COOKBOOK
MANUAL DE INSTANCIAÇÃO DO *FRAMEWORK* ViMeT

VERSÃO 1.1

Ana Cláudia Melo Tiessi Gomes de Oliveira

Fevereiro 2007
UNIVEM – Marília – SP – Brasil

SUMÁRIO

1 INTRODUÇÃO

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

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

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

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

1.2 O framework ViMeT

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

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

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

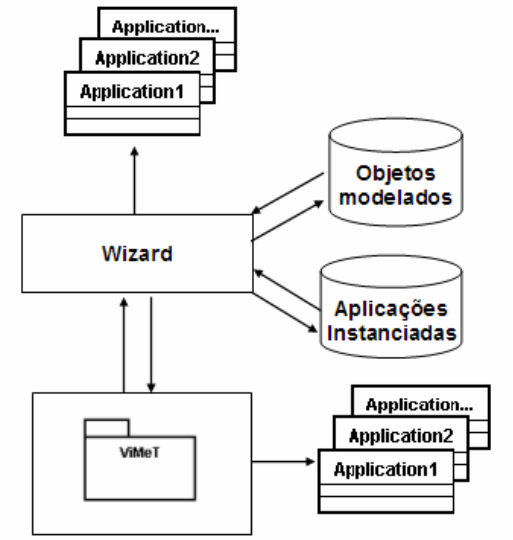


Figura 1 – Projeto arquitetural do *ViMeT*

2 PROCESSO DE INSTANCIAÇÃO MANUAL

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

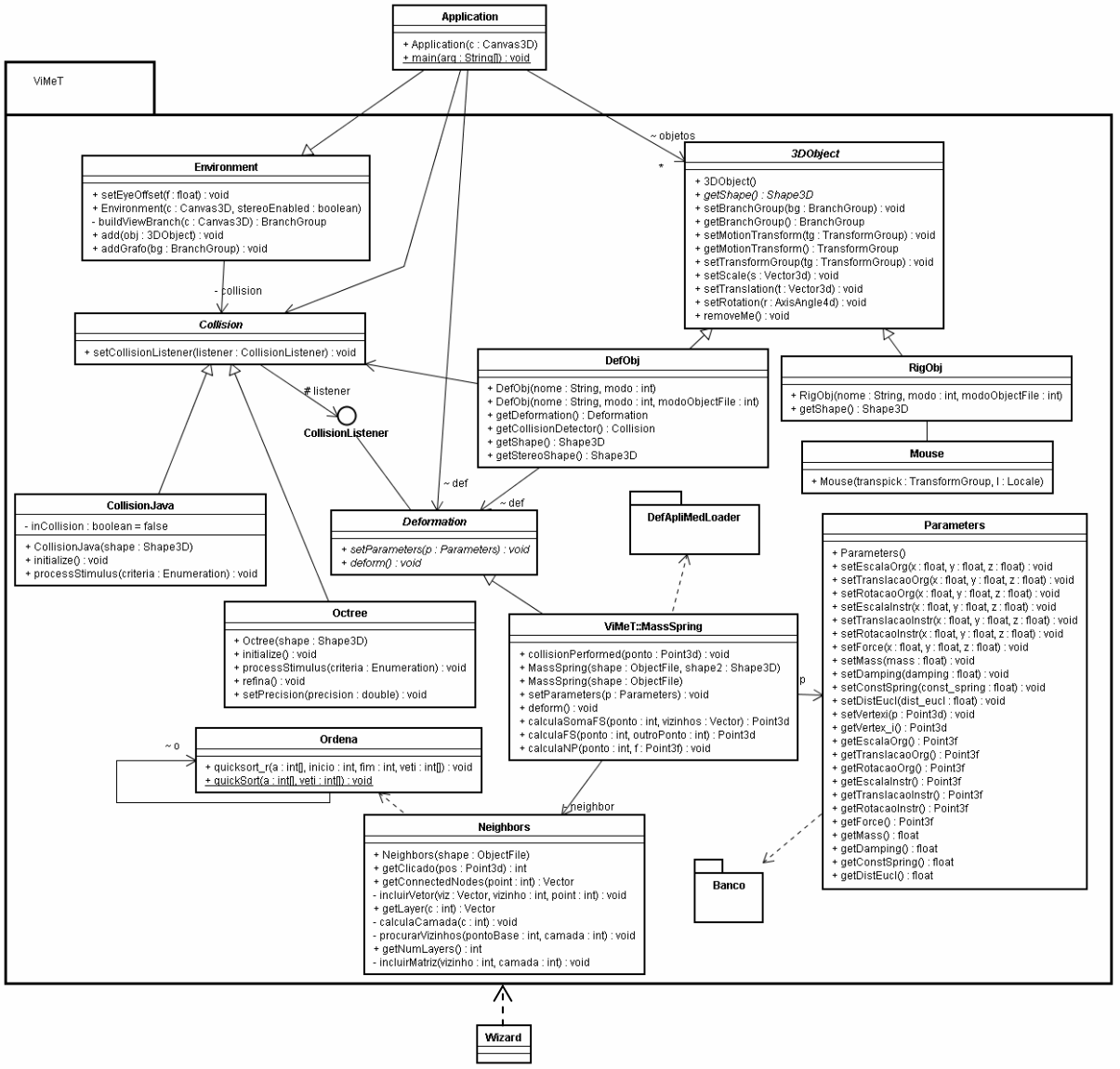


Figura 2 – Diagrama de classes do *ViMeT*

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

2.1 Exemplo de uma instanciação manual

Para facilitar a instanciação do *ViMeT* foi criada uma classe *Application* (Figura 3), que tem como função ser um *template* (gabarito) de instanciação. Nesta classe os *hots spots* (partes variáveis) e os *frozen spots* (partes fixas) permaneceram visíveis. Os *frozen spots* do *ViMeT* são representados pela criação do Ambiente Virtual (AV), *background*, luzes, interação por meio do mouse e interface da aplicação desenvolvida. Esta classe serve como um padrão a ser seguido para o desenvolvimento de aplicações para

exames de punção, onde é variável a seleção de técnicas de deformação, detecção de colisão, estereoscopia e seus parâmetros. Porém, se o desenvolvedor desejar, poderá utilizar o *ViMeT* para a construção de aplicações dentro de outro domínio que tenha características semelhantes ao domínio coberto por ele.

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

a. Criação do AV;
b. Carregamento dos objetos e adição das funcionalidades;
c. Alteração dos parâmetros das funcionalidades e objetos modelados;
d. Compilação e execução da aplicação.

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

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

Figura 3 – Código-fonte de classe *Application*

a. Criação do AV

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

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

Figura 4 - Trecho da nova Aplicação denominada *Teste*

b. Carregamento dos objetos e adição das funcionalidades

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

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

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

Figura 5 - Trecho mantido da classe *Application*

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

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

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

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

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

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

c. Alteração dos parâmetros das funcionalidades e objetos modelados;

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

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

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

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

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

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

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

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

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

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

Figura 12 - Trecho mantido da classe *Application*

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

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

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

d. Compilação e execução da aplicação

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

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

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

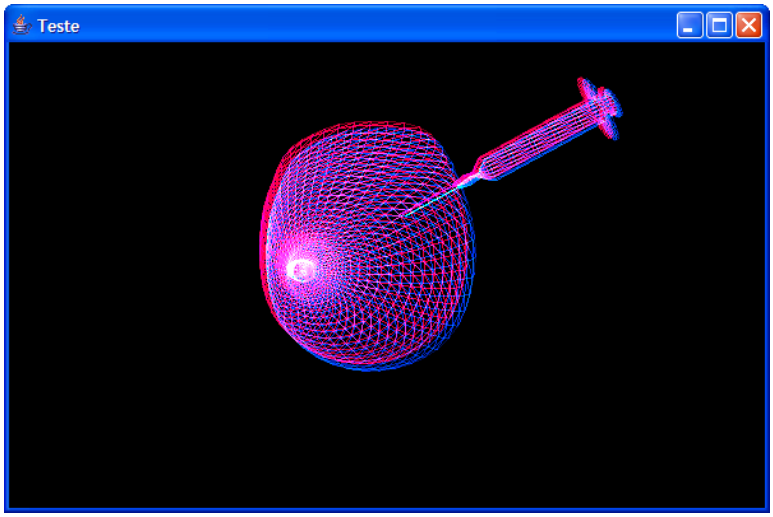


Figura 15 – Resultado da aplicação *Teste*

3 PROCESSO DE INSTANCIAÇÃO AUTOMÁTICO

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

3.1 Instalação do Sistema Gerenciados de Banco de Dados Derby

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

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

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

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

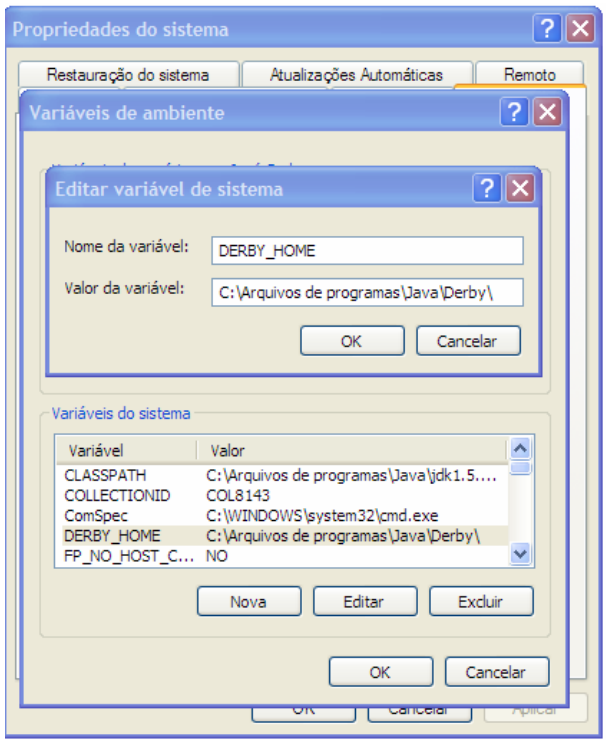
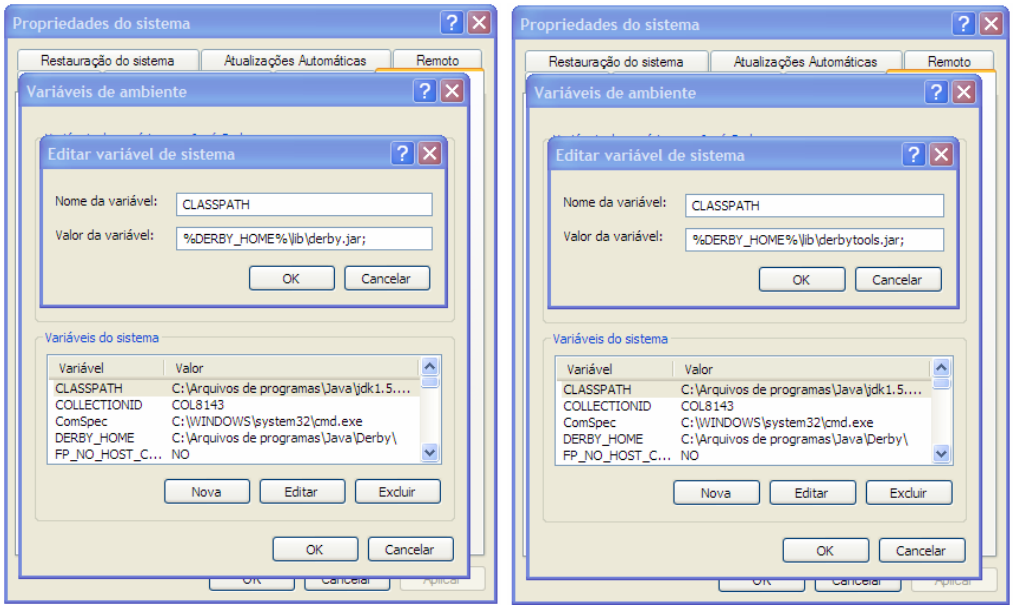


Figura 16 - Configuração da variável DERBY_HOME



(a) (b)

Figura 17 - Configuração da variável CLASSPATH

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

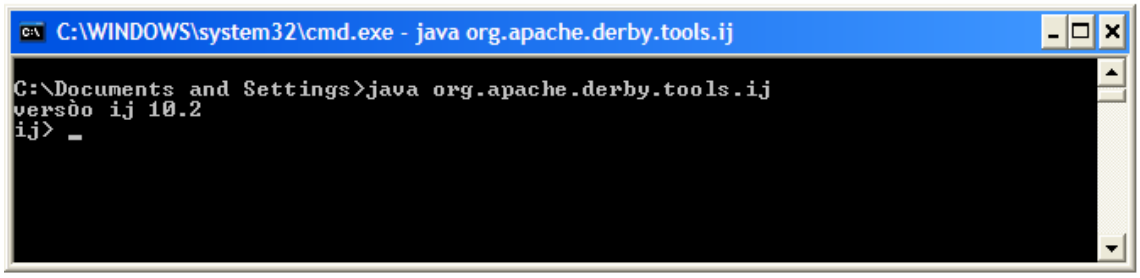


Figura 18 - Execução do aplicativo *ij*

3.2 Exemplo de instanciação automática

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

Figura 19 - Interface da *Wizard*

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

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

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

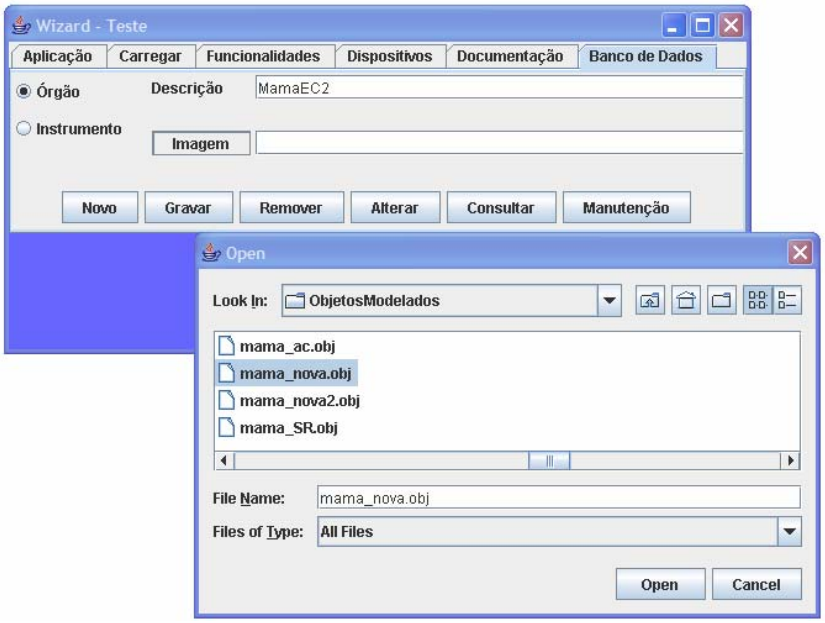


Figura 20- Gravação no Banco de Dados

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

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

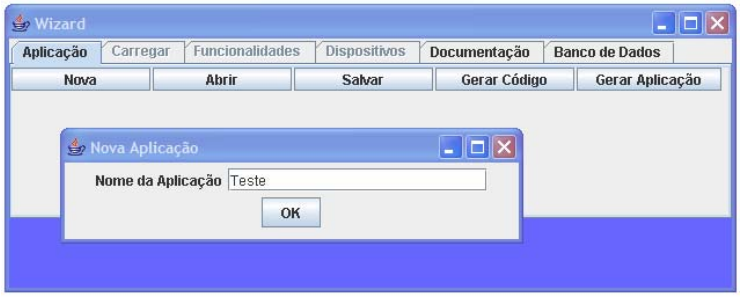


Figura 21 - Criação de uma nova aplicação

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

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

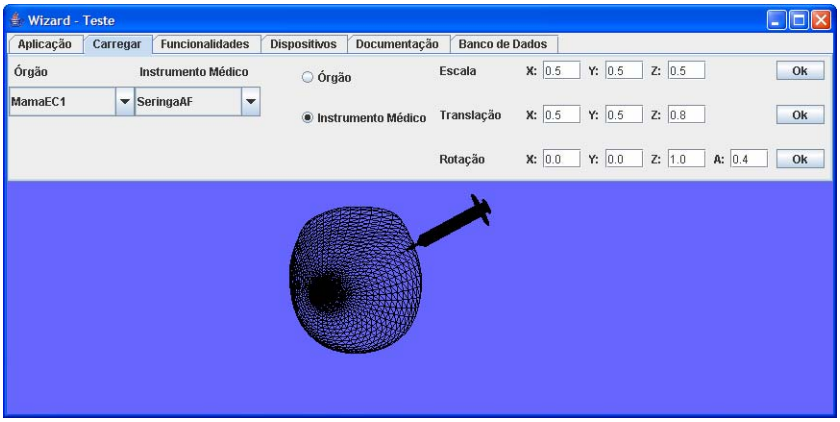


Figura 22 - Carregando os objetos no AV

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

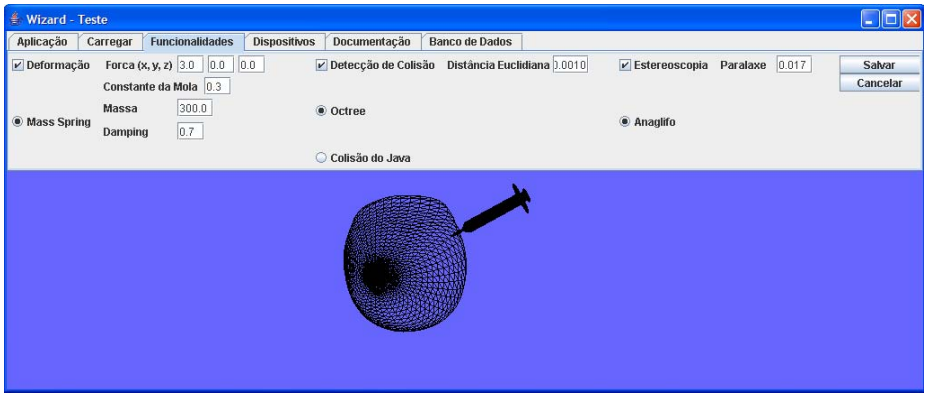


Figura 23- Escolha das funcionalidades e seus parâmetros

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

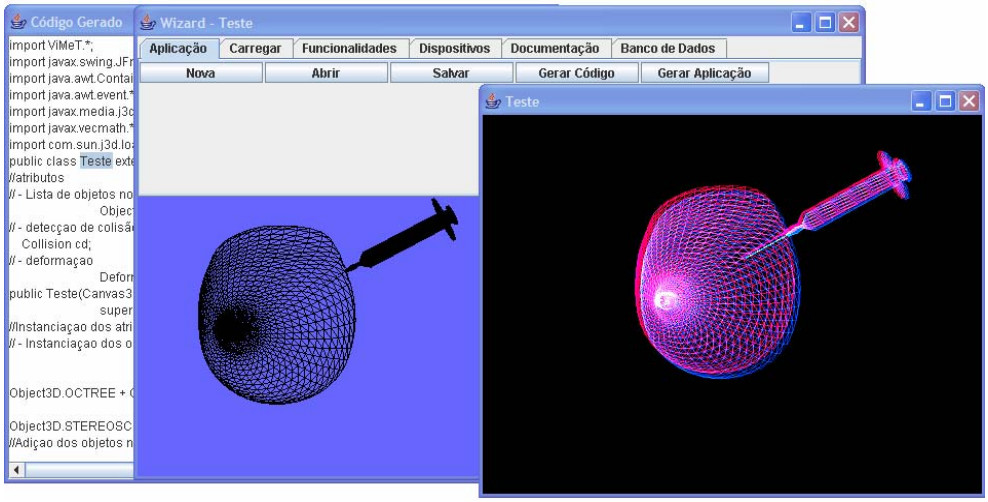


Figura 24 - Código-fonte e aplicação gerada

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

Figura 25 - Abrir uma aplicação salva no BD.

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

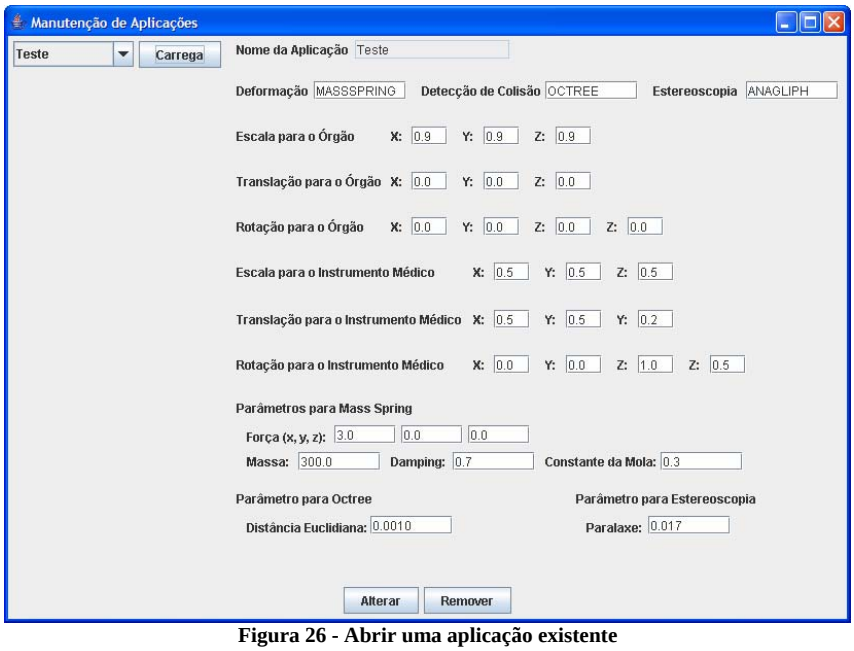


Figura 26 - Abrir uma aplicação existente

4 CONSIDERAÇÕES FINAIS

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

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

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

REFERÊNCIAS

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

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

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

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

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

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

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