - deformaçao

	public Application(Canvas3D c)
	{
		super(c);

		Parameters p = new Parameters();

		objetos = new Object3D[2];

// ----------------------- Instanciaçao do objeto deformável -----------------------------
		objetos[0] = new ObjDef("glúteo.obj",ObjDef.COLLISION +
ObjDef.DEFORMATION,ObjectFile.RESIZE);

		def = ((ObjDef)objetos[0]).getDeformation();

		// --- Parametros da deformação (Força, Massa, Damping e Constante da mola) ---
		p.setForce(3.0f,0f,0f);
		p.setMass(300f);
		p.setDamping(0.7f);
		p.setConstSpring(0.3f);

		// passa esses parametros para o objeto def.
		def.setParameters(p);

		cd = ((ObjDef)objetos[0]).getCollisionDetector(); //adiciona o método de colisão no objeto de
colisão
		cd.setPrecision(0.000001); //parametros da distância Euclidiana
		cd.setCollisionListener(def); //tratamento do evento

// ---------------------- Instanciaçao do objeto rígido --------------------------
		objetos[1] = new ObjRig("agulha_nova.obj",ObjectFile.RESIZE);

		// --- Parametros de Escala, Transformação e Rotação ---
		objetos[1].setScale(new Vector3d (0.4f, 0.4f,0.4f));
	objetos[1].setTranslation(new Vector3d (0.5f, 0.0f, 0.90f));
	    objetos[1].setRotation(new AxisAngle4d (1.0f, 0.0f, 0.0f, 1.57f));

		//Adiçao dos objetos no universo
		this.add(objetos[0]);
		this.add(objetos[1]);
```

```
	}

	public static void main (String arg[])
	{
		Canvas3D c = new Canvas3D(null);

		System.out.println("stereo is available: " + c.getStereoAvailable());
		System.out.println("stereo is enabled: " + c.getStereoEnable());

		Application n = new Application(c);
		JFrame frm = new JFrame();

		Container ct = frm.getContentPane();
		ct.add(c);

		frm.setSize(600,400);
		frm.setVisible(true);
		frm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
	}
}
```

ANEXO A – INSTALAÇÃO DO SGBD DERBY

Para poder instalar o Derby, primeiro é necessário fazer o *download* do arquivo com extensão zip ou tar do site do Derby, disponível em http://db.apache.org/derby/derby_downloads.html. A página contém várias distribuições do Derby, incluindo pacotes instantâneos e distribuições binárias de versões estáveis. As distribuições se encontram em pacotes *bin*, *lib* e *src*. Após fazer o *download* e extrair o pacote de distribuição *bin*, que conterá os seguintes subdiretórios:

1 – *demo*: contém programas de demonstração.

2 – *frameworks*: contém *scripts* para a execução de utilitários e configuração do ambiente.

3 – *javadoc*: contém a documentação da *API* gerada a partir dos comentários do código fonte.

4 – *doc*: contém a documentação do Derby.

5 – *lib*: contém os arquivos com extensão .jar do Derby.

Configuração do ambiente Java

É necessário configurar a variável de ambiente *path* para que a JVM e os aplicativos Java executem corretamente, porque permite que o sistema operacional encontre os programas apropriados a partir de qualquer diretório.

Se houver mais de uma JVM instalada, a JVM que se deseja utilizar deve estar antes de qualquer outra na variável *path*. Para verificar a variável de ambiente *path*, basta digitar em uma janela de comando:

java -version.

Se o caminho estiver definido corretamente, será exibida uma informação indicando a versão da JVM. Se o comando não retornar a versão correta da JVM, a variável *path* deve

ser configurada adicionando o subdiretório bin do diretório da JVM ao começo do caminho. Por exemplo, se o diretório for *C:\JDK1.5*, deve ser adicionado *C:\JDK1.5\bin* ao começo do caminho.

Utilização das ferramentas e dos utilitários de inicialização

As ferramentas do Derby incluem o *dblook*, o *ij* e o *sysinfo*. O diretório */frameworks/embedded/bin* contém *scripts* para executar algumas das ferramentas e utilitários do Derby no modo incorporado. Podem ser encontrados *script*s semelhantes para executar ferramentas e utilitários do *Network Server* no diretório */frameworks/NetworkServer/bin*. Nesta monografia, o diretório */frameworks/embedded/bin* é referenciado como diretório */bin*, a menos que indicado de outra forma.

Os *scripts* possuem nomes descritivos, como *sysinfo.bat* ou *ij.ksh*. Estes *script*s terminam com extensões diferentes dependendo do ambiente. Os *scripts* para o Windows possuem extensão *.bat*, enquanto os *scripts* para o Unix possuem extensão *.ksh*. Pode ser necessário modificar estes *scripts* para que as ferramentas e utilitários executem de forma apropriada nas plataformas Windows e Unix.

A ferramenta *sysinfo* do Derby mostra informações sobre o ambiente Java e a versão do Derby. O *script* define as variáveis de ambiente apropriadas, incluindo o caminho de classe, e executa o programa *sysinfo*. Uma vez que se tenha o diretório */bin* na variável *path*, o *sysinfo* pode ser executado digitando-se em uma janela de comando:

sysinfo

A ferramenta *ij* pode ser utilizada para conectar a um BD Derby. O diretório */bin* deve estar incluído na variável de ambiente *path* para executar o *ij*. Pode ser executado digitando o seguinte comando:

ij

O *script ij* executa o programa e define variável de ambiente, como o *classpath*. Para se criar um BD utilizando o *ij*, deve ser digitado o seguinte comando:

ij> connect 'jdbc:derby:bdteste;create=true';

Este comando cria o BD chamado *bdteste* no diretório atual, preenche as tabelas do sistema, e faz a conexão ao BD. Em seguida podem ser executadas quaisquer instruções SQL a partir da linha de comandos do *ij*. Para finalizar a execução do aplicativo, basta digitar:

ij> exit;

Configuração manual das variáveis de ambiente e caminhos

Se não for possível executar os *scripts* das ferramentas e utilitários do Derby, determinadas etapas deverão ser concluídas manualmente.

Durante a instalação é criado um diretório base. Assume-se que este diretório se chama *Derby_10*. Caso se planeje utilizar os *script*s presentes no diretório */bin*, e se o sistema operacional suportar, deve ser criada uma variável de ambiente chamada DERBY_HOME e definido seu valor com o caminho para o diretório base do Derby. Isto permite a utilização de comandos curtos para executar as ferramentas e utilitários do Derby.

Por exemplo, se o BD está instalado em *C:\Derby_10*, a variável de ambiente DERBY_HOME deve ser definida com o valor *C:\Derby_10*, demonstrado na Figura 30:

A JVM precisa conhecer o caminho de todos os arquivos de classe necessários ao aplicativo. O caminho de classe é uma lista de bibliotecas de classe requeridas pela JVM e outros aplicativos Java para executar o programa.

A variável de ambiente *CLASSPATH* pode ser definida no sistema operacional de forma temporária, permanente, ou em tempo de execução quando for iniciado o aplicativo Java e a JVM. Se a variável de ambiente for definida temporariamente, esta deve ser definida toda vez que é aberta uma nova janela de comandos.

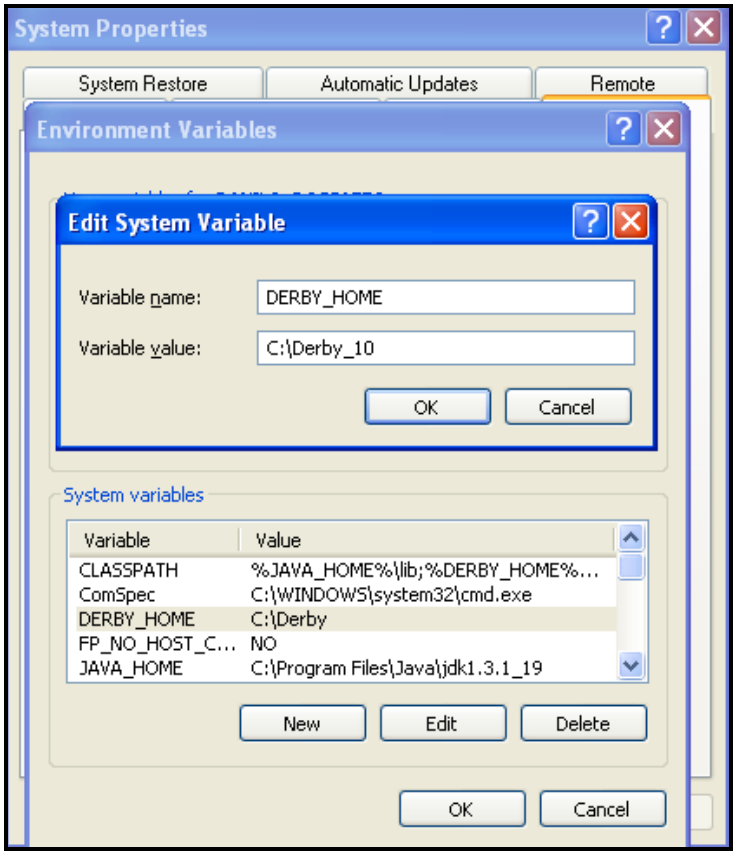
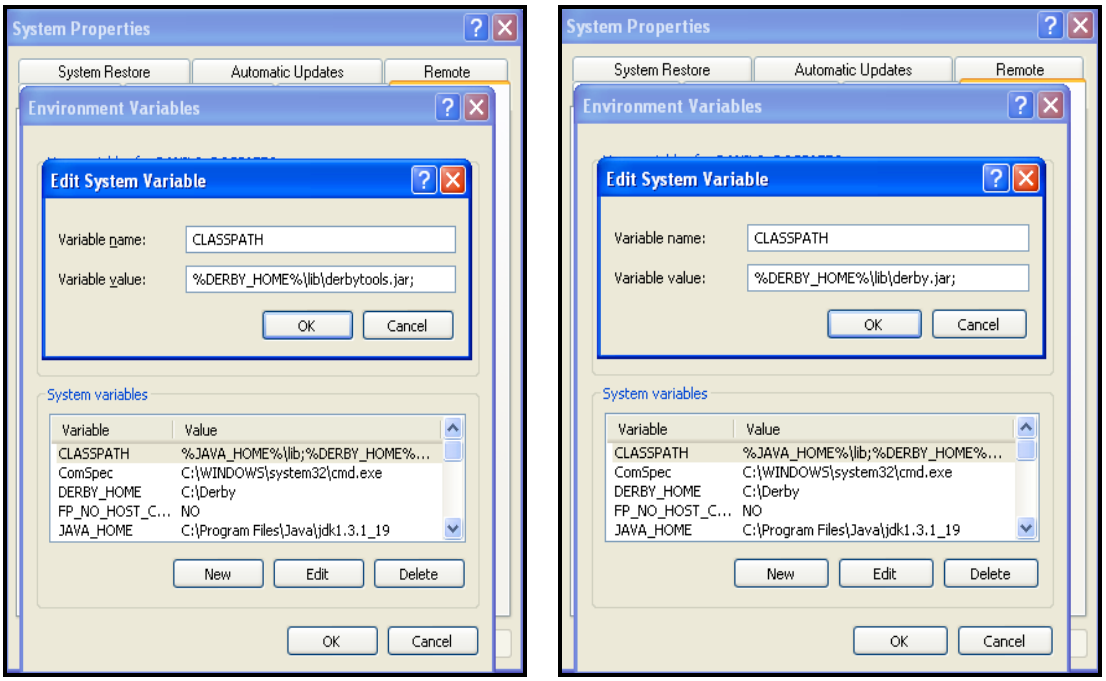


Figura 30 – Configuração da variavel DERBY_HOME

Na maioria dos ambientes de desenvolvimento, é melhor definir a variável de ambiente *CLASSPATH* do sistema operacional temporariamente. O Derby fornece alguns *scripts* para ajudar a definir o caminho de classe desta maneira; estão localizados no diretório */bin* e no diretório */frameworks/NetworkServer/bin*. Para definir o caminho de classe temporariamente, deve ser executado um *script* toda vez que for aberta uma nova janela de comandos.

Para definir a variável *CLASSPATH* de modo permanente, é necessário incluir os arquivos *derby.jar* conforme a Figura 31(a) e *derbytools.jar* na Figura 31(b).



(a) (b)

Figura 31 – Configuração da variavel *CLASSPATH*: (a) *derbytools.jar*; (b) *derby.jar*

Para executar manualmente o utilitário *sysinfo*, deve ser digitado o seguinte comando em uma janela de comandos ou no interpretador de comandos:

java org.apache.derby.tools.sysinfo

Para executar manualmente o utilitário *ij*, deve-se digitar em uma janela de comandos ou no interpretador de comandos:

java org.apache.derby.tools.ij