Universidade Estadual de Campinas
Faculdade de Engenharia Elétrica e de Computação
Departamento de Engenharia de Computação e Automação Industrial

Desenvolvimento de um ambiente para criação de animações de cenas *VRML* para *Web*

Autora: **Isla Carla Felix da Silva**
Orientador: **Prof. Dr. Léo Pini Magalhães**

Banca examinadora:

Prof. Dr. Léo Pini Magalhães (FEEC/DCA – UNICAMP)
Prof. Dr. Marcelo Knörich Zuffo (Engenharia Elétrica – POLI/USP)
Prof. Dr. Gilberto dos Santos Prado (Instituto de Artes – UNICAMP)
Prof. Dr. Ivan Luiz Marques Ricarte (FEEC/DCA – UNICAMP)

Dissertação submetida à Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas, para preenchimento parcial dos requisitos para obtenção do Título de Mestre em Engenharia Elétrica na área de Engenharia de Computação.

Campinas, 3 maio de 2001.

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

Si38d

Silva, Isla Carla Felix da.
Desenvolvimento de um ambiente para criação de animações de cenas VRML para WEB / Isla Carla Felix da Silva. --Campinas, SP: [s.n.], 2001.

Orientador: Léo Pini Magalhães.
Dissertação (mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Computação gráfica. 2. Animação por computador. 3. World Wide Web (Sistemas de recuperação da informação). 4. Realidade virtual. I. Magalhães, Léo Pini. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

É com muito amor que dedico
este trabalho à minha família
e ao meu noivo.

Agradecimentos

Ao terminar de escrever esta dissertação gostaria de fazer alguns agradecimentos:

Primeiramente gostaria de agradecer a Deus por me iluminar, proteger e dar força para que eu pudesse vencer mais uma etapa na minha vida.

Queria agradecer aos meus pais pelos incentivos, pelas palavras de força nos momentos difíceis, pelas orações e principalmente pelo amor e carinho.

Queria agradecer ao meu noivo pelo amor, carinho, companheirismo, pelos incentivos profissionais e pelo grande otimismo que tantas vezes me deixou mais tranqüila e feliz.

Queria agradecer ao meu irmão e à minha cunhada pelo carinho, amizade, companhia e força nesses últimos anos que passei em Campinas.

Queria agradecer ao meu orientador pela confiança depositada em mim, pela amizade e principalmente pela grande colaboração no desenvolvimento deste trabalho.

Queria agradecer aos meus amigos da UNICAMP pelas conversas e risadas que fizeram com que esses dois últimos anos passassem muito rápido e deixassem saudades.

E finalmente gostaria de agradecer ao CNPq pelos dois anos de bolsa auxílio fornecida; sem a qual não seria possível desenvolver este trabalho.

Resumo

O desenvolvimento de um ambiente para a construção de animações interativas na *Web* é o tema abordado neste trabalho. Neste contexto, deve-se entender por animações interativas aquelas definidas através de ações do usuário. A interação com o ambiente desenvolvido possibilita que o usuário obtenha, de forma simples, uma animação.

VRML (Virtual Reality Modeling Language) é a linguagem utilizada para a modelagem das cenas gráficas e *Java* a linguagem de programação usada para atribuir processamento às cenas. Desta forma, as animações interativas são obtidas através da combinação da *VRML* com a linguagem de programação *Java*. Esta combinação realizada entre *Java* e a cena *VRML* é feita através da utilização da *EAI (External Authoring Interface).*

A forma final de apresentação do ambiente desenvolvido é uma página *HTML* contendo uma cena *VRML* e um painel de controle (*applet Java*) através do qual o usuário irá elaborar e visualizar sua animação. O uso da *EAI* para fazer a comunicação do painel de controle com a cena gráfica *VRML* possibilita que o ambiente desenvolvido seja facilmente divulgado através da *Web.*

A construção de animações interativas é um processo que pode exigir do usuário um amplo esforço e o conhecimento de técnicas de modelagem e tecnologias de programação. O ambiente acima referido foi desenvolvido para facilitar este processo e tornar desnecessário ao usuário o conhecimento dos detalhes envolvidos na criação de animações interativas. Isto amplia a gama de usuários pois possibilita sua utilização sem a necessidade de profundos conhecimentos técnicos.

Abstract

The development of an environment for interactive animations building in the Web is the subject of this thesis. In this context, interactive animations should be understood like the ones defined through user actions. The interaction with the developed environment enables the animations building in a simple way.

VRML (Virtual Reality Modeling Language) is the language used to model the graphical scenes and Java the programming language used to command the scenes animation. Hence, the interactive animations are built through the combination between VRML and Java language. The integration between Java and VRML is enabled by EAI (External Authoring Interface).

The final layout for the developed environment is an HTML page containing a VRML scene and a control panel (Java applet) on which the user will elaborate and visualize her animation. The use of EAI to enable the communication between the control panel and the VRML scene promotes the propagation of the developed environment through the Web.

The interactive animation building is a process that requires a great amount of efforts and deep knowledge on modeling techniques and programming technologies. The developed environment offers an easier way to create interactive animations, hiding the technical details of the referred process. Such automation broadens considerably the number of users, enabling the interactive animations building needless deep technical knowledge.

Índice

Lista de figuras

Lista de tabelas

Glossário

API	*Application Programming Interface*. A especificação de como um programador escrevendo uma aplicação acessa o estado e comportamento de classes e objetos [TERMINOLOGY, 2001].
Applet	*Applet* é um programa escrito na linguagem de programação *Java* que pode ser incluído em uma página *HTML*, da mesma forma que uma imagem [SUN, 1998]. Quando o *browser* está habilitado com a tecnologia *Java* o código *applet* é transferido para o sistema do cliente e é executado pela máquina virtual *Java* do *browser*.
AWT	*Abstract Windowing Toolkit* – é um conjunto de classes que permite criar uma interface gráfica com o usuário e receber entrada do usuário a partir do *mouse* e do teclado [JAWORSKI, 1999].
Browser	Um programa de computador que interpreta arquivos e apresenta o seu conteúdo ao usuário.
EAI	*External Authoring Interface*. Uma interface para permitir que ambientes externos acessem os nós de uma cena *VRML*.
Home page	Originalmente, *Home Page (*ou *Homepage)*é a página *Web* que é carregada quando o *browser* inicia.
HTML	*HyperText Markup Language* – é a linguagem utilizada para a criação de páginas para a WWW.

Java	*Java* é uma linguagem de programação orientada a objetos que pode ser utilizada em diferentes plataformas e que é muito conveniente para o desenvolvimento de *software* que funcione em conjunto com a *Internet*.
JDK	*JDK – Java Development Kit*. Um pacote desenvolvido pela Sun Microsystems que implementa um conjunto básico de ferramentas necessárias para escrever e testar *applet* e aplicações *Java*.
Link	Seu termo técnico é *URL (**U**niform **R**esource **L**ocator)*. Também é usado para definir um endereço que pode levar a um arquivo ou computador em qualquer parte da *Internet*.
Plug-in	Um módulo de *software* que estende as capacidades de outros programas. *Browsers* para *World Wide Web* fazem uso de *plug-in* para estender suas capacidades em áudio, vídeo, telefonia, realidade virtual e muitas outras aplicações [BLAXXUN, 2000a].
RGB	*Red Green Blue*. Um espaço de cor em que as cores são descritas em valores de vermelho, verde e azul.
VRML	*Virtual Reality Modeling Language* – linguagem textual independente de plataforma para a descrição de cenas gráficas *3D*.
Web	*World Wide Web* (conhecido como "*WWW*", "*Web*" ou "*W3*") é um subconjunto da *Internet*. Trabalha sobre um protocolo conhecido como *HTTP* (*HyperText Transfer Protocol*). Através do uso de técnicas de hipertextos e multimídia, a *Web* é um ambiente de fácil navegação e publicação de informações.

Capítulo 1.

Introdução

O objetivo deste trabalho é explorar animações interativas modeladas por computador e cujo comportamento seja determinado por ações do usuário.

A motivação para o desenvolvimento deste trabalho surgiu a partir da leitura e discussão de uma dissertação de mestrado [TAMIOSSO, 1998]. Naquele trabalho, animações interativas eram obtidas através da combinação de *scripts* programados em *Java* e *VRML (Virtual Reality Modeling Language)*. O processo para construir este tipo de animação interativa consistia em definir a cena *VRML*, programar um *script Java* responsável pela animação e definir uma outra aplicação *Java* responsável pela interação com o usuário.

A criação de animações interativas é um trabalho complexo que exige do usuário considerável esforço e conhecimento de técnicas de modelagem e de programação.

A proposta desenvolvida neste trabalho visa facilitar o processo de criação de animações interativas através de um ambiente versátil que estende as facilidades oferecidas pelo trabalho anterior. Também visa possibilitar que estas animações sejam criadas e visualizadas na *Web*.

O Capítulo 2 expõe algumas tecnologias e interfaces para animação na *Web*. Um enfoque maior é dado à linguagem *VRML*, interfaces gráficas via *Java Applet* e à interface *EAI (External Authoring Interface)* utilizada para comunicação entre programas *Java* e cenas *VRML*.

A *VRML* foi a linguagem utilizada para a modelagem das cenas gráficas a serem animadas. Esta escolha foi feita devido à popularidade, padronização, solidez, suficiente realismo das cenas, capacidade de interação e quantidade de recursos oferecidos por esta linguagem.

A combinação do *Applet Java* e da *VRML* através do uso da *EAI* possibilitou a criação de um ambiente acessível através da *Web*.

O Capítulo 3 descreve o projeto do ambiente desenvolvido para a geração de animações interativas, suas funcionalidades e a metodologia utilizada para o seu desenvolvimento.

No Capítulo 4, a exposição do ambiente implementado e sua forma de utilização são descritos através de exemplo comentado.

No Capítulo 5, as conclusões obtidas, contribuições e trabalhos futuros são apresentados.

A finalização deste trabalho é realizada através de uma exposição detalhada de como o ambiente desenvolvido foi implementado (Apêndice) e da apresentação das referências bibliográficas utilizadas.

Capítulo 2.

Tecnologias e interfaces para animação na *Web*

Neste capítulo, o objetivo é apresentar o estudo realizado sobre algumas tecnologias de animação para *Web,* bem como uma introdução às tecnologias adotadas neste trabalho.

Na primeira seção, será dada uma introdução ao tema animação. Em seguida, algumas tecnologias para animação na *Web* serão descritas (*Flash*, *Java 3D* e *VRML*). Por último, serão discutidas as formas de interface utilizadas no desenvolvimento do trabalho (*EAI* e *Applet*).

2.1. Animação

A animação em ambientes computacionais é a criação da ilusão de movimento em componentes gráficos. Esta ilusão é criada através do uso de uma particularidade da visão humana: a persistência. Esta característica da nossa visão faz com que a informação apresentada aos nossos olhos persista. Desta maneira, caso seja apresentada uma outra imagem semelhante antes do desaparecimento da anterior, poder-se-á criar a ilusão de movimento.

Esta definição pode ser expandida ao incluir a variação no tempo da posição, cor, textura e estrutura de um objeto, bem como mudanças na iluminação, posição, orientação e foco da câmera.

Animação é usada amplamente na indústria de entretenimento, na educação, na pesquisa científica e em aplicações industriais, tais como sistemas de controle e simuladores de vôo.

Atualmente, o assunto animação também está fortemente relacionado ao uso da *Web*. A utilização do ambiente *Web* propicia uma grande e rápida distribuição de

informação, possibilitando que muitos usuários visualizem e interajam com cenas de animação disponibilizadas na *Internet*. Este é o foco de interesse do presente trabalho.

Animações disponibilizadas no ambiente *Web* podem tornar-se demoradas e desestimulantes em situações que exigem grande quantidade de processamento computacional. Outro problema no uso da *Web* é que algumas das ferramentas utilizadas para o desenvolvimento dessas animações necessitam da instalação de programas *plug-ins* no sistema do usuário. Estes assuntos serão tratados nas próximas seções.

2.2. Tecnologias e produtos para animação na *Web*

Serão apresentadas a seguir as principais tecnologias para animação na *Web* utilizadas atualmente.

A tecnologia escolhida para o desenvolvimento do trabalho foi a *VRML* devido à sua popularidade, padronização, solidez, suficiente realismo das cenas, capacidade de interação e quantidade de recursos oferecidos.

2.2.1. *Flash*

Flash [MACROMEDIA, 2000] é um dos produtos mais recentes para desenvolvimento de aplicações *Web* avançadas, auxiliando, por exemplo, na criação de seqüências animadas com som.

Esta tecnologia tem como principal característica o fato de gerar imagens e animações em formato vetorial. Isto permite que as imagens tenham ótima qualidade gráfica, podendo ser redimensionadas sem problemas. Permite também que os arquivos sejam extremamente compactos, adaptando-se de forma ideal às limitações de transmissão de dados pela *Internet*. Outra característica importante da tecnologia *Flash* é sua eficiente implementação de técnicas de *anti-alising* [1], garantindo boa

[1] Anti-alising: técnica utilizada para minimizar ruídos em imagens

qualidade de visualização das animações.

Para que uma animação *Flash* possa ser vista é necessário que a máquina que esteja sendo utilizada para sua visualização tenha instalado em seu sistema um programa *plug-in* da *Macromedia* (empresa criadora do *Flash*). Alternativamente, é possível a execução de uma animação *Flash* sem a instalação explícita do referido programa *plug-in*, sendo necessário para tanto que se tenha instalado uma versão de *browser* que já o contemple.

Existem dois métodos para se criar uma seqüência de animação no *Flash*: animação quadro a quadro e animação interpolada. Na animação quadro a quadro, a imagem é criada em cada quadro. Enquanto na animação interpolada, os quadros chaves são definidos e o *Flash* cria os quadros intermediários.

2.2.2. *Java 3D*

A *API Java 3D* [COMPUTING, 1999] [RAPOSO, 2000a] é uma extensão da *API Java* padrão. Esta *API 3D* pode ser utilizada para criar gráficos tridimensionais, *applets* e aplicações.

Em *Java 3D* são várias as plataformas de execução (*Windows*, *UNIX*, *Macintosh, Linux* e *JavaOS*) possíveis para seus programas podendo estes fazer uso de *APIs* gráficas (*OpenGL* [NEIDER, 1996], *Direct3D* e *QuickDraw3D*) pertencentes ao sistema operacional escolhido.

Através da *API Java 3D* os programadores de aplicações gráficas tridimensionais passam a ter as facilidades e vantagens apoiadas pela plataforma *Java*. Dentre elas destacam-se a orientação a objetos, a portabilidade e a segurança.

A *API Java 3D* consiste de dois pacotes: javax.media.j3d e *javax.vecmath* além de se utilizar classes e interfaces do pacote *com.sun.j3d.utils* [JAWORSKI, 1999]. O pacote *javax.media.j3d* provê as interfaces e classes básicas que implementam a *API Java 3D*. A classe *javax.vecmath* apoia operações matemáticas baseadas em vetor. O

pacote *com.sun.j3d.utils* não é documentado como parte da *API Java 3D*, mas contém classes e interfaces para construção de *applets* e aplicações *3D*.

- **Pacote *javax.media.j3d***

  javax.media.j3d consiste de três interfaces e mais de 100 classes que apoiam operações *3D* básicas. As características dadas pelas classes e interfaces *3D* incluem:

  1- Representação do objeto *3D* – existência de pontos, retângulos, triângulos, texto e outros objetos *3D*.
  2- Visão múltipla – mundos *3D* podem ser visualizados de várias perspectivas.

- **Pacote *javax.vecmath***

  É um conjunto de classes que implementa objetos *3D*. Estas classes representam pontos, cores, vetores, matrizes e ângulos em várias dimensões.

Uma aplicação *Java 3D* é projetada a partir de um grafo de cena possibilitando ao programador criar mundos virtuais com personagens que podem interagir entre si e/ou com o usuário. O grafo utilizado é uma estrutura no formato de árvore cujos nós são objetos gráficos, luz, som, entre outros.

2.2.2.1. Animação

Em *Java 3D* existem dois tipos de animações: as animações interativas e as não interativas.

2.2.2.1.1. Animações Interativas

Programar um objeto para reagir a determinados eventos é uma capacidade desejável na grande maioria das aplicações *3D* [RAPOSO, 2000a]. E a reação gerada

por um determinado evento tem por objetivo provocar alguma mudança no mundo virtual. Esta mudança dependerá da natureza da aplicação e será determinada pelo programador. Um exemplo de evento pode ser pressionar uma tecla, movimentar o *mouse* ou colidir objetos e sua reação seria alterar algum objeto (mudar cor, forma ou posição) ou o grafo de cenas (adicionar ou excluir um objeto). Quando estas alterações são resultantes diretas da ação do usuário elas são denominadas interações e quando independem do usuário são denominadas animações interativas.

Java 3D implementa os conceitos de interação e animação interativa na classe abstrata *Behavior*. Esta classe disponibiliza ao desenvolvedor de aplicações *3D* uma grande variedade de métodos para capacitar seus programas a perceber e tratar diferentes tipos de eventos. Permite ainda que o programador implemente seus próprios métodos.

Os *behaviors* são os nós do grafo de cena usados para especificar o início da execução de uma determinada ação baseado na ocorrência de um conjunto de eventos, denominado *WakeupCondition*, ou seja, quando determinada combinação de eventos (condição) ocorrer *Java 3D* deve acionar o *behavior* correspondente para que execute as alterações programadas.

2.2.2.1.2. Animações não Interativas

Algumas alterações do mundo virtual podem ser realizadas independentemente da ação do usuário. Elas podem, por exemplo, ser disparadas em função do passar do tempo. Estas alterações são denominadas animações não interativas.

Estas animações em *Java 3D* também são implementadas usando *behaviors*. Para tanto, *Java 3D* possui alguns conjuntos de classes, baseadas em *behaviors*, que são próprias para implementar tais animações *Java*.

Um primeiro conjunto destas classes é formado por interpoladores que são versões especializadas de *behaviors* usadas para gerar uma animação baseada no tempo.

Existem também, as classes *Billboard* e *LOD* que permitem especificar animações mas, neste conjunto de classes as alterações são guiadas segundo a orientação ou posição da visualização.

2.2.3. *Virtual Reality Modeling Language – VRML*

VRML é uma linguagem independente de plataforma que permite a criação de ambientes virtuais. Nestes ambientes pode-se navegar, visualizar objetos por ângulos diferentes e interagir com eles.

A linguagem *VRML* surgiu da necessidade de prover um formato gráfico *3D* para a *Web* seguindo um modelo similar à *HTML*, ou seja, uma linguagem textual independente de plataforma para a descrição de cenas. A linguagem escolhida como referência foi a *Open Inventor* da *SGI*. Em 1995 foi lançada a *VRML* 1.0, que era basicamente um formato para a descrição de cenas estáticas *3D*. Em 1997 foi lançada a *VRML* 2.0 (ou *VRML* 97) [RAPOSO, 2000a] [TAMIOSSO, 1997a] [TAMIOSSO, 1997b] [WEB, 1997], que adicionou à linguagem conceitos, tais como possibilidade de mover objetos na cena e criação de sensores para detectar e gerar eventos.

A linguagem *VRML* foi projetada pelo consórcio das companhias *Silicon Graphics, Sony Research* e *Mitra* com a finalidade de obter uma padronização na descrição de cenas *Web* [TAMIOSSO, 1998]. Até 1999, este consórcio se chamava *VRML Consortium*, e depois passou a se chamar *Web 3D Consortium* [WEB, 1999a].

O objetivo da *VRML* é levar conceitos da realidade virtual ao usuário comum, através da *Internet*. Com o rápido avanço das tecnologias, os computadores pessoais estão cada vez mais rápidos e poderosos e isto faz com que a realidade virtual deixe de ser objeto de estudo dos grandes centros de pesquisa e possa ser utilizada também por usuários comuns.

A linguagem *VRML* acrescenta percepção à navegação na *Web* através de descrições completas da cena em que o usuário se encontra, montando verdadeiros

"mundos virtuais" (razão pela qual os arquivos *VRML* apresentam a extensão *".wrl"* , de *word reality language*).

Para se escrever um código *VRML* apenas um editor de textos é necessário. Uma vez editados, os arquivos são gravados em formato *ASCII*. Na verdade, a linguagem apenas descreve como os ambientes tridimensionais devem ser representados. O código *VRML* é interpretado.

Pode-se, por exemplo, criar um cubo e gravá-lo em um arquivo chamado cubo.*wrl*. O código *VRML* para este cubo descreverá as características do ambiente, como coordenadas, luz e cores.

Objetos localizados remotamente (em outros lugares na *Internet*) e *links* para outras *homepages* também podem ser colocados no mundo *VRML*.

É conveniente comentar que as primitivas geométricas de *VRML* (caixa, cone, cilindro e esfera) são, por definição, sólidas e não permitem a sua visualização interna, a não ser que algumas partes sejam removidas (por exemplo, a base de um cilindro).

Para completar as possibilidades criativas básicas em *VRML* precisa-se deslocar, modificar a escala e rotacionar os objetos previamente definidos. A linguagem *VRML* permite realizar as chamadas transformações geométricas, que dão uma enorme flexibilidade e aumentam notadamente o poder criativo do construtor. De posse deste conhecimento pode-se construir o mundo idealizado.

Para fins de identificação, um arquivo *VRML* 2.0 apresenta o seguinte cabeçalho:

```
#VRML V2.0 utf8.
```

VRML é baseada no sistema cartesiano *3D*, sendo as unidades de medida de distância e ângulos (eixos X, Y e Z) metros e radianos respectivamente. Usando uma página na frente do leitor como referência, o eixo-X positivo está para a direita, o eixo-Y positivo está para cima e o eixo-Z positivo está perpendicular aos dois

anteriores, saindo da página em direção ao leitor. O sentido de rotação para ângulos positivos é o anti-horário, quando a seta vai em direção ao observador.

2.2.3.1. Estrutura hierárquica da cena

O paradigma para a criação de mundos *VRML* é baseado em nós. Estes nós são conjuntos de abstrações de objetos e de certas entidades do mundo real, tais como formas geométricas, luz e som. Nós são os componentes fundamentais de uma cena *VRML* porque esta é constituída a partir da disposição e combinação entre os nós.

Um mundo *VRML* é um grafo hierárquico em forma de árvore. As hierarquias são criadas através de nós de agrupamento, os quais contêm um campo chamado *children* que engloba uma lista de nós filhos. Há vários tipos de nós em *VRML* [HARTMAN, 1996] [VRML, 1997] [WEB, 1997]:

1- **Agrupamento** – como já comentado, nós de agrupamento criam a estrutura hierárquica da cena e permitem que operações sejam aplicadas a um conjunto de nós simultaneamente. Alguns exemplos desse tipo de nó são:

- ***Anchor*** - É um nó de agrupamento que recupera o conteúdo de uma *URL* quando o usuário ativa alguma geometria contida em algum de seus nós filhos (clica com o mouse sobre eles, por exemplo). É o nó que cria *links* com outros mundos *VRML*, páginas *HTML*, ou qualquer outro tipo de documento presente no referido *URL*.

- ***Transform*** - É um nó de agrupamento que define um sistema de coordenadas para seus filhos que está relacionado com o sistema de coordenadas de seus pais. Sobre este sistema de coordenadas podem ser realizadas operações de translação, rotação e escalonamento.

- ***Group*** - É um nó de agrupamento que contém um número qualquer de filhos. Ele é equivalente ao nó *Transform* sem os campos de transformação. A diferença básica entre o *Group* e o *Transform* é que o primeiro é usado quando se deseja criar um novo objeto constituído da combinação de outros. Quando se deseja agrupar espacialmente os objetos, ou seja, posicioná-los em uma certa região da cena, usa-se o nó *Transform*.

2- **Geométrico** – Define a forma e a aparência de um objeto do mundo. O nó *Shape*, em particular, possui dois parâmetros: *geometry*, que define a forma do objeto e *appearance*, que define as propriedades visuais dos objetos. Alguns exemplos de nós geométricos são:

- ***Box*** – Este nó geométrico representa um sólido retangular (uma "caixa") centrado na origem (0, 0, 0) do sistema de coordenadas local e alinhado com os eixos cartesianos. O campo *size* representa as dimensões do sólido nos eixos x, y e z.

```
Box { size 2 2 2 }
```

- ***Cone*** – Representa um cone centrado na origem do sistema local de coordenadas cujo eixo central está alinhado com o eixo y.

- ***Cylinder*** – Define um cilindro centrado na origem do sistema de coordenadas e com o eixo central ao longo do eixo y.

- ***Sphere*** – Especifica uma esfera centrada na origem.

3- ***Appearance*** - Este nó aparece apenas no campo *appearance* de um nó *Shape* e é responsável pela definição das propriedades visuais das figuras geométricas (material e textura).

4- **Câmera** - O nó *Viewport* define, entre outros parâmetros, a posição e orientação da câmera (ponto de vista do usuário). Este tipo de nó é chamado *blindable* porque apenas um pode estar ativo na cena.

5- **Iluminação** - Luzes em *VRML* não são como luzes no mundo real. No mundo real, luzes são objetos físicos que emitem luz e que podem ser vistos, assim como a luz por eles emitida. Em *VRML*, um nó de iluminação é descrito como parte do mundo mas, não cria automaticamente uma geometria para representá-lo. Para que uma fonte de luz em uma cena seja um objeto visível é necessário criar uma geometria e colocá-la em um determinado local na cena. Existem três tipos de nós de iluminação em *VRML*:

   - ***DirectionalLight*** - Nó que define uma fonte de luz direcional com raios paralelos.

   - ***PointLight*** - Nó que define uma fonte de luz pontual em um local *3D* fixo. Este tipo de fonte ilumina igualmente em todas as direções.

   - ***SpotLight*** - Nó que define uma fonte de luz que projeta um cone de iluminação.

Além desses tipos básicos, existem ainda os nós sensores, os interpoladores e o nó *Script*, que serão vistos mais adiante.

2.2.3.2. Prototipação e reutilização

Os mecanismos de prototipação em *VRML* permitem definir um novo tipo de nó baseado na combinação de nós já existentes. Permitem também, a criação de cenas distribuídas pois um subgrafo (protótipo) pode ser definido em um arquivo remoto cujo *URL* é conhecido.

Atribuindo-se um nome a um nó através da palavra *DEF*, pode-se futuramente referenciá-lo através da palavra *USE*. Sendo assim, caso seja necessária à reutilização de um mesmo nó várias vezes em uma cena, é mais eficiente atribuir-lhe um nome na primeira vez que ele é descrito e posteriormente referenciá-lo por este nome. Essa técnica torna o arquivo menor, mais fácil de ser lido, e diminui o tempo necessário para carregar a cena.

2.2.3.3. Tipos de parâmetros e roteamento de eventos

Cada nó *VRML* define um nome, um tipo e um valor *default* para seus parâmetros. Estes parâmetros diferenciam um nó de outros de mesmo tipo (por exemplo, o raio de uma esfera a diferencia de outra). Há dois tipos de parâmetros possíveis: campos e eventos. Os campos podem ser de dois tipos: modificáveis *(exposedFields)* ou não *(fields).*

Os eventos podem ser enviados para outros nós por um parâmetro do tipo *eventOut* (um terminal lógico anexado a um nó do qual eventos são enviados) e recebidos por um *eventIn* (um receptor lógico anexado a um nó que recebe eventos).

Estes eventos sinalizam mudanças causadas por “estímulos externos" e podem ser propagados entre os nós da cena por meio de roteamentos *(Routes)* que conectam um *eventOut* de um nó a um *eventIn* de outro nó, desde que sejam eventos do mesmo tipo.

2.2.3.4. Sensores e interpoladores

Os nós sensores e interpoladores são especialmente importantes porque são os responsáveis pela interatividade e dinamismo dos mundos *VRML*.

Os sensores são responsáveis pela interação com o usuário, mas não estão restritos a gerar eventos a partir de ações dos mesmos. O *TimeSensor*, por exemplo, gera automaticamente um evento a cada pulso do relógio. Os *eventOuts* gerados pelos sensores podem ser ligados a outros *eventIns* da cena, iniciando uma animação.

A seguir alguns sensores serão descritos:

- ***TimeSensor*** - O nó *TimeSensor* gera eventos como passos de tempo e em conjunto com interpoladores pode produzir animações.

- ***TouchSensor -*** O nó *TouchSensor* detecta quando um objeto do grupo do seu pai é ativado (clique do *mouse*, por exemplo). Esse sensor gera um evento de saída chamado *touchTime* que pode disparar o *TimeSensor* iniciando uma animação.

- ***ProximitySensor -*** O nó *ProximitySensor* gera eventos quando o usuário entra, sai e/ou se move em uma região do espaço. O sensor de proximidade é habilitado ou desabilitado pelo envio de um evento *enabled* com o valor *TRUE* ou *FALSE.*

- ***VisibilitySensor*** - O nó *VisibilitySensor* detecta quando certa parte do mundo (área ou objeto específico) torna-se visível ao usuário. Quando a área está visível, o sensor pode ativar um procedimento ou animação.

A forma mais comum de se criar animações é usando *keyframes* (quadros-chave), especificando os momentos-chave na seqüência da animação. Os quadros intermediários são obtidos através da interpolação dos quadros-chave. Nós interpoladores servem para definir este tipo de animação, associando valores-chave

(de posição, cor, etc.) que serão linearmente interpolados. Alguns exemplos de interpoladores são:

- ***PositionInterpolator*** - O nó *PositionInterpolator* permite realizar uma animação *keyframe* no espaço *3D*. Exemplo:

```
PositionInterpolator {
    Key [0, 0.5, 1]
    KeyValue [ 0 0 0, 0 10 0, 0 0 0 ]
}
```

- ***CoordinateInterpolator, ColorInterpolator e OrientationInterpolator*** - De forma similar interpolam, respectivamente, uma lista de coordenadas, valores de cor e valores de rotação.

2.2.3.5. Nó *Script*

Apesar de ser um recurso poderoso, o roteamento de eventos entre os nós não é suficiente para o tratamento de várias classes de comportamento. Por exemplo, não é possível escolher entre duas trajetórias pré-definidas (lógica de decisão). Para superar esta limitação, a *VRML* define um nó especial chamado *Script*, que permite conectar o mundo *VRML* a programas externos, onde os eventos podem ser processados. Este programa externo, teoricamente, pode ser escrito em qualquer linguagem de programação, desde que o *browser* a suporte. Na prática, apenas *Java* [JAWORSKI, 1999] [LEMAY, 1999] [SUN, 1998] e *JavaScript* [SHELLY, 2000] são usadas.

O nó *Script* é ativado pelo recebimento de um evento. Quando isso ocorre, o *browser* inicia a execução do programa definido no campo *URL* do nó *Script*. Este programa é capaz de receber, processar e gerar eventos que controlam o comportamento do mundo virtual.

Por meio do nó *Script* é possível usar técnicas bem mais sofisticadas que a interpolação linear para a geração de animações.

2.3. Interfaces

A seguir, serão discutidas as formas de interface utilizadas no desenvolvimento deste trabalho.

2.3.1. *External Authoring Interface – EAI*

A *EAI* é uma interface para permitir que ambientes externos acessem os nós de uma cena *VRML* [ROEHL, 1997].

Embora a *EAI* tenha objetivos similares aos do nó *Script* (processamento de eventos definindo o comportamento dos objetos do mundo virtual) ela funciona de maneira diferente. A *EAI* opera na forma de um *applet Java* independente, enquanto o nó *Script* opera dentro do *browser VRML*. Dentre as vantagens da *EAI* em relação ao nó *Script* estão incluídas uma maior modularidade e simplicidade dos programas, além de maior liberdade para a construção de interfaces complexas para a interação com o mundo *VRML* [RAPOSO, 2000a].

De uma maneira geral, pode-se dizer que o nó *Script* é adequado quando se deseja adicionar um determinado comportamento a objetos da cena. A *EAI*, por sua vez, é mais adequada para a criação de sistemas complexos.

O *Blaxxun Contact* [Blaxxun, 2000b] é um exemplo de sistema que usa a *EAI*. Ele é um cliente para comunicação multimídia que oferece recursos como apoio a *VRML*, *chats*, *message boards*, interação com agentes, etc. Vários usuários "imersos" em um mesmo mundo virtual *VRML* podem se comunicar e interagir por meio de recursos implementados com a *EAI*.

Para utilizar os recursos da *EAI*, é necessário criar uma página *HTML* incluindo a cena *VRML* e um *applet* que realiza a interação com a cena.

A forma de utilização da *EAI* segue duas seqüências de passos: uma para receber eventos da cena e outra para enviar eventos para a cena.

Neste trabalho, apenas o *applet Java* envia eventos para a cena *VRML*. Este processo é realizado seguindo a seqüência de passos descrita abaixo [ROEHL, 1997]:

1. Obter a referência do nó para o qual se deseja enviar o evento.
2. Obter a referência do *EventIn* deste nó.
3. Enviar os valores desejados por meio do método *setValue.*

Traduzindo em código, consideremos a esfera seguir, que será movimentada através do campo *translation* do nó *Transform.*

```
#VRML v2.0 utf8
DEF move_esfera Transform {
     translation 2 0 −1
     children Shape {
          geometry Sphere { radius 1.5 }
     }
}
```

O seguinte *applet* colocará a esfera em uma nova posição.

```
// Importando algumas classes da EAI
import vrml.external.*;
import vrml.external.field.*;
import java.applet.*;
public class MeuApplet extends Applet {
    public void start ( ) {
      Browser b = b.getBrowser();

      // 1. Obtendo referência do nó Transform
      Node aTransform = b.getNode("move_esfera");

      // 2.Obtendo referência do EventIn "set_translation"
      EventInSVec3f tx =
     (EventInSVec3f)aTransform.getEventIn("set_translation");
```

```
        // 3. Enviando nova posição para o EventIn obtido acima
        float nova_posição[] = { 0, 1, 0.5f };
        tx.setValue (nova_posição);
    }
}
```

Recentemente, a *EAI* e os nós *Script* foram englobados no que está sendo chamado de *Scene Authoring Interface* [WEB, 2000], usada para a manipulação de objetos de uma cena. Esta interface é acessível por meio dos nós *Script* da cena *VRML* ou por meio de aplicações externas ao *browser* (*EAI*).

2.3.2. *Applet*

Os *applets Java* são pequenos programas executados dentro do *browser*.

Applets podem ser carregados e executados por qualquer *Web browser* que ofereça apoio a linguagem *Java*.

Quando um usuário com um navegador compatível com *Java* carregar uma página da *Web* que inclua um *applet*, esse navegador fará o *download* do *applet* a partir de um servidor da *Web* e irá executá-lo no sistema do cliente.

2.3.2.1. Restrições de segurança do *applet*

Como os *applets Java* são executados no cliente da *Web*, existem sérias restrições quanto ao que um *applet* é capaz de fazer [LEMAY, 1999]. Caso essas restrições não existissem, um programador *Java* mal-intencionado poderia facilmente escrever um *applet* que excluísse arquivos do usuário, reunisse informações privativas do sistema e encontrasse outras falhas na segurança.

Como regra geral, os *applets Java* são executados sob um modelo de segurança preventivo. A seguir algumas restrições quanto ao uso de *applets* são listadas:

- Eles não podem ler ou escrever arquivos no sistema do cliente.

- Eles não podem se comunicar com um *site* da *Internet* que não seja aquele que serviu a página da *Web* que o inclui.
- Eles não podem executar nenhum programa no sistema do cliente.
- Eles não podem carregar programas armazenados no sistema do cliente, como programas executáveis e bibliotecas compartilhadas.

Maiores informações sobre segurança em *Java* podem ser obtidas através do endereço: http://java.sun.com/j2se/1.3/docs/guide/security/index.html

A seguir, é apresentada a Figura 2.1 que ilustra o *layout* de uma página *HTML* onde a interação do usuário com o *applet* provoca alterações no mundo *VRML*. Essas alterações são realizadas através da comunicação entre o *browser VRML* e o *applet Java* com uso da *EAI*.

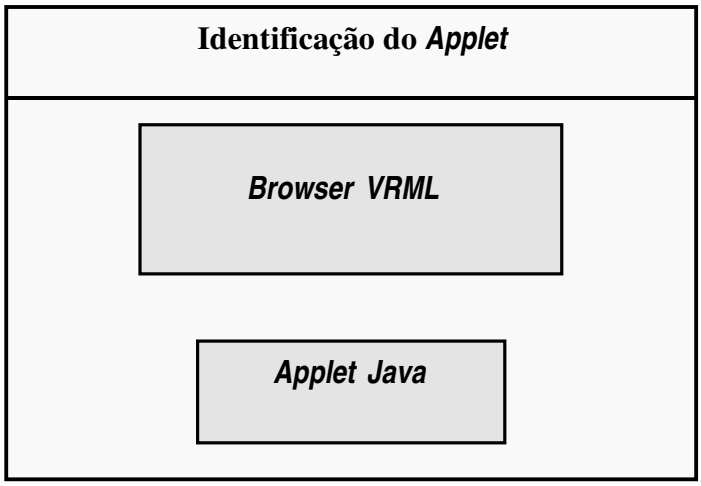

Figura 2.1. *Layout* da página *HTML* com *Applet*

2.4. Considerações finais

Neste capítulo, uma introdução ao tema animação e alguns aspectos da sua relação com a *Internet* foram discutidas. As tecnologias para animação na *Web: flash*, *Java3D* e *VRML* foram expostas, sendo dada uma maior atenção a *VRML* por ser ela a escolhida para o desenvolvimento do trabalho. Por último as interfaces *EAI* e *applet* utilizadas no desenvolvimento do trabalho são também descritas. A exposição do tema de pesquisa (criação de um ambiente para geração de animações interativas) e sua solução são apresentados no Capítulo 3.

Capítulo 3.

Ambiente para geração de animações interativas

O objetivo deste capítulo é descrever o ambiente projetado e desenvolvido para a geração de animações interativas na *Web*.

3.1. Descrição do problema

O problema estudado foi a construção de um ambiente interativo que possibilite a criação de animações gráficas a partir de cenas *VRML* estáticas.

O ambiente criado é subdividido em duas etapas sendo a primeira responsável pelas definições que exigem conhecimento da linguagem *VRML* e conseqüentemente o auxílio de um usuário com conhecimento na área de animação (usuário animador). Na segunda etapa o usuário final fará o trabalho artístico de criação da animação. Ou seja, dois tipos de usuários podem ser identificados neste ambiente:

- Usuário Animador: responsável por definir o arquivo *VRML* a ser utilizado, selecionar os componentes *VRML* que poderão ser animados e definir o tempo de duração da animação.

- Usuário Final: responsável por criar a animação a partir das definições feitas pelo usuário animador. Este usuário, por exemplo, poderá definir movimentos de translação, mudança de escala e alteração de cores de componentes da cena *VRML* em função do tempo.

Na primeira fase de uso do ambiente o usuário animador, de acordo com a Figura 3.1, fornecerá ao sistema uma cena *VRML* estática onde seus componentes não estão necessariamente nomeados e obterá como resultado:

- Uma nova cena *VRML* onde todos os componentes estão nomeados através de *DEF*.

- Um arquivo *Java* correspondente ao painel de controle (*applet Java*), através do qual o usuário final poderá interagir elaborando uma animação.

- Uma página *HTML* que fará referência à cena *VRML* criada e ao painel de controle.

Neste contexto, é importante destacar que a interação com a primeira etapa do sistema é totalmente realizada pelo usuário animador não sendo necessário, por tanto, que o usuário final possua conhecimento de linguagens de programação.

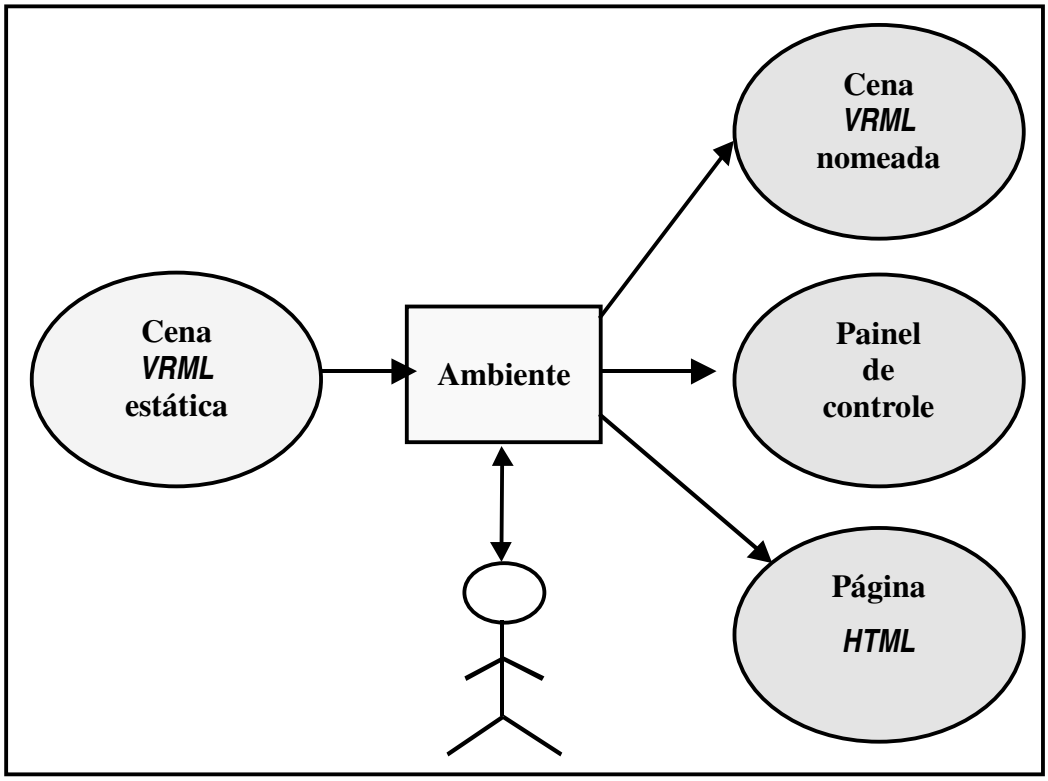

Figura 3.1: Primeira etapa do sistema.

Na segunda fase do sistema o usuário (usuário final), de acordo com a Figura 3.2, interage com o sistema através do painel de controle contido na página *HTML* desenvolvendo a animação correspondente ao resultado final desejado. Nesta fase do sistema, o usuário comporta-se como um artista devendo ser cuidadoso e minucioso na criação dos movimentos, alterações de tamanho e mudanças de cores para os componentes da cena *VRML*.

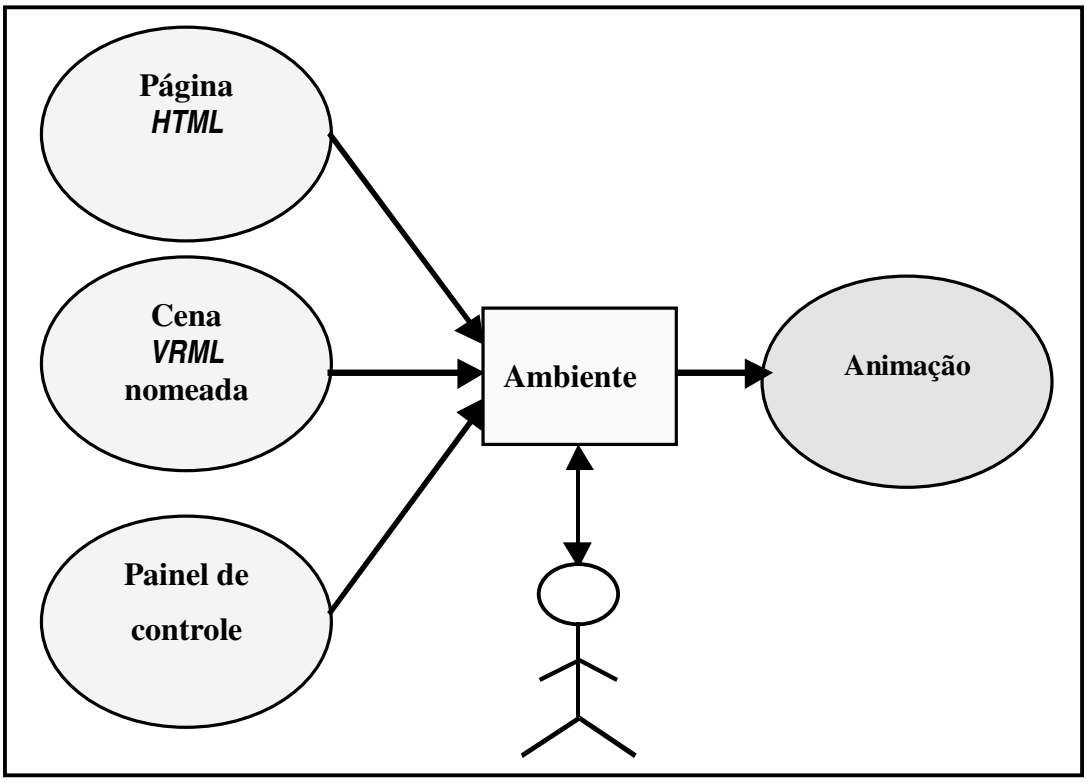

Figura 3.2: Segunda etapa do sistema.

A animação da cena *VRML* estática, como pode ser visto na Figura 3.3, é feita através do painel de controle. A comunicação entre o painel de controle (*applet Java*) e a cena *VRML* é realizada através da utilização da *EAI* (*External Authoring Interface*) descrita anteriormente.

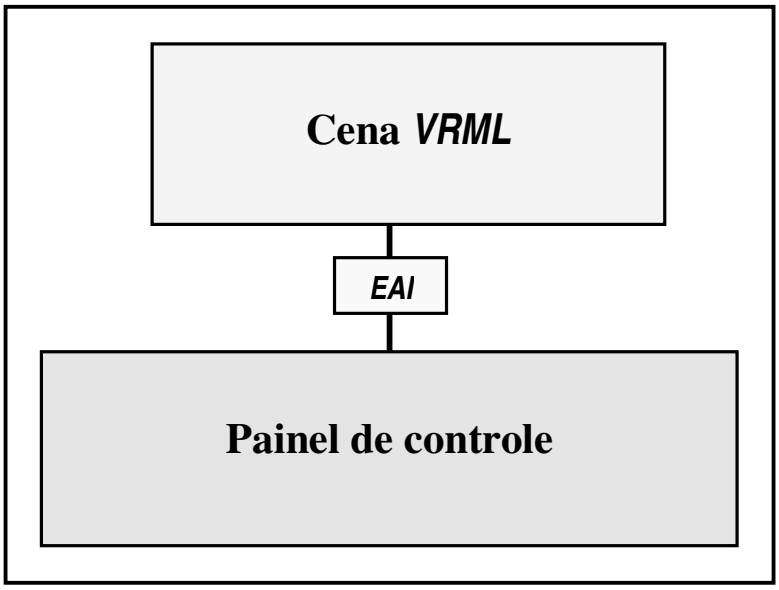

Figura 3.3: Ambiente para criação de uma animação.

3.2. Diagrama funcional do ambiente

A seguir será apresentado, na Figura 3.4, o diagrama funcional do ambiente e em seqüência todos seus módulos serão apresentados.

Uma apresentação detalhada de como foram implementadas as funcionalidades do ambiente desenvolvido será realizada no Apêndice deste trabalho.

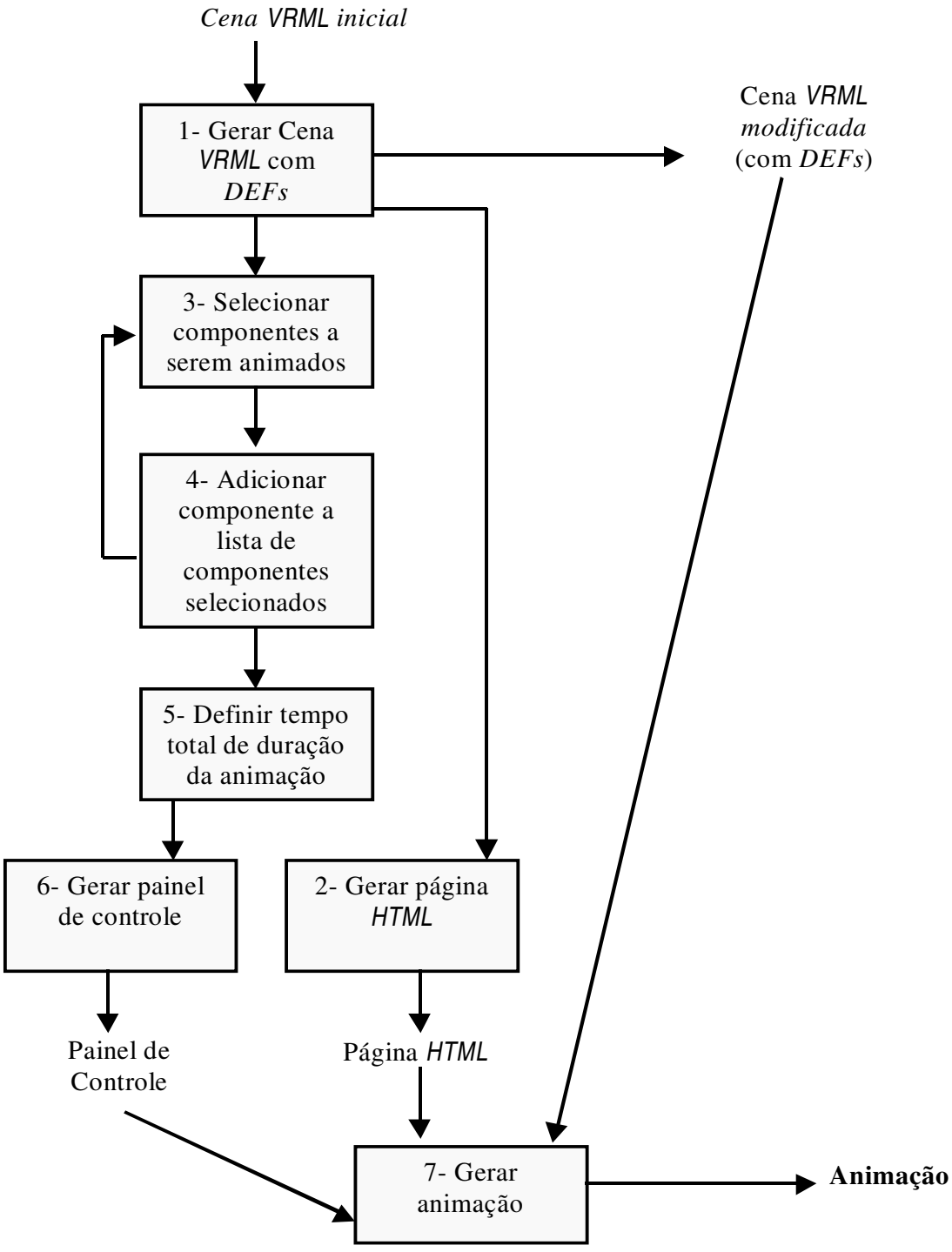

Figura 3.4: Diagrama funcional do ambiente

- **Módulo 1: inclusão de *DEFs***

O usuário animador informa ao sistema uma cena *VRML* dando início ao processo de criação da animação interativa. Porém, como já foi dito anteriormente, para que cena de entrada possa ser animada através do uso da *EAI* é necessário que os nós *VRML* a serem animados sejam nomeados.

Prevendo a entrada de cenas com nós não nomeados, o sistema faz inicialmente a análise do arquivo *VRML* fornecido pelo usuário e gera uma nova cena *VRML* onde todos os componentes são nomeados (definidos). Para que isso se torne possível o ambiente (primeira etapa) deve conhecer a sintaxe da linguagem *VRML*

- **Módulo 2: geração da página *HTML***

Após analisar a cena *VRML* o sistema gera automaticamente o arquivo texto correspondente a página *HTML* (Animation.html).

- **Módulos 3 e 4: seleção e adição de objetos animados pelo animador**

Após a análise da cena *VRML* o ambiente identifica e exibe os componentes da cena relevantes para a animação. Esta exibição é feita para que o usuário possa selecionar, um a um, os nós da cena que deseja animar.

O usuário animador deverá adicionar o componente (nó *VRML*) selecionado à lista de componentes escolhidos para serem animados.

- **Módulo 5: definição do tempo de duração da animação pelo animador**

É neste módulo que o usuário animador define o tempo total de duração da animação que está sendo construída.

- **Módulos 6: geração do *applet Java***

Após terminar a fase de determinação de quais componentes da cena *VRML* serão animados, o sistema desenvolvido fará a geração automática do arquivo texto correspondente ao painel de controle (Animation.java). Esse arquivo juntamente com a página *HTML* (módulo 2) irão fazer parte da segunda fase do ambiente.

- **Módulo 7: animação**

Para que a animação da cena *VRML* possa ser artisticamente realizada o usuário final deverá carregar em um *browser* a página *HTML* gerada.

Depois de carregada a página *HTML* o usuário final criará e visualizará a animação final através da sua interação com o painel de controle (*applet Java*). Nesta etapa, alterações de cores, posições e escalas serão atribuídas à cena em função da componente tempo, produzindo uma animação.

3.3. Metodologia de desenvolvimento do ambiente

O ambiente foi subdividido em níveis para facilitar os testes de validação durante todo o desenvolvimento do ambiente. Os níveis serão apresentados, de acordo com a Figura 3.5, na ordem em que foram desenvolvidos, ou seja, na seqüência do nível 1 ao 4.

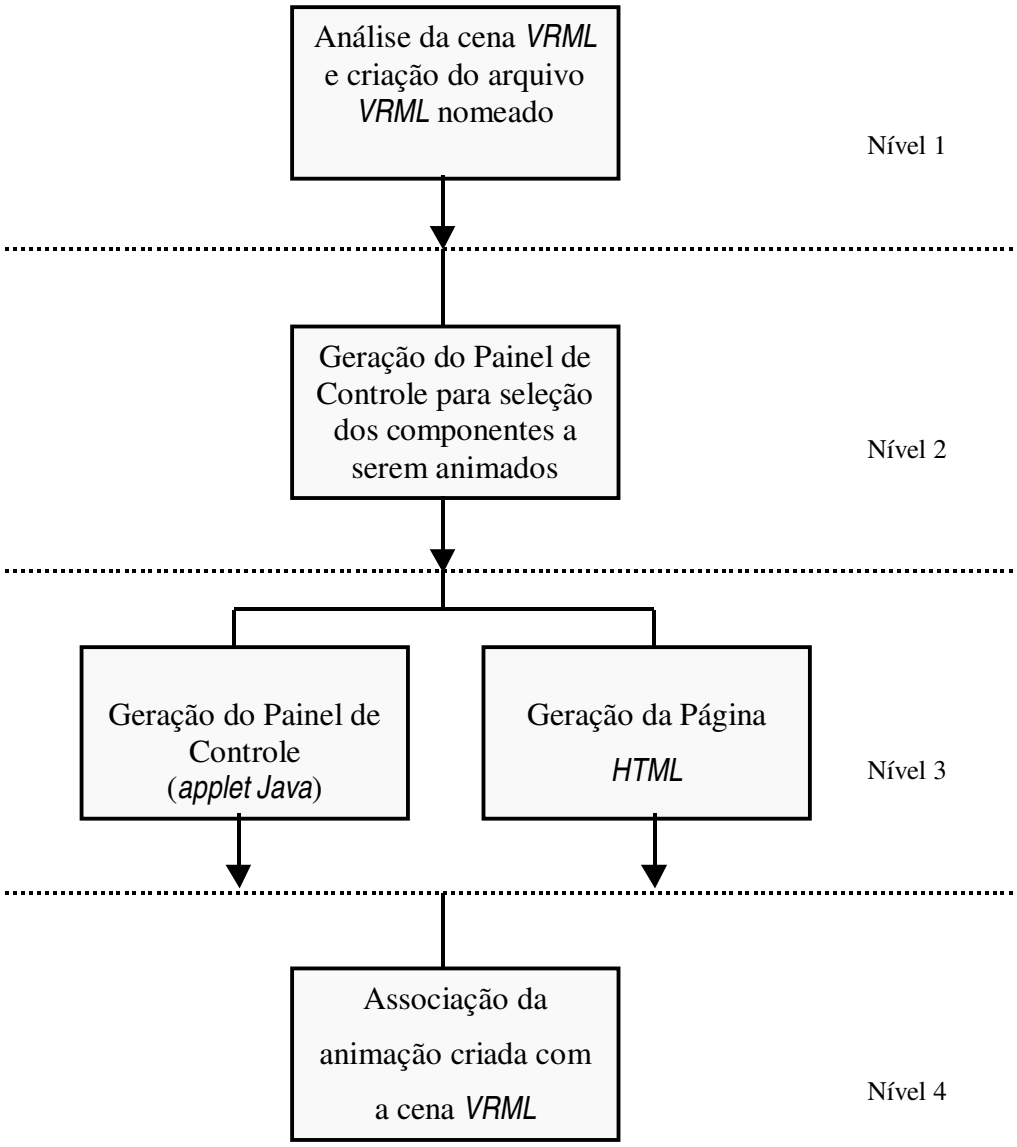

Figura 3.5: Metodologia de desenvolvimento

Para que uma cena *VRML* estática seja animada através da interação com o painel de controle deve ser estabelecida uma comunicação entre eles. A forma de comunicação escolhida para a animação de cenas *VRML* estática é através do uso da *EAI* (*External Authoring Interface*).

No entanto, para que uma cena *VRML* seja animada através do uso da *EAI* é necessário que os componentes a serem animados sejam nomeados e por isso é que o primeiro nível desenvolvido é dedicado à implementação de um analisador léxico *VRML* associado a um gerador de código *VRML* nomeado através da primitiva *VRML DEF*.

O nível seguinte é responsável por estabelecer quais componentes da cena *VRML* poderão ser animados. Para tanto, foi desenvolvido um painel de controle através do qual o animador pode selecionar os componentes que deseja animar.

O terceiro nível é encarregado de gerar automaticamente uma página *HTML* e um painel de controle que serão carregados e manipulados pelo usuário final na criação da animação.

Após o usuário final criar uma animação através da sua interação com o painel de controle contido na página *HTML,* o último nível do ambiente fará a associação dos dados fornecidos para a animação pelo usuário final com a cena *VRML*. Esta associação realizada através da *EAI* é responsável por possibilitar a visualização da animação.

3.4. Considerações finais

Esse capítulo introduziu e descreveu o ambiente para construção de animações interativas, expôs suas funcionalidades e a metodologia seguida para o seu desenvolvimento.

A descrição de como o ambiente foi implementado e de como deve ser utilizado é realizada no Capítulo 4.

Capítulo 4.

Ambiente implementado

Este capítulo tem como objetivo descrever como o ambiente para a construção de animações interativas é disponibilizado aos usuários e como foi implementado.

Nesta seção será feita uma breve exposição de como foram desenvolvidas as duas etapas do ambiente (ambiente do usuário animador e ambiente do usuário final) com base em um exemplo simples de criação de uma animação interativa.

Quanto ao modo de operação do sistema este será descrito através das interfaces gráficas do ambiente.

4.1. Exemplo motivador

O exemplo utilizado para auxiliar a descrição do ambiente é uma cena *VRML* com quatro objetos geométricos (um cubo, uma esfera, um cone e um cilindro) sobre um plano.

Para que uma cena *VRML* seja animada através da utilização da *EAI* é necessário que os nós *VRML* a serem animados estejam definidos através do uso do *DEF*. Caso a cena *VRML* não possua esta característica a animação não poderá ser realizada.

Prevendo o problema descrito acima o ambiente implementado nomeia os nós *VRML* de forma automática e sem a intervenção do usuário.

Para demonstrar esta característica do ambiente o exemplo a seguir apresenta a cena *VRML* inicial onde os nós *Transform* e *Material* referentes ao nó geométrico *Box* não estão nomeados. O código fonte resultante é apresentado abaixo e sua cena (*Scene.wrl*) pode ser visualizada através da Figura 4.1.

```
#VRML V2.0 utf8

Transform {
      children Shape {
            appearance Appearance {
```

```
                material Material {
                ambientIntensity 0.0984762
                diffuseColor     0.797872 0.124668 0.124668
          specularColor    0 0 0
                emissiveColor    0 0 0
                shininess        0.157143
                transparency     0
          }
     }
     geometry    Box {}
  }
  translation -8 0 0
}

DEF Sphere_Green Transform {
  children Shape {
     appearance Appearance {
          material DEF Material_Sphere Material {
                ambientIntensity 0.283774
                diffuseColor     0.0846193 0.56383 0.0595097
                specularColor    0.13092 0.87234 0.0920716
                emissiveColor    0 0 0
                shininess        0.2
                transparency     0
          }
     }
     geometry    Sphere {}
  }
     translation -4 0 0
}

DEF Cone_Blue Transform {
  children Shape {
     appearance Appearance {
          material DEF Material_Cone Material {
                ambientIntensity 0.306939
                diffuseColor     0.0316982 0.159593 0.521277
                specularColor    0 0 0
                emissiveColor    0.0608088 0.306157 1
                shininess        0.2
                transparency     0
          }
     }
     geometry Cone {}
  }
  translation 0 0 0
}

DEF Cylinder_Yellow Transform {
  children Shape {
     appearance Appearance {
          material DEF Material_Cylinder Material {
                ambientIntensity 0.2
                diffuseColor     0.8 0.757982 0.224457
                specularColor    0 0 0
                emissiveColor    0 0 0
                shininess        0.2
                transparency     0
```

```
            }
      }
      geometry    Cylinder {}
  }
  translation 4  0  0
}

DEF FloorGroup Group {
  children [
      DEF FloorShape Shape {
            appearance Appearance {
                  material DEF FloorColor Material {
                        diffuseColor          0 0 0
                        specularColor         0 0 0
                        ambientIntensity      0
                        emissiveColor         0.2 0.2 0.4
                  }
            }
            geometry DEF FloorIFS IndexedFaceSet {
                  coord DEF FloorCoordinates Coordinate {
                        point [ -50 -1 -50,
                              50 -1 -50,
                              -50 -1 50,
                              50 -1 50 ]
                  }
                  solid            TRUE
                  creaseAngle      0.5
                  coordIndex       [ 0, 2, 1, -1, 1, 2, 3, -1 ]
                  colorIndex       [ 0, 0, 0, -1, 0, 0, 0, -1 ]
            }
      }
  ]
}

DEF Entry Viewpoint {
  position    0 1 15
  orientation 0 1 0  0.1
  description "Entry"
}

DEF Spy ProximitySensor {
  center      0 0 0
  size        1000 1000 1000
  enabled     TRUE
}
```

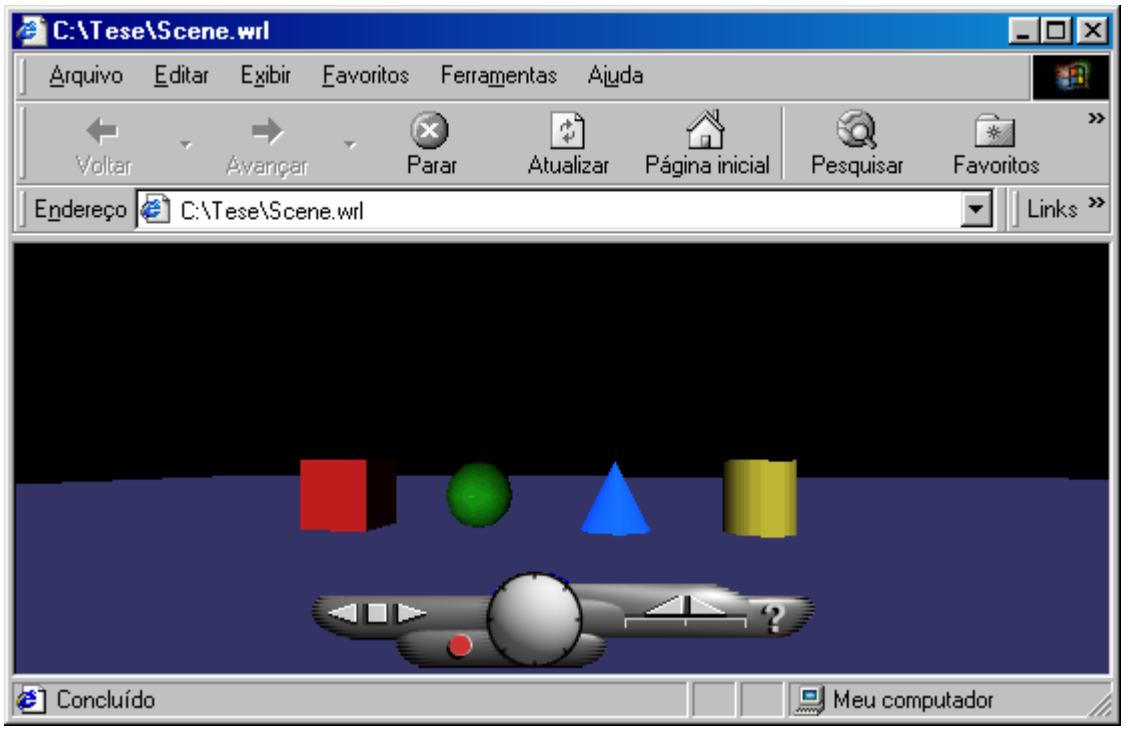

Figura 4.1: Cena *VRML (Scene.wrl)*

O ambiente desenvolvido para geração de animações interativas é subdividido em duas etapas.

A primeira etapa do ambiente é responsável por gerar uma cena *VRML* nomeada semelhante à fornecida como entrada pelo usuário, selecionar os componentes que serão animados e definir a duração da animação (ambiente para o usuário animador). A segunda etapa é responsável pela parte artística da animação (ambiente para o usuário final). Todo o ambiente foi desenvolvido utilizando-se componentes *AWT* do *JDK1.2* sendo a primeira etapa desenvolvida na forma de uma aplicação *Java* e a segunda na forma de um *applet*.

4.2. Ambiente para o usuário animador

Esta etapa do ambiente para geração de animações interativas foi criada com o objetivo de tornar mais simples e automática a forma de criação da animação para a cena *VRML*. O processo envolvido nesta primeira etapa é a seguir descrito.

A criação da animação inicia-se com a execução da aplicação *Java* que corresponde à etapa de interação do usuário animador.

No exemplo a ser descrito uma cena *VRML* estática capturada através da *Web* foi fornecida ao ambiente (como pode ser visto na Figura 4.2) e baseada nela todo o processo de criação da animação descrito na Figura 3.4 é realizado.

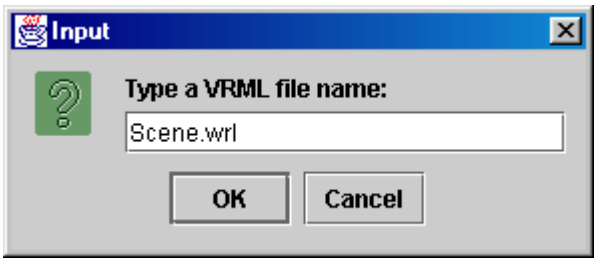

Figura 4.2: Arquivo *VRML* que se deseja animar

O arquivo *VRML* fornecido ao ambiente é analisado (análise léxica) de acordo com a sintaxe *VRML* e seus nós não nomeados são definidos através do uso de *DEF*. Isto é uma exigência para o funcionamento da *EAI* que será utilizada mais a frente.

Para que o arquivo de entrada não sofra nenhuma alteração um outro arquivo *VRML* chamado de *VRMLScene* é gerado e as inclusões de *DEF* são nele feitas. Todo esse processo de preservação do arquivo de entrada e nomeação dos nós é realizado de forma totalmente transparente ao usuário animador através da classe *VRMLAnalise*.

A cena *VRML* (Scene.wrl) descrita anteriormente na Seção 4.1 é substituída pelo arquivo *VRMLScene.wrl* onde todos os seus componentes estão nomeados. Isto

pode ser visto através da comparação do código da cena apresentado anteriormente e do trecho de código a seguir.

```
#VRML V2.0 utf8

DEF Transform1 Transform {
      children DEF Shape2 Shape {
            appearance DEF Appearance3 Appearance {
                  material DEF Material4 Material {
                        ambientIntensity  0.0984762
                        diffuseColor      0.797872 0.124668 0.124668
                        specularColor     0 0 0
                        emissiveColor     0 0 0
                        shininess         0.157143
                        transparency      0
                  }
            }
            geometry DEF Box5 Box {}
      }
      translation -8 0 0
}

DEF Sphere_Green Transform {
```

O nome dado aos nós *VRML*, como pode ser visto no trecho de código acima, é uma junção do nome do nó *VRML* com um número inteiro.

O próximo passo após o usuário animador ter entrado com o nome da cena *VRML* é realizar uma seleção de quais componentes da cena ele deseja animar. Esta seleção é realizada através da interface mostrada na Figura 4.3. No ambiente desenvolvido optou-se por restringir os tipos de componentes *VRML* a serem animados sendo listados para possível seleção apenas os componentes *Transform* (correspondente às animações de alteração de escala e movimento de translação) e *Material* (correspondente à alteração de cores).

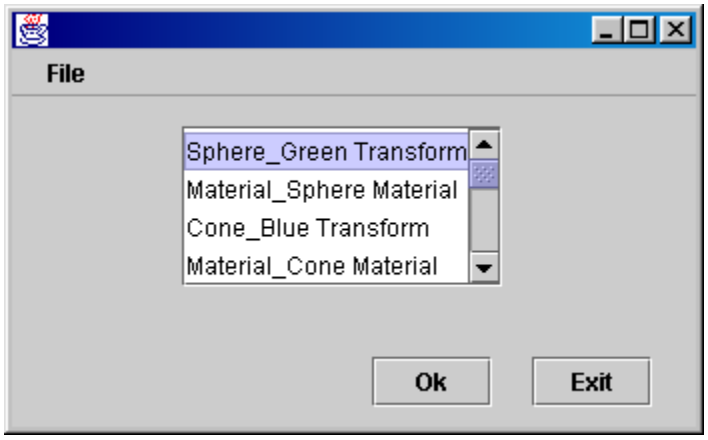

Figura 4.3: Interface para seleção de componentes a serem animados.

Ao terminar a seleção dos componentes da cena *VRML* e pressionar o botão *Exit* o tempo de duração da animação será solicitado através da interface exibida na Figura 4.4. A duração da animação é o valor de entrada do usuário animador multiplicado pelo valor de 300 milisegundos (valor estabelecido neste trabalho) de forma a possibilitar que este consiga visualizar a animação.

O método *sleep*( ) da classe *Thread* é responsável por fazer o *applet* dar uma pausa; sem este método o *applet* seria executado de forma ininterrupta e a animação não seria visualizada corretamente.

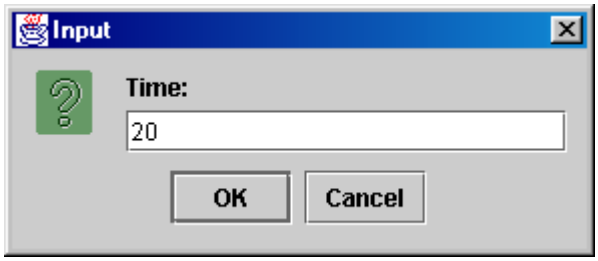

Figura 4.4: Duração da animação.

Na seqüência, o painel de controle (*applet* – *Animation.java*) e a página *HTML* que serão utilizados na segunda etapa do ambiente são gerados automaticamente sem exigir a intervenção do usuário animador. A classe *MakeApplet* implementada gera o painel de controle e a *HTMLMake* a página *HTML*.

Como pode ser visto em negrito no código fonte da página *HTML* gerada (Figura 4.5) a cena *VRML* e o painel de controle criados anteriormente são referenciados. No entanto, é necessário que o usuário animador compile o arquivo *Animation.java* para gerar o arquivo *Animation.class* realizando assim, de acordo com a Figura 3.4, a última tarefa desta etapa do ambiente.

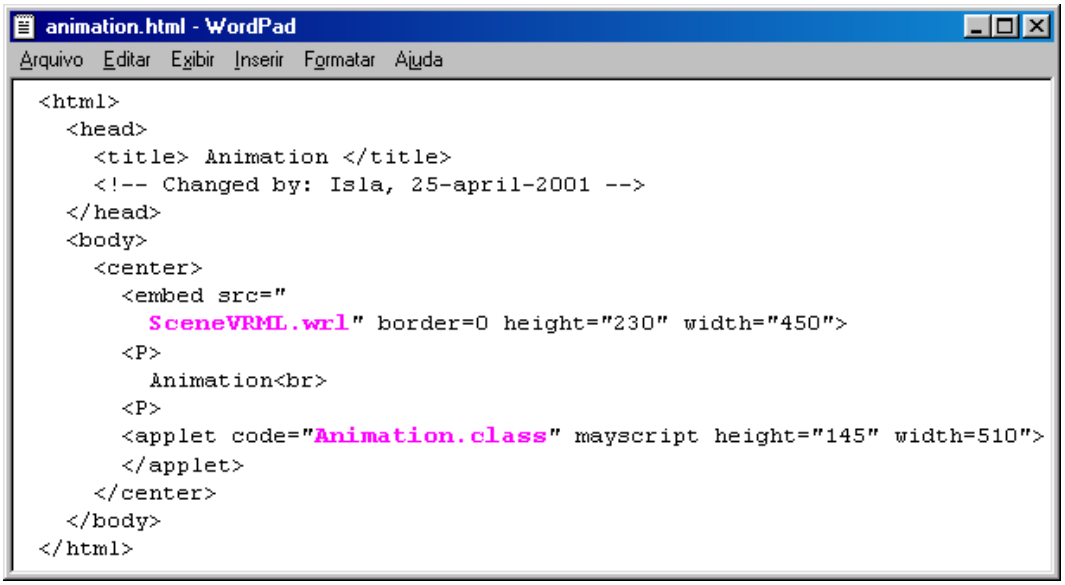

Figura 4.5: Página *HTML*

4.3. Ambiente para o usuário final

O usuário final é qualquer pessoa que queira colocar em prática suas idéias artísticas para elaborar uma animação a partir da definição do usuário animador.

Este usuário deverá carregar em um *browser* a página *HTML* (animation.html) e iniciar a nova etapa no processo de criação da animação. A página referida acima pode ser vista através das Figuras 4.6 e 4.7.

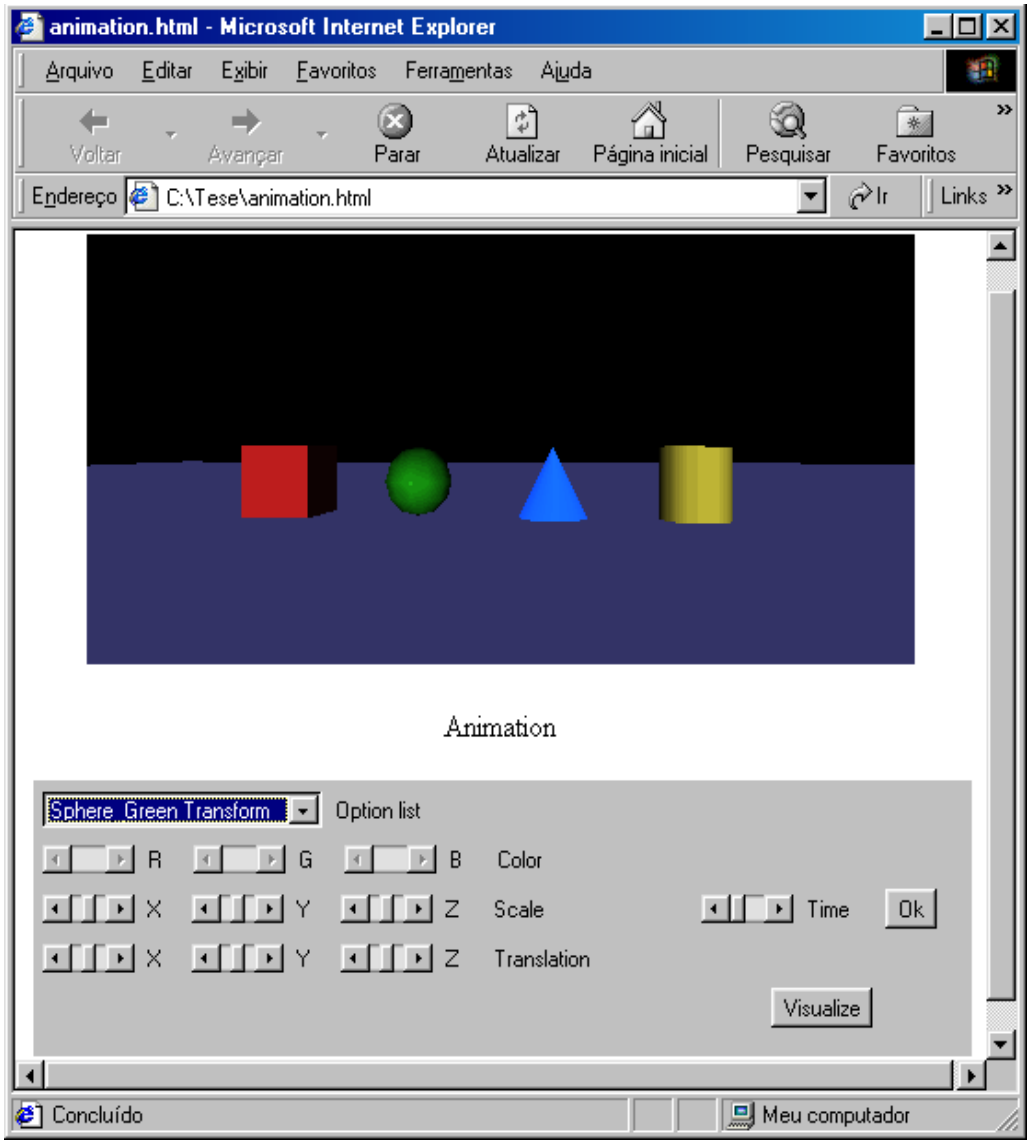

Figura 4.6: Página *HTML* – Ambiente do usuário final (nó *Transform* habilitado)

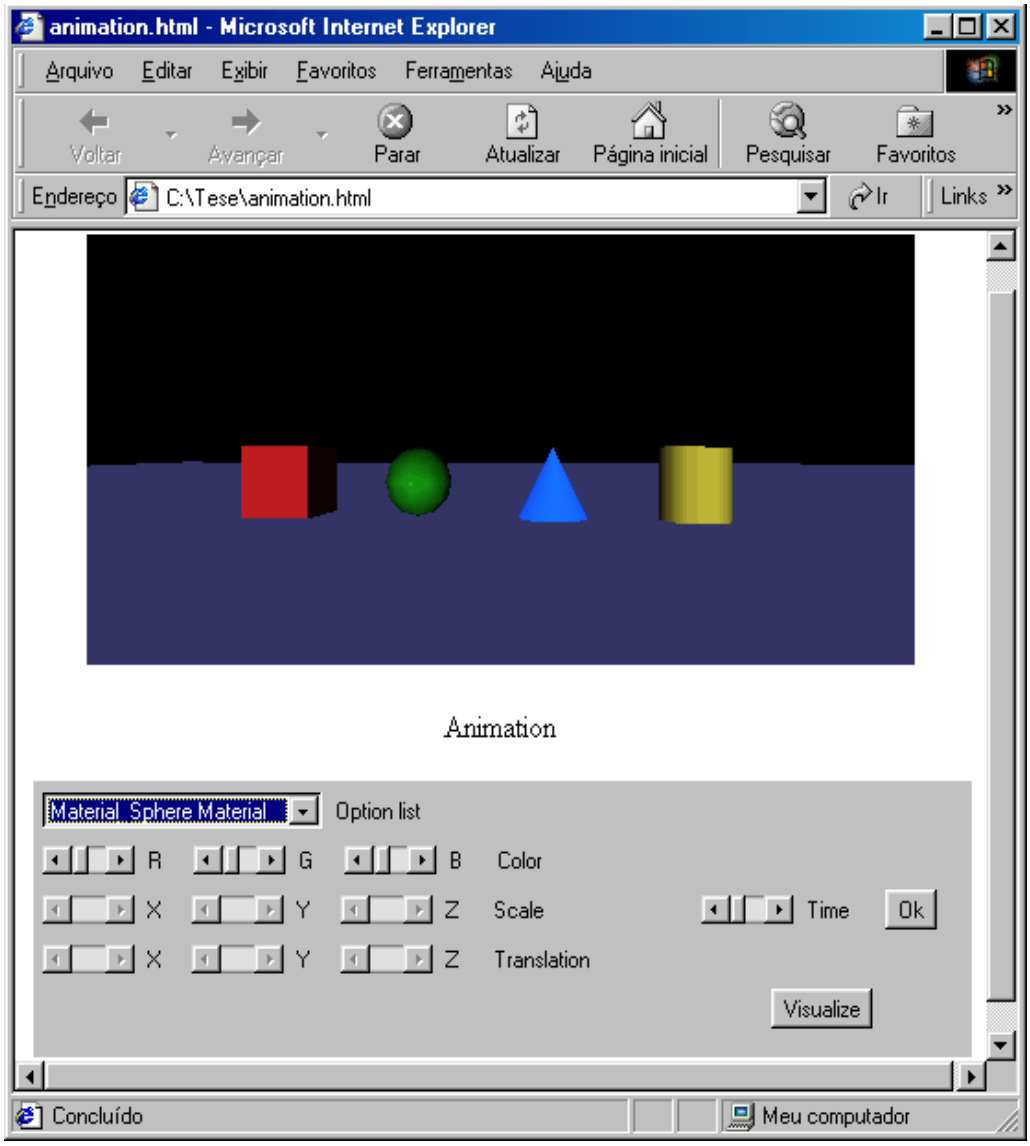

Figura 4.7: Página *HTML* – Ambiente do usuário final (nó *Material* habilitado)

O próximo passo após carregar a página *HTML* é selecionar na lista de opções (*Option List*) um componente da cena a ser animado. Esta lista de opções corresponde aos itens selecionados na primeira etapa do ambiente.

Por exemplo, vamos supor que o usuário animador tenha selecionado todos os itens da lista de seleção na primeira etapa deste ambiente. Então, a lista de opções

(*Option List*) contida na página HTML conterá os seguintes itens: *Sphere_Green Transform, Material_Sphere Material, Cone_Blue Transform, Material_Cone Material, Cylinder_Yellow Transform, Material_Cylinder Material, FloorColor Material, Transform1 Transform* e *Material4 Material.*

Se o componente selecionado pelo usuário final for do tipo *Transform* (ex: *Sphere_Green Transform*) as animações que poderão ser a ele atribuídas serão de dois tipos:

- Mudança de escala - aumentar ou diminuir o objeto nos eixos X, Y e Z.
- Translação - deslocamento do objeto para cima, para baixo, para o lado direito, para o lado esquerdo, para frente ou para trás (X, Y e Z).

Neste caso, a parte do painel de controle correspondente a animação de cores ficará desabilitada (Figura 4.6).

Porém, se o componente selecionado for do tipo *Material* a única animação possível será a de mudança de cores (Figura 4.7).

Cada nova posição, escala ou cor de um objeto deve ser atribuída a um instante de tempo. O tempo irá variar de zero ao tempo total da animação menos um (20 – 1 = 19, por exemplo). E cada atribuição de valor só será efetuada se o botão *"Ok"* for acionado.

Ao final de todas as atribuições (montagem da animação) ou caso se deseje verificar o estado atual da animação, o usuário deverá acionar no botão *Visualize*. Ao acionar no botão *Vizualize* os valores atribuídos aos componentes da cena VRML são interpolados linearmente gerando a animação. Esta interpolação tem por base os valores e intervalos de tempo atribuídos aos componentes.

A seguir um exemplo de atribuição de valores é mostrado:

Instante de Tempo (t)	Componentes da cor:		
	R	G	B
0	1	0	0
7	0	1	0
15	0	0	1

Tabela 4.1: Atribuições de valores para animação

Como pode ser visto na Tabela 4.1 acima, o usuário final seleciona o item *Material_Sphere Material* e faz várias atribuições de cor.

O resultado da animação será a mudança de cor da esfera contida na cena *VRML*. A esfera terá cor vermelha no instante zero, cor verde no instante sete e cor azul a partir do instante quinze. Nos demais instantes de tempo a esfera possuirá cores intermediárias (interpolação).

Na animação exemplificada anteriormente, as cores dos objetos da cena *VRML* são alteradas passando a possuir os novos valores fornecidos pelo usuário. Estas alterações de cor são realizadas através do uso da *EAI* (interface responsável pela comunicação entre o painel de controle e a cena *VRML*).

4.4. Considerações finais

Esse capítulo descreveu através de um exemplo a forma de implementação do ambiente desenvolvido. Os papéis do usuário animador e do usuário final foram descritos detalhadamente.

O último capítulo é dedicado a exposição das conclusões obtidas ao término do desenvolvimento do trabalho e a sugestão de trabalhos futuros que possam contribuir para a melhoria do ambiente.

Capítulo 5.

Conclusões

O objetivo deste trabalho foi possibilitar a criação de animações interativas através da *Web*. Este objetivo foi atingido através do desenvolvimento de um ambiente interativo para geração de animações interativas, apresentado ao longo dos capítulos anteriores.

As funcionalidades do ambiente expostas na Seção 3.2 foram disponibilizadas ao usuário para facilitar a forma de acesso e manipulação do ambiente.

A forma simples de interação do usuário com este ambiente pode ser certificada através dos Capítulos 3 e 4 onde a definição e implementação do ambiente são descritas.

Neste capítulo são descritas as contribuições trazidas pelo ambiente para as animações interativas e trabalhos futuros são propostos visando tornar o ambiente mais completo e versátil.

5.1 Contribuições

O ambiente desenvolvido visa facilitar a construção de animações interativas que envolvem o uso de conhecimentos de programação *Java* e *VRML*. Apenas a primeira etapa deste ambiente necessita da manipulação de um usuário com conhecimentos sobre a linguagem *VRML*.

VRML é a linguagem de programação utilizada para a modelagem da cena gráfica e que através da *EAI* possibilita a associação da cena *VRML* estática com o *Applet Java* criando uma animação.

Java é a linguagem de programação orientada a objetos usada neste trabalho para desenvolver as interfaces gráficas do ambiente e estabelecer a comunicação entre a cena *VRML* e o painel de controle.

A interface final do ambiente desenvolvido para a criação de animações interativas é uma página *HTML* contendo a cena *VRML* e o painel de controle através do qual o usuário final interagirá elaborando o trabalho artístico de criação da animação. A utilização do formato *HTML* possibilita que o ambiente final para criação e visualização da animação seja facilmente acessível aos usuários através da *Web*.

Uma outra característica do sistema que facilita a sua difusão através da *Web* é o fato de que o ambiente desenvolvido torna desnecessário ao usuário final o conhecimento das linguagens de programação *Java* e *VRML*. Esta última característica apresentada tem grande importância, pois o estilo de usuários que poderão acessar o ambiente é muito variado podendo grande parte deles não ter conhecimento algum sobre as linguagens acima mencionadas.

A utilização da *EAI* para animar uma cena *VRML* estática só se torna possível quando os nós da cena a ser animada são definidos através de *DEF*. Este processo de definição dos nós da cena *VRML* é realizado na primeira etapa do ambiente de forma transparente ao usuário animador. Outro ponto positivo neste trabalho é o fato de que a cena original fornecida pelo usuário permanece intacta, pois todas as alterações são realizadas em uma cópia da cena de entrada.

O estudo da tecnologia *VRML* utilizada no desenvolvimento deste trabalho foi publicado no tutorial apresentado no SIBGRAPI'2000 [RAPOSO, 2000a] e na Revista Rita [RAPOSO, 2000b].

Concluindo, o processo de criação de uma animação interativa através do ambiente desenvolvido concentra-se na geração de uma nova cena *VRML* aos moldes da *EAI*, na determinação dos componentes que poderão ser animados na cena e na geração de um painel de controle através do qual o usuário fará a atribuição das alterações que irão compor a animação. Comparando com o esforço que um

animador despenderia efetuando todo o processo de construção de uma animação interativa, a utilização do ambiente desenvolvido é muito vantajoso, pois o animador é poupado em tempo e conhecimento desnecessário, além do fato de possibilitar que usuários comuns da *Internet* utilizem o ambiente.

5.2 Trabalhos futuros

As sugestões de trabalhos futuros feitas a seguir têm como objetivo tornar este ambiente mais versátil, incentivando sua utilização.

Durante o desenvolvimento do trabalho a grande preocupação foi obter uma animação interativa a partir de uma cena *VRML*. Tendo conseguido isto, um passo a frente ao trabalho que seria de muita utilidade ao ambiente é a construção de um módulo de armazenamento para as animações criadas. O armazenamento das animações possibilitaria que animações complexas fossem construídas em várias etapas.

A forma de interação do usuário com o ambiente desenvolvido é muito simples mas, em se tratando de um ambiente que será disponibilizado através da *Web* seria muito útil a implementação de um auxílio ao usuário. Esta extensão do ambiente poderia ser composta de um manual de utilização do ambiente e de um *help* que poderia estar sendo acessado durante todo o processo de criação da animação. Também seria de considerável contribuição ao trabalho a construção de um *site* com exemplos de animações para divulgação do mesmo. E para facilitar ainda mais a utilização do ambiente seria muito interessante possibilitar a interação do usuário, com os próprios objetos da cena, na construção da animação.

Para que caminhos que possam levar o ambiente a casos de erros possam ser evitados seria importante a implementação de um sistema para validação dos dados de entrada. A validação na entrada dos dados é um trabalho rápido que poderia tornar o ambiente mais estável eliminando situações de erro.

Apêndice

Descrição da Implementação

do Ambiente

Descrição da implementação do ambiente

O objetivo deste apêndice é dar uma visão detalhada de como foi implementado o ambiente para criação de animações interativas para *Web*.

A exposição de como foi implementado o ambiente será feita com o auxílio das Figuras A.1, A.2 e 3.4.

1. Principais classes implementadas

A exposição das principais classes implementadas no desenvolvimento deste trabalho será realizada seguindo a ordem estabelecida nos gráficos de execução do sistema mostrados nas Figuras A.1 e A.2.

O uso do ambiente implementado neste trabalho é subdividido em duas fases possuindo um tipo de usuário para cada uma delas (usuário animador e usuário final). Na primeira fase o usuário animador irá interagir com uma cena *VRML* necessitando portanto de conhecimentos sobre esta linguagem. Na fase seguinte o usuário (usuário final) será qualquer pessoa que deseje criar uma animação.

1.1 Primeira etapa do ambiente

A primeira etapa do ambiente foi implementada na forma de uma aplicação *Java* que possui cinco classes principais (Figura A.1):

- Animator
- VRMLAnalysis
- HTMLMake
- Selection
- AppletMake

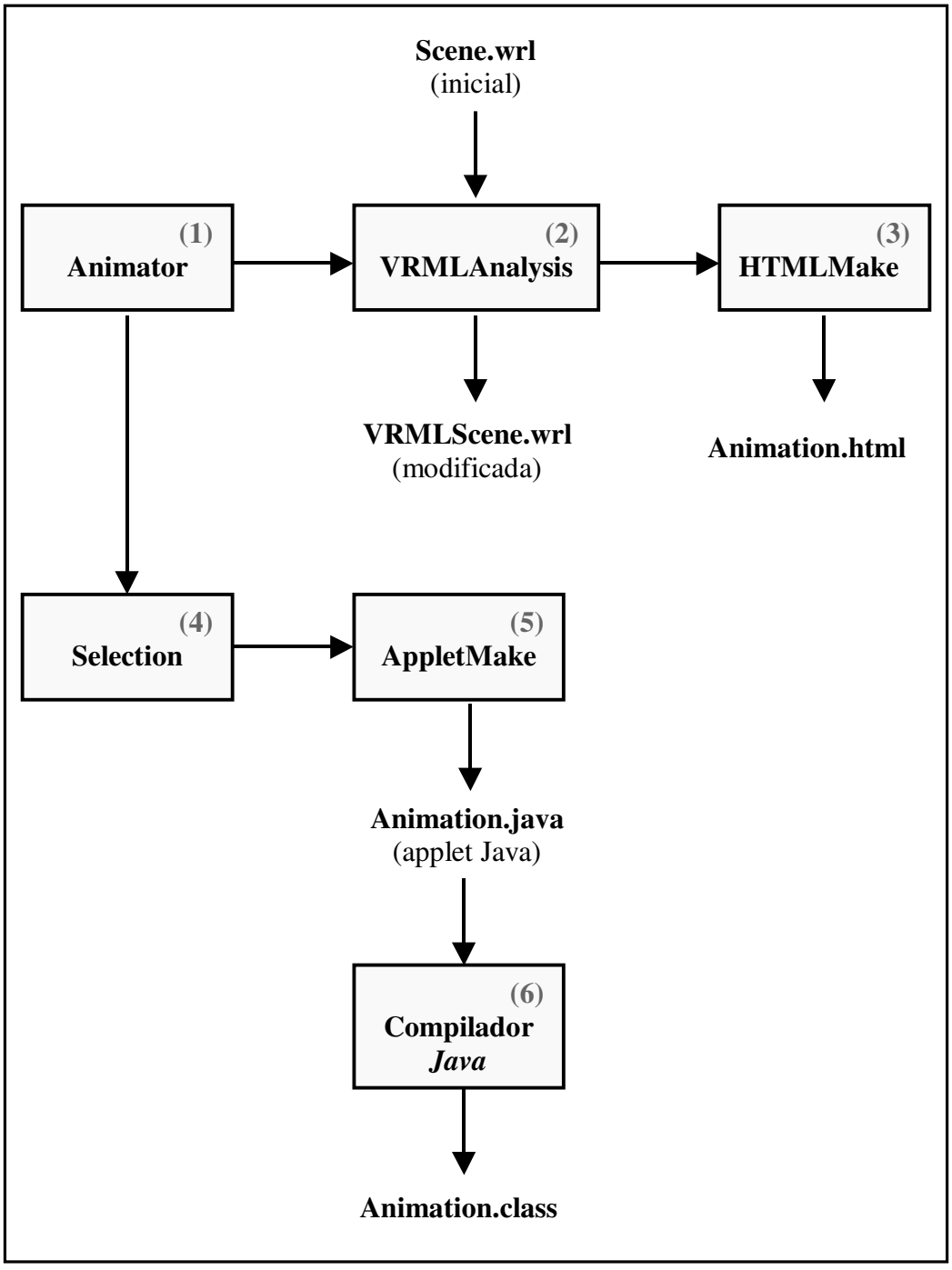

Figura A.1: Primeira etapa do sistema.

Animator

Animator é a classe responsável por iniciar a execução da primeira etapa do sistema (como pode ser visto na figura anterior). Seu código (Animator.java) está descrito logo a baixo:

```
import java.awt.*;
import java.io.*;
     ...
import VRMLAnalysis;

public class Animator extends JFrame {

     public Animator() {

          // Através da execução do método Analyse
          // pertencente a classe VRMLAnalysis o arquivo
          // VRML fornecido como entrada é analisado. Como
          // resultado é gerada uma nova cena VRML cujos nós
          // são definidos através da primitiva VRML DEF.

          System.out.println("VRML analising...");
          VRMLAnalysis win = new VRMLAnalysis();
          win.Analyse();
     }

     public static void main(String[] args) {
          Animator frame = new Animator();
     }
}
```

VRMLAnalysis

VRMLAnalysis é a classe responsável por:

- Analisar a cena *VRML* inicial fornecida como entrada para o ambiente;
- Gerar uma nova cena *VRML* a partir da cena fornecida. Nesta nova cena os nós *VRML* são definidos através da primitiva *DEF*.

Esta etapa do ambiente pode ser visualizada no módulo 2 da Figura A.1 e no módulo 1 da Figura 3.4.

```
class VRMLAnalysis extends JFrame {
```

```
void Analyse () {

     // Solicitação de entrada do nome do arquivo
     // contendo a cena gráfica VRML
     String responde = JOptionPane.showInputDialog (null,
     " Type a VRML file name:");

     try {
          // Nome do arquivo fornecido
          BufferedReader reader = new BufferedReader (new
          FileReader(responde));

          // Novo arquivo criado - SceneVRML
          FileOutputStream outStream = new
          FileOutputStream("SceneVRML.wrl");

          // Lista para armazenagem dos nós que podem //
          ser animados
          List definedNodesList = new List();

          // Analisador de código VRML - espécie de
          // compilador

          // No caso do componente (nó) VRML não estar
          // nomeada a primitiva DEF para este será
          // adicionada ao novo arquivo

          String temp, h;
          StringTokenizer st;
          int tam, i;

          do {
               VRMLAnalysis win = new VRMLAnalysis();

               // Uma linha do código do arquivo que
               // está sendo analisado
               temp = reader.readLine();

               if ( temp != null ) {
                    st = new StringTokenizer(temp);

                    while (st.hasMoreTokens()) {

                      // Uma única palavra do código
                      h = st.nextToken();

                      char ch[] = h.toCharArray();

                      // tamanho da palavra
                      tam = h.length();
```

```
                        // se for um nó - Ex: Transform, //
                        Material
                        // Se não possuir DEF ele será
                        // incluído
                        if ( win.isNode ( h, ch[0] ) ) {
                          mString_ultima = h; //Transform

                          //Transform1
                          mString_nome = h + cont_DEF;
                        // Caso não tenha DEF ele será
                        // colocado
                        if ( tem_DEF == 0 ) {
                          for ( i=0; i<3; ++i)
                            outStream.write  (def.charAt(i));
                              .
                              .
                              .
                    } while (Temp != null);

                    FileOutputStream outStream1 =
                    newFileOutputStream  ("Objects.txt");


                    // Armazenagem no arquivo Objects.txt da
                    // lista de componentes VRML que poderão ser //
                    animados
                              .
                              .
                              .
                    // Execução da classe HTMLMake para geração
                    // da página HTML
                    HTMLMake htmlMak = new HTMLMake(responde);


                    // Fechamento do arquivos
               } // *** fim try
               catch (Exception e) {
                    System.out.println("Error!!!"+ e.toString());
               } // *** fim catch
          }
     }
```

A cena *VRML* modificada (VRMLScene.wrl) gerada no código acima a partir da cena inicial Scene.wrl pode ser vista na Seção 4.2 do Capítulo 4.

O arquivo Objects.txt contém todos os componentes *VRML* que poderão ser animados. Um trecho deste arquivo é a seguir exposto:

Sphere_Green Transform
Material_Sphere Material
Cone_Blue Transform
Material_Cone Material
Cylinder_Yellow Transform
...

HTMLMake

A geração da página *HTML* (módulo 3 da Figura A.1 e módulo 2 da Figura 3.4) é feita através da criação de um arquivo texto contendo o nome do painel de controle (*applet Java* – Animation.java) e da cena *VRML* modificada. O código da página *HTML* resultante pode ser visto na Figura 4.5 do Capítulo 4.

Selection

Esta classe é responsável por permitir ao usuário animador selecionar as componentes *VRML* da cena que poderão ser animadas na segunda fase do ambiente (módulo 4 da Figura A.1 e módulos 3, 4 e 5 da Figura 3.4).

Nesta etapa todos os componentes *VRML* do tipo *Transform* e *Material* armazenados no arquivo Objectos.txt são expostos ao usuário animador através da interface gráfica mostrada na Figura 4.3 do Capítulo 4. E através desta interface o usuário animador faz a seleção dos componentes que permite animar e determina o tempo total de duração da animação.

Por fim, a classe AppletMake é acionada.

AppletMake

Esta classe é responsável por gerar o *applet Java* (painel de controle) e fazer a comunicação deste com a cena *VRML* através do uso da *EAI*. O painel de controle gerado será utilizado na segunda etapa do ambiente.

Porém, antes de ser iniciada a segunda etapa do ambiente, o *applet Java* (Animation.java) deverá ser compilado para que o arquivo Animation.class utilizado na página Animation.html seja gerado.

Esta etapa pode ser visualizada através do módulo 5 da Figura A.1 e do módulo 6 da Figura 3.4.

1.2 Segunda etapa do ambiente

Nesta etapa do ambiente o usuário final interagirá com a página *HTML* fazendo atribuições de valores para cor, escala e translação em relação ao tempo (Figura A.2 e módulo 7 da Figura 3.4). Estes valores são armazenados em uma lista juntamente com os instantes de tempo estabelecidos. Quando concluídas as referidas atribuições definidas pelo usuário final, os valores armazenados são interpolados e a animação visualizada.

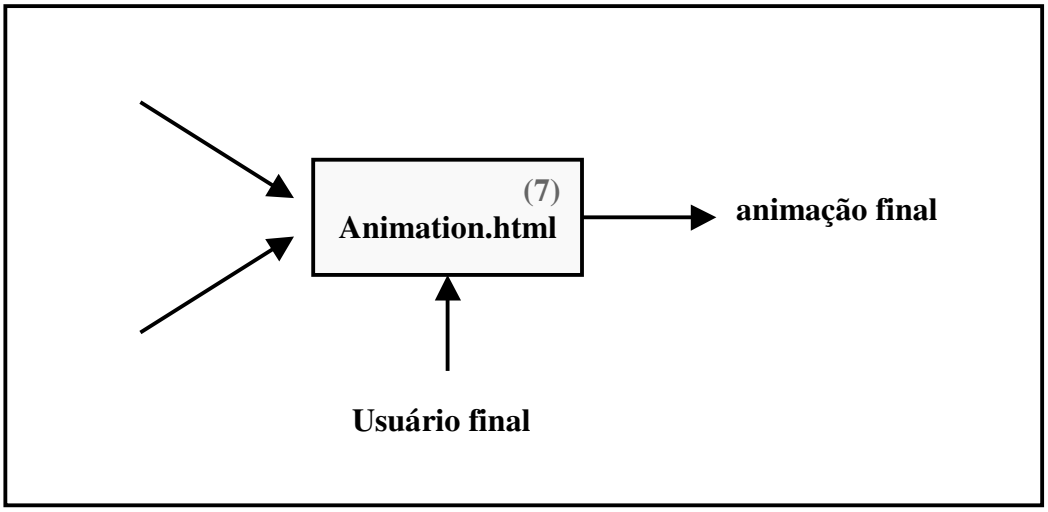

Figura A.2: Segunda etapa do sistema.

A seguir algumas partes do arquivo animation.java são apresentadas.

```
            .
            .
            .
      EventInSFColor diffuseColor = null;
      EventInSFVec3f scale = null;
      EventInSFVec3f translation = null;

      Node Transform0 = null;
      Node Material1 = null;
      EventInSFVec3f scale0 = null;
      EventInSFVec3f translation0 = null;
      EventInSFColor diffuseColor1 = null;

      // referência ao browser
      browser = Browser.getBrowser(this);
      try {
            //referência ao nó Transform Sphere_Green
            Transform0 = browser.getNode("Sphere_Green");


            //escala
            scale0 = (EventInSFVec3f)
            Transform0.getEventIn("set_scale");

            // translação
            translation0 = (EventInSFVec3f)
            Transform0.getEventIn("set_translation");

            //referência ao nó Material
            Material1 = browser.getNode("Material_Sphere");

            //cor
            diffuseColor1 = (EventInSFColor)
            Material1.getEventIn("set_diffuseColor");
      }
      catch (InvalidNodeException ne) {
            add(new TextField("Failed to get node:" + ne));
            error = true;
      }
      catch (InvalidEventInException ee) {
            add(new TextField("Failed to get EventIn:" + ee));
            error = true;
}
            .
            .
            .
public void animacao() {
//animação de cor
      atual_lista = ListaCor.first;//primeiro elemento da lista
      while(atual_lista != null) {
```

```
atual_no = atual_lista.first;
while((atual_no != null) && (fim == 0)) {
     if (atual_no.tempo == k) {
          //setar valor
          val[0] = atual_no.val_aux1; //R
          val[1] = atual_no.val_aux2; //G
          val[2] = atual_no.val_aux3; //B

          //alteração da cor
          diffuseColor1.setValue(val);
.
.
.
```

Referências bibliográficas

[BLAXXUN, 2000a] *Blaxxun Interactive. Blaxxun Contact 5.0* 2000. http://www.blaxxun.com/support/glossary.html

[BLAXXUN, 2000b] *Blaxxun Interactive. Blaxxun Contact 5.0* 2000. http://www.blaxxun.com/products/contact

[COMPUTING, 1999] K. Computing. *Getting Started with Java 3D™ API*, 1999. http://sun.com/desktop/java3d/collateral

[HARTMAN, 1996] J. Hartman and J. Wernecke. *The VRML 2.0 Handbook – Building Moving Worlds on the Web.* Addison Wesley, 1996.

[JAWORSKI, 1999] J. Jaworski. *Java 2 Platform Unleashed* . Sams, 1999.

[LEMAY, 1999] L. Lemay & R. Cadenhead. *Aprenda em 21 dias Java 1.2*. Editora Campus. 1999.

[MACROMEDIA, 2000] http://www.macromedia.com/

[NEIDER, 1996] Jackie Neider et al. *OpenGL Programming Guide.* Addison-Wesley Publishing Company, 1996 .

[RAPOSO, 2000a] A. B. Raposo, A. J. da Cruz, A. L. Bicho, A. K. Kojima, C. A. M. dos Santos, I.C.F. Silva, L. P. Magalhães & P. C. P. de Andrade. *Software Livre para Computação Gráfica e*

Animação por Computador. SIBGRAPI'2000. Gramado. Outubro 2000.

[RAPOSO, 2000b] A. B. Raposo, A. J. da Cruz, A. L. Bicho, A. K. Kojima, C. A. M. dos Santos, I.C.F. Silva, L. P. Magalhães & P. C. P. de Andrade. *Software Livre para Computação Gráfica e Animação por Computador.* Revista RITA – edição especial dos artigos do SIBGRAPI '2000.

[ROEHL, 1997] B. Roehl et al. *Late Night VRML 2.0 with Java.* Ziff-Davis Press, 1997.

[SHELLY, 2000] Gary B. Shelly and Thomas J. Cashman. ***Java Script :*** *Introductory Concepts and Techniques*. 2000.

[SUN, 1998] Sun Microsystems. *Linguagem Java*. 1998. http://java.sun.com

[TAMIOSSO, 1998] F. S. Tamiosso. *Desenvolvimento de um ambiente para construção de animações interativas: combinando VRML e Java*. Universidade Estadual de Campinas (UNICAMP). Dissertação de mestrado. Campinas. Novembro de 1998.

[TAMIOSSO, 1997a] F. S. Tamiosso. *VRML 2.0 - Resumo baseado em exemplos*. 1997. http://www.dca.fee.unicamp.br/~fabiana/vrml/tutorial.html

[TAMIOSSO, 1997b] F. S. Tamiosso, L. P. Magalhães, I. L. M. Ricarte e A. B. Raposo. *Building interactive animations using VRML and*

Java. X Simpósio Brasileiro de Computação Gráfica e Processamento de Imagens. SIBGRAPI'97. pp.42-48. Campos do Jordão, SP. 1997.

[TERMINOLOGY, 2001] *Internationalized Glossary of Java™ Terminology*. 2001. http://192.9.48.9/applets/glossary/index.html

[VRML, 1997] VRML Consortium. *The Virtual Reality Modeling Language Specification ISO/IEC DIS* 14772-1. Abril, 1997. http://www.vrml.org/Specifications/VRML97/DIS/

[WEB, 1997] *Web 3D Consortium. The Virtual Reality Modeling Language. International Standard* ISO/IEC DIS 14772-1. 1997. http://www.web3d.org/technicalinfo/specifications/vrml97/index.htm

[WEB, 1999a] Web 3D Consortium. 1999. http://www.web3d.org

[WEB, 1999b] Web 3D Consortium. *The Virtual Reality Modeling Language – Part 2: External Authoring interface and bindings*. ISSO/IEC 14772-2:xxxx. 1999. http://www.web3d.org/WorkingGroups/vrml-eai/

[WEB, 2000] Web 3D Consortium. *VRML 200x – Draft Specification*. 2000. http://www.web3d.org/TaskGroups/x3d/specification