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

Desenvolvimento de um ambiente para construção de animações interativas: combinando VRML e Java

Autora: **Fabiana Saldanha Tamiosso**
Orientador: **Prof. Dr. Léo Pini Magalhães**
Co-orientador: **Prof. Dr. Ivan Luiz Marques Ricarte**

Tese 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 Automação.

Campinas, dezembro de 1998.

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

T152d

Tamiosso, Fabiana Saldanha
Desenvolvimento de um ambiente para construção de amimações interativas: combinando VRML e Java. / Fabiana Saldanha Tamiosso.--Campinas, SP: [s.n.], 1998.

Orientadores: Léo Pini Magalhães, Ivan Luiz Marques Ricarte.
Dissertação (mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Animação por computador. 2. Interfaces gráficas de usuário (Sistema de computador). 3. Interfaces (Computador). I. Magalhães, Léo Pini. II. Ricarte, Ivan Luiz Marques. III. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. IV. Título.

Banca da dissertação

Apresentação: 14 de dezembro de 1998.

Banca:

Orientador: **Prof. Dr. Léo Pini Magalhães**

Membro interno ao programa: **Profª. Dra. Beatriz Mascia Daltrini**

Membro externo ao programa: **Profª. Dra. Carla Maria Dal Sasso Freitas**

Membro convidado: **Prof. Dr. Ivan Luiz Marques Ricarte**

Sumário

Lista de figuras

Lista de tabelas

Resumo

O desenvolvimento de um ambiente para a construção de animações interativas é o tema deste trabalho. Esse ambiente possibilita que, de uma forma fácil e com informações simples, uma animação seja obtida incluindo um tipo de movimento e capacidade de interação com o usuário.

Animações interativas são definidas por terem seu comportamento determinado por ações do usuário. As animações, construídas através do ambiente desenvolvido, apresentam a interatividade caracterizada no controle de parâmetros envolvidos na determinação do movimento. Neste tipo de animação o usuário interage com uma interface gráfica, associada à cena animada, de onde controla o movimento.

VRML (*Virtual Reality Modeling Language*) é a linguagem utilizada neste trabalho para a definição de animações interativas. VRML, entre outras características, associa capacidade de processamento às cenas, tornando-as mais "inteligentes". Esta é a característica que torna possível a combinação de VRML com a linguagem Java e, é através desta combinação, que as animações interativas são obtidas.

O processo para criar animações interativas pode ser complexo, envolvendo conhecimento de técnicas de modelagem e tecnologias de programação. O ambiente foi desenvolvido para facilitar este processo para os animadores, resumindo-o à definição de movimento e a sua associação a um objeto de uma cena VRML. Diversas funcionalidades foram disponibilizadas buscando a versatilidade na utilização do ambiente e atendendo às necessidades de um número razoável de aplicações. Dentre as principais características implementadas estão uma biblioteca, contendo vários movimentos pré-definidos, e a possibilidade de reutilização de movimentos criados pelo animador.

Abstract

The development of an environment for interactive animations building is the subject of this thesis. This environment allows that, in an easy way and with simple information, an animation is created including a kind of movement and user's interaction capacity.

Interactive animations are defined for having their behaviour controlled by user's actions. The animations, built by the developed environment, present the interactivity based on the control of parameters used in movement determination. In this kind of animation the user interacts with a graphical interface, linked with the animated scene, from where he controls the movement.

VRML (*Virtual Reality Modeling Language*) is the language used in this thesis to define interactive animations. Among other features, it links processing capacity to the scenes, making them more "intelligent". This is the feature which makes possible the combination between VRML and Java language and, by this combination, the interactive animations are gotten.

The process of interactive animations conception may be complex, involving knowledge of modeling techniques and programming technologies. The environment was developed to turn the process easier to the animators, summarizing it to the definition of movement and its association to a VRML scene object. Many funcionalities are supported, in order to offer versatility in the environment exploration and reaching the exigences of a considerable number of applications. Among the main implemented features, there are a library, with many pre-defined movements, and the possibility of reusing movements created by the animator.

Capítulo 1.

Introdução

Animações modeladas por computador são exploradas neste trabalho, especialmente as animações interativas, que têm seu comportamento determinado por ações do usuário.

Diversas áreas, relacionadas a criação de simulações, estudos científicos ou educacionais, podem se beneficiar de animações interativas. Entretanto, o processo para criar este tipo de animação pode ser complexo e inviável para usuários que não dominam técnicas de modelagem ou tecnologias de programação. Sistemas que possibilitam ou facilitam a criação de animações interativas não são ainda popularizados.

Uma linguagem para modelagem de animações que tem sido muito explorada é VRML (*Virtual Reality Modeling Language*), pois oferece muitas potencialidades, tais como: realismo às cenas, capacidade de interação, um conjunto de animações pré-definidas e a introdução de inteligência às cenas. Uma importante característica das cenas VRML, que contribuiu para a disseminação da linguagem, é a capacidade de visualização através da Internet. Desta maneira é possível que uma animação seja manipulada em um ambiente corporativo (através da Internet).

VRML é utilizada neste trabalho para a definição de animações interativas. VRML oferece a capacidade de interação através de sensores que detectam ações do usuário. Entretanto, a interação pretendida é mais complexa; o objetivo é permitir que o usuário controle a animação através da definição de valores para os parâmetros usados nas equações responsáveis pelo movimento. Isto é possível através da associação de um *script* à cena VRML, escrito em linguagem Java e provido de comunicação com a cena. A responsabilidade da geração do movimento é atribuída a ele. Como o *script* é um programa Java, há a possibilidade de comunicação com uma outra aplicação que faz a interface com o usuário. É através desta aplicação que o usuário controla a animação.

O processo para construir este tipo de animação interativa consiste em definir a cena VRML, programar o *script* Java envolvendo as equações para o cálculo do movimento, definir uma outra aplicação Java responsável pela interação com o usuário, possibilitando a comunicação com o *script*. Isto exige que o usuário domine alguns tipos de movimento (o que envolve a definição de equações), tenha conhecimento avançado de VRML e experiência com programação Java, o que não é muito comum entre os animadores.

A fim de facilitar este processo e evitar que um animador dispense esforços com a programação Java, um ambiente para a construção de animações interativas foi desenvolvido. Esse ambiente possibilita que, de uma forma fácil, com informações simples, uma cena VRML estática seja associada a um tipo de movimento com capacidade de interação do usuário. O desenvolvimento deste ambiente e suas possibilidades de utilização são descritas neste trabalho.

O Capítulo 2 introduz as animações interativas geradas pelo ambiente desenvolvido. Algumas técnicas para modelagem de movimentos são apresentadas e animações interativas são definidas. Para a modelagem e visualização das animações, VRML é apresentada através de uma rápida descrição dos principais aspectos deste paradigma. Animações que podem ser geradas com a utilização de VRML, envolvendo a capacidade de interação, são exemplificadas. Para possibilitar maior grau de interação, as tecnologias de VRML e Java são integradas. Detalhes sobre o mecanismo desta integração são apresentados. A combinação de VRML e Java possibilita a utilização de vários tipos de movimentos e maior interação do usuário na animação. Este tipo de animação é exemplificado, concluindo o capítulo.

O usuário do ambiente é o animador, que deve informar uma cena VRML que deseja animar. O ambiente identifica os objetos contidos na cena e permite seleção. O objeto selecionado será associado a um tipo de movimento, promovendo a animação. O ambiente oferece alguns tipos de movimentos pré-definidos em uma biblioteca e, para utilizá-los, o animador não precisa ter conhecimento de suas equações, basta escolher um deles e a animação interativa é gerada automaticamente. Entretanto, como a biblioteca é restrita a algumas aplicações, o ambiente também possibilita a definição de um novo movimento, onde o animador deve informar as equações que o representam. Outras funcionalidades relacionadas a definição de um novo movimento são providas

pelo ambiente a fim de facilitar a interação com o animador e proporcionar maior versatilidade de utilização. Elas são apresentadas no Capítulo 3.

Definidas as animações que o ambiente deve gerar (no Capítulo 2), o Capítulo 3 mostra como o ambiente é especificado para suprir este objetivo. A motivação para a construção do ambiente é apresentada com base na animação que o Capítulo 2 introduziu. O ambiente é descrito como uma "caixa preta", onde a entrada esperada e a saída gerada são definidas. Os requisitos que o ambiente deve obedecer são determinados. Toda a funcionalidade do ambiente é descrita, mostrando os módulos que compõem a "caixa preta" citada. A arquitetura é apresentada em módulos onde as funcionalidades são agrupadas. Uma metodologia para desenvolvimento em níveis é descrita e utilizada como base à implementação que é apresentada no Capítulo 4.

No Capítulo 4, seguindo a metodologia de desenvolvimento, a implementação do ambiente é apresentada. As classes modeladas são exibidas em diagramas. As interfaces são apresentadas e o modo de operação é descrito com o auxílio de um exemplo que define um tipo de movimento para a criação de uma animação interativa. Desta forma, a funcionalidade do ambiente para a definição de um novo movimento é demonstrada. Além da implementação do ambiente, é mostrado o procedimento para compor a biblioteca de movimentos. A biblioteca é uma funcionalidade disponibilizada pelo ambiente, mas é de responsabilidade do usuário a inclusão de novos movimentos, o que torna o ambiente mais versátil para utilização.

No Capítulo 5 a utilização da biblioteca de movimentos para a definição de animações interativas é explorada. Os movimentos existentes na biblioteca do ambiente são apresentados. A utilização desses movimentos é apresentada para um cenário que mostra a aplicação das animações interativas geradas.

A finalização deste trabalho consiste na apresentação das conclusões obtidas, de algumas sugestões para possíveis extensões que motivam trabalhos futuros e da bibliografia utilizada.

Capítulo 2.

Animações: paradigmas e modelagem

Neste capítulo, o objetivo é introduzir animações modeladas por computador e alguns de seus aspectos correlatos. Em particular, a utilização de VRML (*Virtual Reality Modeling Language*) será descrita como forma de modelagem. Após essa descrição, será visto como obter animações interativas que exploram a combinação de VRML e Java.

Uma introdução sobre animações e seus paradigmas será apresentada na primeira seção. Animações interativas e técnicas para modelagem de movimento serão igualmente descritas.

Em seguida, VRML será apresentada, com a descrição do paradigma e principais nodos. Para a visualização de exemplos e como material complementar, existem tutoriais na Internet que podem ser consultados.

A terceira e última seção deste capítulo discute a integração entre Java e VRML. A linguagem Java será introduzida e suas características relacionadas à VRML serão abordadas.

2.1. Animação

O processo convencional de criação de animações envolve um conjunto de atividades, tais como desenho dos personagens, ações e quadros principais, que, quando automatizadas, proporcionam grande otimização deste processo [CAMARGO, 1995]. Sistemas para modelagem de animações por computador devem oferecer, ao usuário, vantagens visadas por outros tipos de sistema: tempo de desenvolvimento minimizado, baixo custo e boa confiabilidade.

Nas próximas seções, animações serão discutidas seguindo duas diretivas: técnicas para modelagem de movimento e um tipo de animação que envolve interação com o usuário.

2.1.1. Modelagem do movimento

Existem várias técnicas para modelagem de movimento em animação por computador. Modelos dinâmicos e cinemáticos são freqüentemente usados para obter diversos tipos de movimento para objetos em uma animação. Isto é válido para representações realistas, pois as leis físicas modelam a realidade de modo mais fiel. Muitos trabalhos têm sido apresentados para modelagem de movimentos realistas [KALRA, 1992]. O problema com este tipo de modelagem é identificado quando se trata de movimentos mais complexos, que envolvem um grande e complexo conjunto de equações diferenciais.

O modelo dinâmico para a animação de objetos é complementar ao cinemático, envolvendo propriedades físicas adicionais, como por exemplo, torque e força. A utilização de um modelo mais completo apresenta maior grau de realismo no movimento gerado, mas, ao mesmo tempo, insere maior complexidade. O custo computacional exigido, nestes casos, nem sempre é recompensado, pois o resultado visual pode não ser o esperado, visto que as variáveis envolvidas nem sempre podem ser manipuladas de forma intuitiva.

Neste trabalho, será considerada uma classificação para modelagem de movimento, através das abordagens de interpolação, cinemática e dinâmica.

Na técnica de **interpolação** o animador pré-define uma seqüência de quadros e uma função de interpolação ou conjunto de valores interpolados, reponsável pela ligação entre esses quadros definidos. O tempo da animação ou o número de quadros por segundo são informações relevantes e dadas pelo animador. Este é um tipo de animação inteiramente pré-definida. Os trabalhos de Raposo [RAPOSO, 1997b] e Tamiosso [TAMIOSSO, 1997a] exemplificam esta técnica.

Usando as técnicas de **cinemática** e **dinâmica**, o animador pré-define o movimento como função do tempo em um sistema de equações. Animações são construídas pela amostragem de movimento de objetos em instantes de tempo específicos. As questões apresentadas anteriormente, a respeito de realismo *versus*

complexidade, devem ser consideradas quando se trata da escolha da técnica a ser utilizada para a geração da animação de interesse.

Uma outra proposta, apresentada por Isaacs [ISAACS, 1987], classifica os sistemas de animação em: interpolação, procedimental (caracterizado em *Tookima* [RAPOSO, 1997a]) e simulação dinâmica (onde as leis físicas são as ferramentas de modelagem).

2.1.2. Animações interativas

A modelagem do movimento é uma área de muita pesquisa, devido a sua complexidade e importância para a animação. Seguindo outra linha, mas diretamente associada, este trabalho trata de animações onde o usuário deve atuar, tomar decisões e interferir no movimento. Para isso, são utilizadas as três abordagens, interpolação, cinemática e dinâmica, para modelar o movimento de animações interativas.

As animações interativas, aqui tratadas, são aquelas que apresentam comportamento ou movimento capaz de receber e responder a estímulos externos, relativos a ações do usuário. Por exemplo, o usuário, através do *mouse*, pode escolher qual, entre duas trajetórias, um objeto pode seguir.

Para modelar animações interativas, segundo as técnicas de movimento apresentadas, será utilizada VRML como linguagem para definição de cenas animadas. A linguagem apresenta ainda funcionalidades que proporcionam a criação de animações "inteligentes", ou seja, animações às quais podemos associar um processamento. Apesar de VRML possibilitar modelagem de realidade virtual, será utilizada somente como modelador e gerador de animações no âmbito da Internet.

2.2. VRML – Virtual Reality Modeling Language

Virtual Reality Modeling Language (VRML) é uma linguagem para descrição de cenas 3D interativas a serem visualizadas, por exemplo, na *World Wide Web* (WWW). O arquivo que representa a cena é descrito através de nodos, como por exemplo, geométricos, de iluminação, interpoladores, etc. Esta linguagem foi projetada pelo consórcio das companhias Silicon Graphics, Sony Research e Mitra, num esforço conjunto, a fim de se obter uma padronização na descrição de cenas para a *Web*.

VRML 2.0 acrescentou maior capacidade de interação à versão inicial da proposta de padronização, que proporcionava a criação e visualização de cenas 3D estáticas. Além de mais rica e com maior interação, esta nova versão está preparada para futuras expansões mantendo o máximo de simplicidade e rapidez.

As características de VRML 2.0 incluem:

- maior realismo às cenas, tais como a criação de terra-céu, terrenos irregulares, ilusão de distâncias e efeitos atmosféricos;
- interação direta com objetos através de sensores, que determinam eventos relacionados a operações do usuário, como movimento em certas áreas e seleção de alguns objetos. Um outro tipo de sensor mantém relação com o tempo, proporcionando a base para a criação de animações;
- uma variedade de nodos interpoladores que geram animações pré-definidas. A mudança de cores de objetos à medida que se movem e a criação de "*tours*" guiados que movem automaticamente o usuário ao longo de um caminho, são exemplos da utilização destes nodos;
- prototipação, a capacidade para criar bibliotecas e para encapsular vários tipos de nodos em um novo tipo, que pode ser utilizado por várias cenas;
- a criação de animações "inteligentes" através da utilização de *scripts*, programas que recebem entradas de sensores e geram eventos que podem alterar outros nodos.

2.2.1. Nodos, campos e eventos

VRML define um formato de arquivo para descrição de objetos chamados nodos, que contêm dados armazenados em campos e podem definir eventos.

Os nodos têm zero ou mais parâmetros, chamados campos, que os diferenciam de outros nodos do mesmo tipo; por exemplo, o raio de uma esfera. Cada nodo define nome, tipo e valor *default* para seus campos. Há dois tipos de campos: `field`, considerado privado e `exposedField`, público, possibilitando a modificação de seu valor por outro nodo.

VRML define um mecanismo de comunicação entre os nodos baseado em eventos. Os nodos podem enviar ou receber eventos de outros nodos, o que é denotado por `eventOut` e `eventIn`, respectivamente. Isso também pode acontecer com os campos

exposedField, que podem ser considerados como a combinação de field com eventIn e eventOut. Um nodo que produz evento de um determinado tipo pode ser conectado a outro nodo que recebe evento deste mesmo tipo. A conexão entre os nodos é feita através de ROUTE, mecanismo da linguagem responsável pela definição do roteamento de eventos entre os nodos.

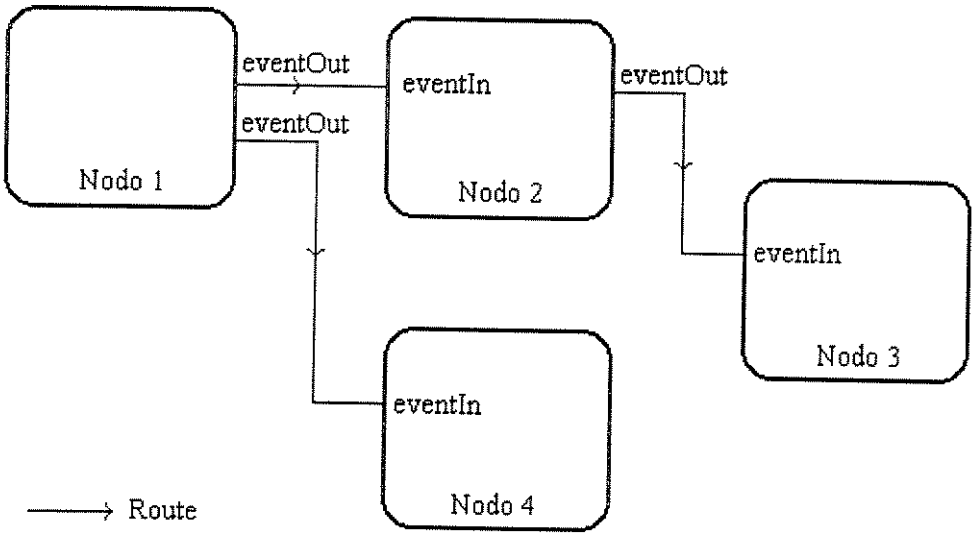

Figura 2.1: Conexão entre nodos VRML

A Figura 2.1 exemplifica a conexão entre quatro nodos, através da interligação de eventOuts e eventIns. O nodo 1 gera dois eventOut que correspondem aos eventIn dos nodos 2 e 4, respectivamente. Isto significa que o nodo 1 envia dados que são atribuídos aos nodos 2 e 4. Por sua vez, o nodo 2, depois de receber o evento, realiza algum processamento e gera um eventOut que é recebido como um eventIn pelo nodo 3.

2.2.2. Estrutura hierárquica

Hierarquias podem ser criadas através de nodos de agrupamento, os quais contêm um campo chamado children, que engloba uma lista de nodos filhos. Cada nodo de agrupamento define um espaço de coordenadas para seus filhos. As operações de transformação são acumuladas na hierarquia de nodos, mapeando as coordenadas locais para o sistema de coordenadas do mundo, onde a cena é exibida.

Os diversos nodos de agrupamento: Anchor, Billboard, Collision, Group e Transform, permitem que alguma operação seja aplicada a um conjunto de nodos, ao

mesmo tempo. Os nodos de agrupamento mais comuns serão brevemente descritos a seguir:

- **Anchor** é um nodo que busca um recurso identificado por um URL (*Uniform Resource Locator*) quando qualquer estrutura geométrica filha deste nodo é ativada;
- **Group** define um sistema de coordenadas para seus filhos relativo ao de seus pais;
- **Transform** é equivalente ao Group; no entanto, apresenta campos usados para transformações geométricas 3D, que são realizadas sobre o sistema de coordenadas definido. Estas transformações são rotação, escala e translação.

2.2.3. Instanciação

Um nodo pode ser referenciado em VRML várias vezes, com a utilização de DEF e USE, semelhantemente ao uso de macros. Com **DEF** é possível atribuir um nome a um nodo para posterior utilização junto a **USE**, tantas vezes quantas forem necessário. Quando a definição do nodo é modificada todas as referências a ele são alteradas. O escopo da definição é restrito a um arquivo.

2.2.4. Prototipação

O mecanismo de prototipagem permite, através da primitiva **PROTO**, a definição de outros tipos de nodos, encapsulando e parametrizando geometria, atributos e comportamentos, estendendo e customizando as primitivas VRML no âmbito de uma aplicação.

Para permitir a criação de bibliotecas, como por exemplo de cores ou formas, é utilizada a primitiva **EXTERNPROTO** para mencionar o arquivo (biblioteca) onde protótipos estão definidos. Essa é uma referência externa, que normalmente deve aparecer no início do arquivo. O arquivo referenciado será transmitido uma única vez, independentemente da quantidade de vezes que seus protótipos sejam utilizados.

2.2.5. Geometria

Os nodos geométricos básicos **Sphere, Box, Cylinder** e **Cone**, permitem a definição de parâmetros, como raio, altura, arestas, que especificam diferentes objetos. Outros mais complexos podem ser criados através de combinações destes.

ElevationGrid permite a especificação de superfícies irregulares, através da definição de uma *grid*.

Extrusion representa sólidos de rotação.

O nodo **Text** trabalha com cadeias de caracteres e para formatação utiliza um nodo chamado FontStyle.

Estes nodos são utilizados no campo geometry do nodo **Shape**. O outro campo deste nodo, appearance, define a textura e o material do objeto, através de um nodo chamado **Appearance**.

2.2.6. Iluminação e visualização

Há três tipos de nodos fontes de luz; todos contêm uma intensidade, uma cor e uma intensidade ambiente.

DirectionalLight ilumina numa determinada direção somente os objetos descendentes do pai deste nodo.

PointLight ilumina geometrias dentro de um raio, a partir da posição definida. A luz é emitida igualmente em todas as direções. O raio e a posição são afetados pelas transformações do nodo pai.

SpotLight define um ângulo e raio para incidência da iluminação.

Outro ponto de vista da cena pode ser obtido com a mudança do observador, o que é feito pelo nodo **Viewpoint**.

WorldInfo fornece de maneira estruturada algumas informações sobre a construção da cena, como autor e título.

2.2.7. Sensores

Os nodos sensores geram eventos, baseados por exemplo, em ações do usuário com o *mouse*. **CylinderSensor**, **PlaneSensor**, **SphereSensor**, **TouchSensor** e **Anchor** são os nodos sensores que detectam este tipo de ação.

O nodo **TimeSensor** gera eventos como passos de tempo. É muito útil pois, em conjunto com interpoladores, pode produzir animação de objetos. Em particular `TimeSensor` determina o tempo de animação (através do campo `cycleInterval`) e se é repetitiva ou não (campo `loop`).

TouchSensor detecta quando um objeto, do grupo de seu pai, é ativado. Este sensor gera um evento de saída chamado `touchTime` que pode disparar o `TimeSensor` (se o campo `loop` for `FALSE`, senão já é disparado automaticamente), e assim, pode-se começar uma animação.

VisibilitySensor detecta quando certa parte do mundo torna-se visível ao usuário.

ProximitySensor detecta quando o usuário está navegando em uma região próxima ao objeto de interesse.

2.2.8. Interpoladores

Os nodos interpoladores determinam os *keyframes* da animação, em forma de uma função linear. Considerando $f(t)=n$ a função linear, os valores t, para os quais n é definido, determinam o intervalo entre os *keyframes*.

Existem seis tipos de nodos interpoladores; a diferença entre eles está no tipo de valores que são interpolados (n), como o próprio nome já especifica. São eles:

- **ColorInterpolator**: pode ser usado para variar a cor (RGB) de um objeto;
- **CoordinateInterpolator**: interpola linearmente um conjunto de coordenadas;
- **NormalInterpolator**: gera vetores normais;
- **OrientationInterpolator**: usado para rotacionar um objeto em torno de um eixo e ângulo especificados;
- **PositionInterpolator**: usado para transladar um objeto;

- **`ScalarInterpolator`**: apropriado para interpolar parâmetros definidos como um valor em ponto flutuante, por exemplo: raio, altura, etc.

2.2.9. Animações geradas com a utilização de VRML

Os nodos interpoladores, juntamente com `TimeSensor`, são responsáveis pela geração de animações *keyframes*.

O uso de sensores possibilita interação com o usuário, limitada pela funcionalidade dos sensores disponíveis, que se restringem à detecção de ações originárias do *mouse*.

Combinando interpoladores e sensores conseguimos obter animações interativas, limitadas a movimento *keyframe* e interação simples. Essa combinação é esquematizada na Figura 2.2. Um exemplo simples, do que se pode gerar, seria um objeto associado a um nodo interpolador e outro sensor. Quando for detectada uma ativação do sensor, o ciclo de movimento é iniciado. Outros exemplos podem ser vistos em tutoriais de Tamiosso [TAMIOSSO, 1997b] e Magalhães [MAGALHÃES, 1997].

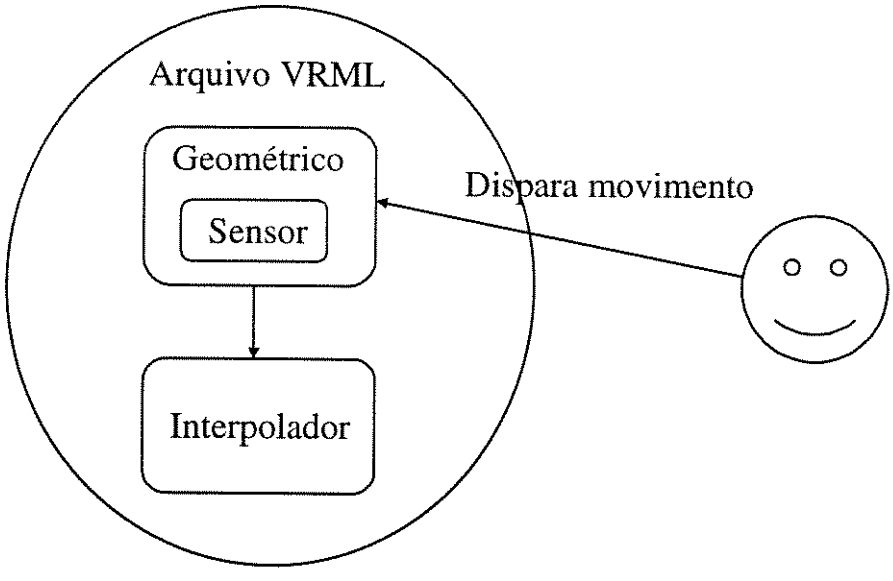

Figura 2.2: Animação interativa usando sensor e interpolador

2.3. Combinando VRML e Java

Nesta seção será apresentada a combinação de VRML e Java para a geração de animações mais complexas do que as obtidas apenas com a utilização de VRML, comentadas na Seção 2.2.9.

2.3.1. Introdução à Java

Java é uma linguagem orientada a objetos desenvolvida pela companhia Sun Microsystems e projetada para ser pequena, simples e com portabilidade para diversas plataformas e sistemas operacionais. [SUN, 1998]

Trabalhar com linguagens e ambientes de programação orientados a objetos é vantajoso pela capacidade de reutilizar código e criar programas modulares e flexíveis. Essas vantagens minimizam o tempo de implementação de um projeto e facilitam a sua manutenção.

Muitos conceitos de Java são herdados de C++. Como a maioria das linguagens orientadas a objetos, Java inclui uma biblioteca de classes que provê tipos de dados básicos, suporte à interface com o usuário, sistema de entrada e saída, suporte à rede, entre outras funcionalidades. Essas classes são escritas em Java e por isso são portáveis.

Embora Java pareça sintaticamente com C ou C++, em Java não há ponteiros; cadeias de caracteres e arranjos são objetos reais e a gerência de memória é automática.

Independência de plataforma é uma das mais importantes vantagens que Java tem sobre outras linguagens de programação, principalmente quando são considerados sistemas que precisam trabalhar em diversas plataformas.

Para aplicações distribuídas, Java oferece mecanismos de comunicação. Até recentemente, *sockets*[1] eram o único meio para escrever aplicações cliente/servidor em Java. Complementando a utilização de *sockets*, Java apresenta componentes que permitem a transferência de dados entre eles, sob a metáfora de arquivos: é possível enviar e receber dados através da rede como ler e escrever dados em arquivos. Atualmente, outros mecanismos são também utilizados para aplicações cliente/servidor, tais como RMI[2] (*Remote Method Invocation*) e a combinação Java/CORBA[3] (*Common Object Request Broker Architecture*), pois são implementados em um nível de abstração acima de *sockets* e, portanto, oferecem maior simplicidade ao programador.

Como Java é uma das linguagens de *script* que VRML especifica, existem alguns pacotes e classes específicas que devem ser providas (como extensão da

---

[1] Comunicação *peer-to-peer* que utiliza uma porta e um endereço de rede. [ORFALI, 1997]

[2] RMI é parte de JDK 1.1 (*Java Development Toolkit*) e foi desenvolvido para suportar invocações de métodos remotos sobre objetos localizados em máquinas virtuais Java. [ORFALI, 1997]

linguagem, pelo *browser* VRML[4] utilizado), para que seja possível a conexão de um programa Java com uma cena VRML.

2.3.2. Nodo Script de VRML

Uma das importantes características adicionadas à versão 2.0 de VRML é a capacidade de *scripting*, introduzida e associada com o nodo `Script`.

A animação gerada apenas com o roteamento de nodos sensores e interpoladores, como comentada na Seção 2.2.9, é limitada, pois não trata comportamentos que dependem de operações lógicas. Não é possível, por exemplo, escolher uma trajetória para um objeto, entre duas pré-definidas [TAMIOSSO, 1997a] [RAPOSO, 1997b]. VRML supera esta limitação através do nodo **`Script`**.

O `Script` é usado para programar comportamento em uma cena. Esse nodo é capaz de receber, processar e gerar eventos. O processamento é feito através de um programa associado (que será referenciado, de agora em diante, por ***script***) que realiza alguma computação, cujo resultado pode ser roteado, e assim, alterar o comportamento de outros nodos. Deste modo é possível modelar animações mais complexas que *keyframe*, ou seja, mais "inteligentes".

Ao nodo `Script` está associada uma linguagem de programação (por exemplo, Java ou JavaScript), informada no campo `url`; a referência a um arquivo com extensão `.class` indica que a linguagem utilizada é Java.

Os eventos recebidos e os que podem ser gerados pelo *script* devem ser declarados no nodo, respectivamente como:

```
eventIn     tipo (no padrão VRML)   nome
eventOut    tipo (no padrão VRML)   nome
```

Deste modo, o nodo `Script` tem a seguinte definição:

```
Script {
    exposedField  MFString    url           []
    field         SFBool      directOutput  FALSE
    field         SFBool      mustEvaluate  FALSE
```

---

[3] CORBA é o produto de um consórcio formado pela OMG (*Object Management Group*), que define um modelo para objetos distribuídos, sendo uma alternativa ao uso de RMI, com a vantagem de ser independente da linguagem de programação. [ORFALI, 1997]

[4] Um *browser* ou *plug-in* VRML é uma aplicação usada para visualizar arquivos VRML. Através da interface do *browser* é possível que o usuário movimente-se sobre um mundo digital, onde sons e imagens proporcionam uma sensação de realidade. [VRML, 1997] [SGI, 1997] [SONY, 1997]

```
# e um número de:
        eventIn      eventTypeName    eventName
        field        fieldTypeName    fieldName    initialValue
        eventOut     eventTypeName    eventName
    }
```

Não são permitidos campos `exposedField`, como pode-se notar na definição acima.

A localização do nodo no arquivo não tem efeito algum.

Um exemplo da utilização do nodo `Script` poderia ser:

```
DEF SOMESCRIPT Script {
    url "a_java_class.class"
}
```

Isto define um nodo para o qual eventos podem ser roteados. Todos os eventos dirigidos a ele são passados para a classe Java especificada no campo `url`, para futuro processamento.

2.3.3. Script programado em Java

Nesta seção Java será considerada como linguagem para definição do comportamento dos nodos **`Script`**. Será apresentado como um *script* interage com as cenas VRML e como programas Java podem dirigir a animação dessas cenas.

Funcionalidades como utilização de rede, interfaces com usuário e *multithread* são úteis se oferecidas pela linguagem de *script*, já que a potencialidade limite de uma cena VRML, ou seja, a "inteligência" do *script* associado, é determinada pela linguagem utilizada.

Se um arquivo `.class` for informado no campo `url` do nodo `Script`, deve ser assegurado que ele tenha a definição de uma classe com o mesmo nome, e esta seja subclasse de `Script` (classe do pacote vrml.node).

O *script* tem acesso a qualquer nodo VRML, que pode ser fornecido via `field` ou `eventIn`, podendo ler o conteúdo de todos os `exposedFields`.

Quando um *script* recebe um `eventIn`, o evento é tratado no método processEvent(Event), na forma de objeto Java (da classe Event). Para estabelecer qual `eventIn` exatamente está sendo tratado, e assim, determinar que tipo de processamento será efetuado, pode-se usar o método getName(), como por exemplo:

```
if ( e.getName().equals( "startTime" ) ) {
    // realiza algum processamento, onde "e" e um objeto
Event
  }
```

A classe vrml.Event tem três métodos associados: getName(), getTimeStamp() e getValue().

O evento VRML, que é passado para o processEvent(Event), deve estar declarado como `eventIn` na declaração do nodo `Script`, no arquivo VRML. Mas isso não é suficiente, pois se não houver `ROUTEs` para a estrutura `eventIn`, nenhum evento é passado para o processEvent(Event) processar.

O método getValue() de vrml.Event retorna o valor do nodo que foi roteado para o `Script`. Esse valor é um objeto vrml.ConstField, que é a superclasse de qual todos os tipos de variável derivam.

Extrair o valor de um evento roteado ao *script* é simples, basta testar o nome do evento sendo processado e usar *cast* na variável retornada de getValue().

Outro método que precisa, normalmente, ser definido no *script* é o initialize(), que é invocado quando a classe Java é instanciada no *browser* VRML. Nos *browsers*, cada nodo `Script` é instanciado uma vez, após verificação de que a cena sendo carregada é válida.

O método eventsProcessed() é chamado depois de toda invocação de processEvent(Event), ou seja, depois que algum conjunto de eventos é recebido. Geralmente, é neste método que o processamento é efetuado e os `eventOuts` são gerados.

Os `fields`, `eventIns` e `eventOuts` de um nodo `Script` podem ser acessados no *script* correspondente. Os `fields` definidos no nodo são acessados pelo nome e seus valores podem ser lidos ou escritos. Os métodos getField(), getEventIn e getEventOut() obtém referências para o `field`, `eventIn` e `eventOut`, respectivamente, cujo nome estiver como argumento (tipo String). O valor retornado pode ser convertido para uma classe Java apropriada.

Quando o método setValue(), por exemplo, é chamado sobre um objeto obtido por getEventOut(), o valor especificado como argumento gera um evento na cena VRML.

O efeito deste evento é especificado pela primitiva ROUTE, que pode associá-lo a algum outro elemento da cena, provocando mudança de comportamento.

Existem alguns outros métodos, mas os considerados principais para a construção de qualquer *script* e os que são utilizados neste trabalho foram aqui abordados. Para obter mais detalhes, o Apêndice C da especificação VRML pode ser consultado [VRML, 1997].

2.3.4. Exemplo de definição de *script*

A construção de um *script* pode ser melhor entendida com a ajuda de um exemplo, que consiste em uma cena VRML onde é necessário determinar a trajetória de um objeto em função de algum processamento envolvendo o tempo.

Para este exemplo serão considerados três nodos: `TimeSensor`, `Script` e `Transform`; a Figura 2.3 mostra o roteamento de eventos entre eles. Os nodos são mostrados com seus campos e eventos de interesse a este exemplo ("..." indica que o nodo possui mais campos ou eventos definidos). O nodo `TimeSensor`, reponsável por gerar o tempo da animação, envia um evento `fraction_changed` ao `Script` a cada passo de tempo gerado. O `Script` recebe este evento em `tempo_corrente`, realiza algum processamento para determinar uma posição, que é passada como evento ao campo `translation` do nodo `Transform`. Desta forma, o nodo `Box`, filho de `Transform`, definido no campo `children`, é deslocado para a posição gerada.

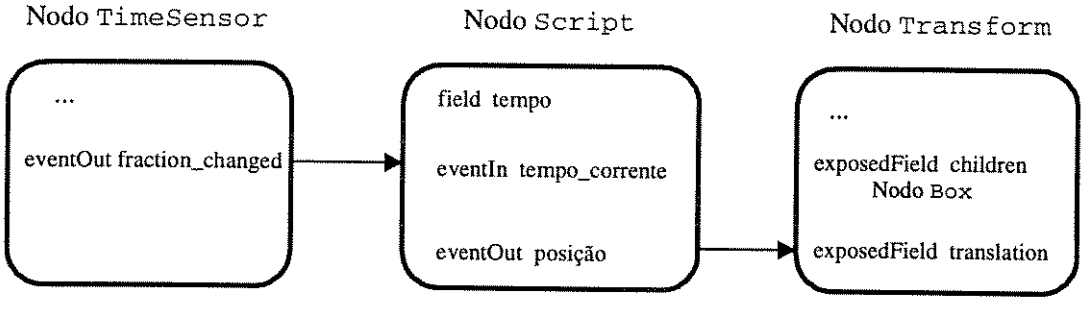

Figura 2.3: Exemplo de roteamento de eventos

O *script* Java, associado ao nodo `Script` da Figura 2.3, tem sua construção esquematizada na Figura 2.4, mostrando a utilização dos principais métodos.

O método initialize(), chamado quando a classe Java é instanciada no *browser* VRML, deve conter todas as inicializações de variáveis. Neste método, `fields` e `eventOuts`, definidos no nodo `Script`, devem ter suas referências obtidas para poderem

ser utilizados no processamento do *script*. Os objetos `time` e `position`, definidos no *script*, recebem as referências do `field tempo` e do `eventOut posição`, obtidas pelos métodos getField() e getEventOut(), respectivamente.

Script Java

```
initialize()
      time = getField("tempo");
      position = getEventOut("posição");


processEvent(Event e)
    if (e.getName().equals("tempo_corrente"))
         time.setValue(e.getValue())

eventsProcessed()
            ...
            time.getValue()
            ...
            position.setValue(translation)
```

Figura 2.4: *Script* programado em Java

Para receber eventos de tempo o *script* deve definir o método processEvent(Event), que é invocado a cada evento gerado pelo nodo `TimeSensor`. O parâmetro Event recebido pode ser comparado ao `eventIn tempo_corrente` para ter certeza de que o evento recebido corresponde ao tempo. O nome do evento recebido é obtido pelo método getName() e seu valor, pelo método getValue(). Este valor de tempo obtido é atribuído ao objeto `time` pelo método setValue().

Imediatamente após o recebimento de cada evento de tempo, o método eventsProcessed() é invocado. Este método realiza algum processamento para gerar uma posição em função do tempo. O valor do tempo é obtido do objeto `time` pelo método getValue(). A posição é calculada e atribuída à variável `translation`, definida no *script*. O `eventOut posição` é gerado pelo *script* quando o valor do objeto `position` é determinado, com o valor da variável `translation`, pelo método setValue(). Desta forma, o nodo `Transform` recebe o evento através de seu campo `translation` e desloca seus nodos filhos para a posição gerada.

2.3.5. Animações geradas com a utilização de VRML e Java

Para associar, às cenas, comportamentos que dependem de operações lógicas e, assim, estender o conjunto inicial de animações, obtendo maior flexibilidade para aplicações, um nodo `Script` deve ser usado (conforme exemplificado na seção anterior).

Um cenário onde o usuário pode escolher uma trajetória para um objeto, entre duas pré-definidas, utiliza o `Script` para determinar qual das trajetórias o objeto deve seguir, de acordo com a interação do usuário. As trajetórias são definidas por nodos interpoladores e, por isso, a animação resultante é *keyframe*. A Figura 2.5 mostra o esquema de controle de uma animação *keyframe* interativa que utiliza um *script* para definir decisão lógica ou processamento.

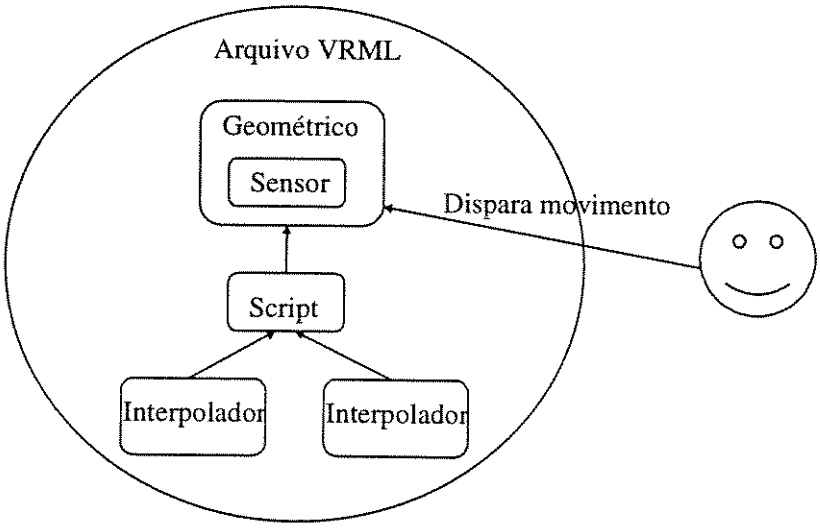

Figura 2.5: Animação interativa usando sensor, interpolador e *script*

A modelagem de movimentos cinemáticos ou dinâmicos também pode ser feita através do nodo `Script`, onde eventos de tempo devem ser recebidos pelo *script*, que resolve um sistema de equações e gera uma nova posição. Posteriormente, a posição gerada pelo *script* pode ser roteada para algum objeto da cena e, desta forma, promover a animação. Este tipo de animação segue o esquema da Figura 2.6. Nestes casos, onde o *script* é usado somente para gerar posições, a interação restringe-se ao disparo do movimento; por exemplo, quando o usuário ativar o objeto que tem o sensor e o movimento (*script*) associado.

Nas animações apresentadas anteriormente, a interação se resume à utilização de nodos sensores. A fim de criar maior nível de interação com o usuário, recursos da linguagem Java foram utilizados.

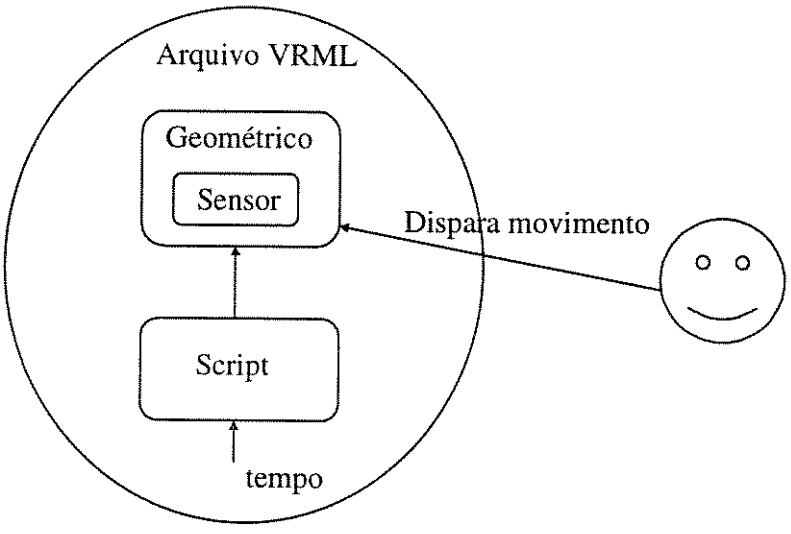

Figura 2.6: Animação interativa usando sensor e *script*

Utilizando Java como linguagem de *script*, seus recursos de comunicação entre aplicações distribuídas são disponíveis. Deste modo, é possível que o *script* receba e envie dados de e para outra aplicação Java, respectivamente. Essa aplicação, da qual vêm os dados, poderia ser responsável pela interface com o usuário, já que Java possui suporte a diversos componentes gráficos (por exemplo, campos de texto, botões, etc.). Através da interface gráfica o usuário informa parâmetros que podem corresponder a variáveis das equações usadas para cálculo do movimento no *script*. No caso de movimento cinemático, um desses parâmetros poderia ser a velocidade. Deste modo, os parâmetros informados controlam o movimento e a interface pode ser considerada um **painel de controle** para a animação.

Resumindo, de posse dos recursos de rede e de interface com o usuário oferecidos pela linguagem Java, combinados aos nodos, especificação para linguagem de *script* Java e *browsers* VRML, pode-se obter animações mais complexas e com maior capacidade de interação. Essas animações seguem o esquema da Figura 2.7. Neste exemplo, o nodo `Script` está associado a um *script* Java, usado para resolver equações de movimento. Este cálculo envolve recebimento de eventos de tempo e variáveis controladas pelo usuário, via painel de controle. Desta forma, para cada evento

de tempo recebido e considerando parâmetros informados pelo usuário, uma nova posição é obtida para o objeto.

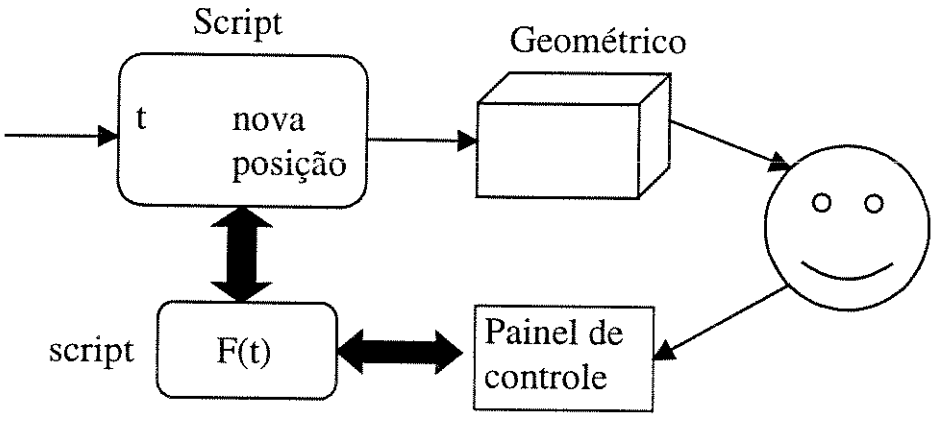

Figura 2.7: Animação interativa que combina *script* programado em Java e VRML

2.4. Considerações finais

Neste capítulo, várias técnicas de animação foram discutidas e animações interativas foram definidas. Para a modelagem de cenas tridimensionais, VRML foi introduzida. Esta linguagem permite a definição de cenas animadas e interativas. A animação é obtida através da utilização de nodos interpoladores e a interação é introduzida pelos nodos sensores, que proporcionam a detecção de ações simples do usuário. Animações mais complexas, apresentadas na Seção 2.3.5, que envolvem maior interação do usuário no movimento, são obtidas através da utilização do nodo `Script`. Este nodo pode ser associado a um *script* que é capaz de realizar algum processamento disponível pela linguagem Java. Com a utilização dos recursos de rede e interface gráfica providos pela linguagem Java, é possível definir um painel de controle para a animação, através do qual o usuário define valores de parâmetros que são enviados ao *script* para processamento. Os resultados gerados pelo *script*, ao final do processamento, podem interferir no movimento definido na cena VRML e, assim, o usuário pode avaliar sua interação. Este tipo de animação é a motivação para o desenvolvimento de um ambiente que facilita seu processo de criação. Esse ambiente para a construção de animações interativas será apresentado no Capítulo 3.

Capítulo 3.

Um ambiente para geração de animações interativas

As animações interativas modeladas em VRML e que utilizam a linguagem Java para *script*, permitindo a comunicação com outras aplicações Java chamadas painéis de controle, são exploradas pelo ambiente descrito neste capítulo. Diversas aplicações, como por exemplo, na área de educação e em simulações, podem ser beneficiadas com a utilização destas animações. Considerando uma aplicação deste tipo, pode-se identificar dois tipos de usuários: o que modela a animação interativa, também chamado de animador, e o que se utiliza dela para obter resultados, chamado de usuário final.

Inicialmente, o animador deve dominar VRML e ser um programador com experiência em Java. O esforço deste usuário na geração de animações pode ser minimizado, já que este tipo de animação apresenta pontos comuns a diversos tipos de aplicação, todos relacionados com a implementação em Java. A semelhança está na comunicação entre o *script* VRML e o painel de controle, nos componentes gráficos (definidos pela linguagem) e na forma do *script*. A diferença consiste na cena VRML, nos objetos de interface gráfica escolhidos e nos dados utilizados na comunicação. Deste modo, um processo automatizado para a modelagem de animações interativas seria viável e proporcionaria muitos benefícios. O ambiente deveria ainda encapsular toda a programação Java, tornando-a transparente ao animador, que se restringiria a entrada de simples informações, operando um sistema de fácil aprendizado e interação.

O objetivo deste capítulo é mostrar o ambiente projetado para a construção de animações interativas. Partindo de uma motivação, o ambiente é descrito e seus requisitos são levantados. A funcionalidade deste ambiente, a entrada para processamento e os produtos que devem ser gerados são apresentados. Este capítulo encerra a exposição com a

arquitetura do ambiente e uma metodologia para o desenvolvimento de animações interativas, que introduzirão o processo de implementação, discutido no próximo capítulo.

3.1. Motivação

A modelagem de uma animação usando VRML é limitada à técnica de *keyframe*, provida pelos nodos interpoladores. Para que o usuário final possa interagir com essas animações, os nodos sensores são os recursos disponíveis, o que entretanto, restringe a interação do usuário a ações simples com o *mouse*.

Animações mais complexas, construídas com diferentes técnicas para modelagem do movimento e maior interatividade, podem beneficiar usuários finais, que utilizam essas animações interativas para obter resultados em outros trabalhos (de simulação ou ensino, por exemplo). Por outro lado, são também mais difíceis para construir, exigindo do animador, usuário responsável pela modelagem, conhecimento relacionado à programação.

Essas animações mais complexas, como exemplificado na Seção 2.3.5, podem ser obtidas pela combinação de VRML e Java. A Figura 3.1 mostra o esquema geral para este tipo de animação, onde o usuário final interage com a animação através de um painel de controle e um *browser* VRML. O *browser* é responsável pela apresentação da cena. A interação pode ser direta na cena, através de nodos sensores, ou pela definição de parâmetros que controlarão o movimento. O painel de controle é responsável por enviar esses parâmetros ao *script* VRML. O *script* calcula o movimento baseado nesses parâmetros e gera uma nova posição, que é passada para o objeto da cena VRML sendo animado. O painel de controle e o *script* são escritos em Java.

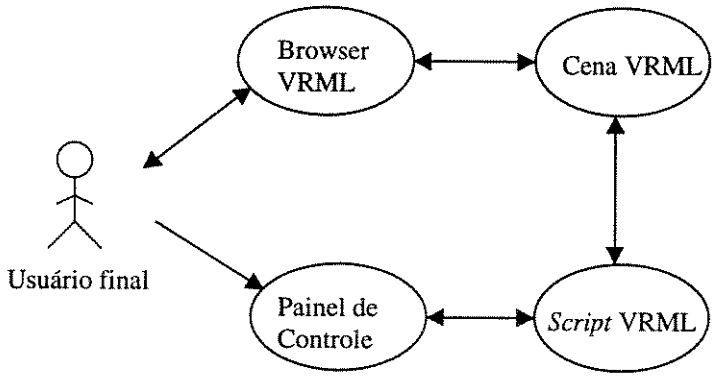

Figura 3.1: Animação interativa

O objetivo geral deste trabalho é implementar um ambiente capaz de gerar animações que seguem o padrão da Figura 3.1, tentando minimizar o esforço do usuário responsável pela modelagem. Este ambiente possibilita que, a partir de uma cena VRML estática, o animador obtenha uma cena VRML animada, que pode ter seu comportamento alterado através de parâmetros informados em um painel de controle. Desta forma, o ambiente gera animações com as seguintes características:

- podem usar nodos sensores para interação;
- devem utilizar o nodo `Script` para processamento de informações originadas pelo usuário final e cálculo de algum tipo de movimento;
- pode existir um painel de controle responsável pela interação com o usuário final e envio de parâmetros ao *script* VRML;
- os parâmetros informados pelo usuário final devem fazer parte do processamento do *script*;
- o processamento do *script* deve interferir na cena VRML; somente assim, a interação com o usuário final via painel de controle e o *script* teriam sentido.

3.2. Descrição do ambiente

A Figura 3.2 mostra o ambiente para a criação de animações interativas como uma "caixa preta" que recebe como entrada uma cena VRML estática e gera uma outra animada e interativa.

A utilização do ambiente é iniciada pela definição da cena VRML a ser animada. O animador interage com o ambiente, escolhendo algum tipo de movimento a ser aplicado a algum objeto desta cena. O tipo de movimento pode ser completamente especificado ou selecionado de uma lista contendo movimentos pré-definidos. Se o movimento for especificado, a definição do painel de controle contendo parâmetros da animação também é requerida. O animador deve informar os parâmetros da animação e quais componentes gráficos estarão associados a eles.

Como resultado, o ambiente gera, para cada cena VRML estática sendo manipulada, uma nova cena animada e interativa, que tem associados um *script* e um painel

de controle responsáveis pelo movimento e interação com o usuário final respectivamente, seguindo o padrão da Figura 3.1.

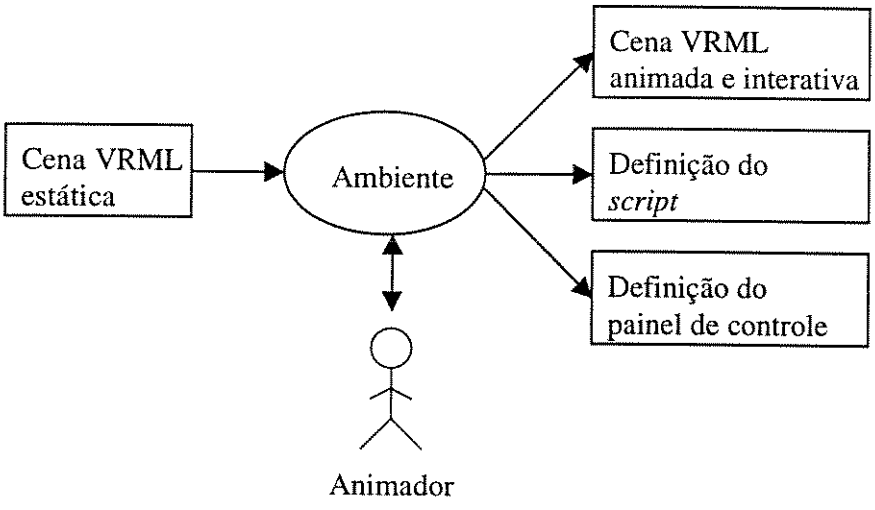

Figura 3.2: Descrição do ambiente para geração de animações interativas

Considerando a descrição da Figura 3.2, o conhecimento exigido aos usuários deste ambiente, os animadores, se resume à modelagem de cenas VRML estáticas e tipos de movimento. Os animadores constrõem as animações interativas para aplicações de usuários finais. Dos usuários finais não se requer nenhuma interação com o ambiente, mas sim com a animação interativa gerada.

Para a geração de qualquer tipo de animação interativa, o ambiente produz sempre o mesmo resultado: uma nova cena VRML, um *script* e um painel de controle. Esta nova cena reflete modificações referentes à inclusão de *script* e movimento.

3.3. Requisitos

Tendo em mente a descrição das animações interativas que se pretende gerar automaticamente, cujo padrão é mostrado na Figura 3.1, definem-se os requisitos para o ambiente. As características do ambiente para construção de animações interativas são classificadas de acordo com dois aspectos: interação e construção da animação.

Quanto à interação:

- deve ser de fácil utilização e proporcionar rápido aprendizado;
- deve ser capaz de associar a objetos de uma cena VRML algum tipo de movimento;
- o animador deve informar que movimento deseja aplicar a qual objeto de uma cena. O ambiente deve prover meios para isto;
- além do movimento, o animador deve ser capaz de informar parâmetros da animação que serão controlados via interface. O ambiente deve ser "inteligente" para associá-los a equações do movimento e prover meios para definição de interface.

Relacionado à construção da animação:

- o movimento deve ser informado como função do tempo e pode estar associado com uma interface gráfica;
- movimentos podem ser pré-definidos, pertencentes a uma biblioteca. O ambiente pode possibilitar e facilitar a integração desses movimentos ao seu ambiente;
- uma vez definida a animação interativa, o movimento e a interface gráfica, o animador deve ser capaz de reutilizar a animação para outras aplicações ou editá-la para modificações.

Pode-se notar que os requisitos gerais do ambiente não se referem especificamente a nenhuma linguagem de programação, pois é uma característica da animação interativa transparente aos usuários do ambiente. Isto garante que o animador não precisa de conhecimento em programação, muito menos o usuário final que interagirá com a animação construída.

3.4. Funcionalidades

A idéia geral do ambiente é que, através de interfaces, o animador possa entrar com informações de forma simples, operando um sistema de fácil aprendizado e interação e, como resultado, obtenha uma cena animada e interativa.

Os módulos funcionais do ambiente são apresentados na Figura 3.3, onde as seqüências utilizadas para a execução de funções podem ser observadas, através do número

que cada um dos módulos apresenta. A seguir todos os módulos funcionais serão detalhados tomando como referência os números do diagrama.

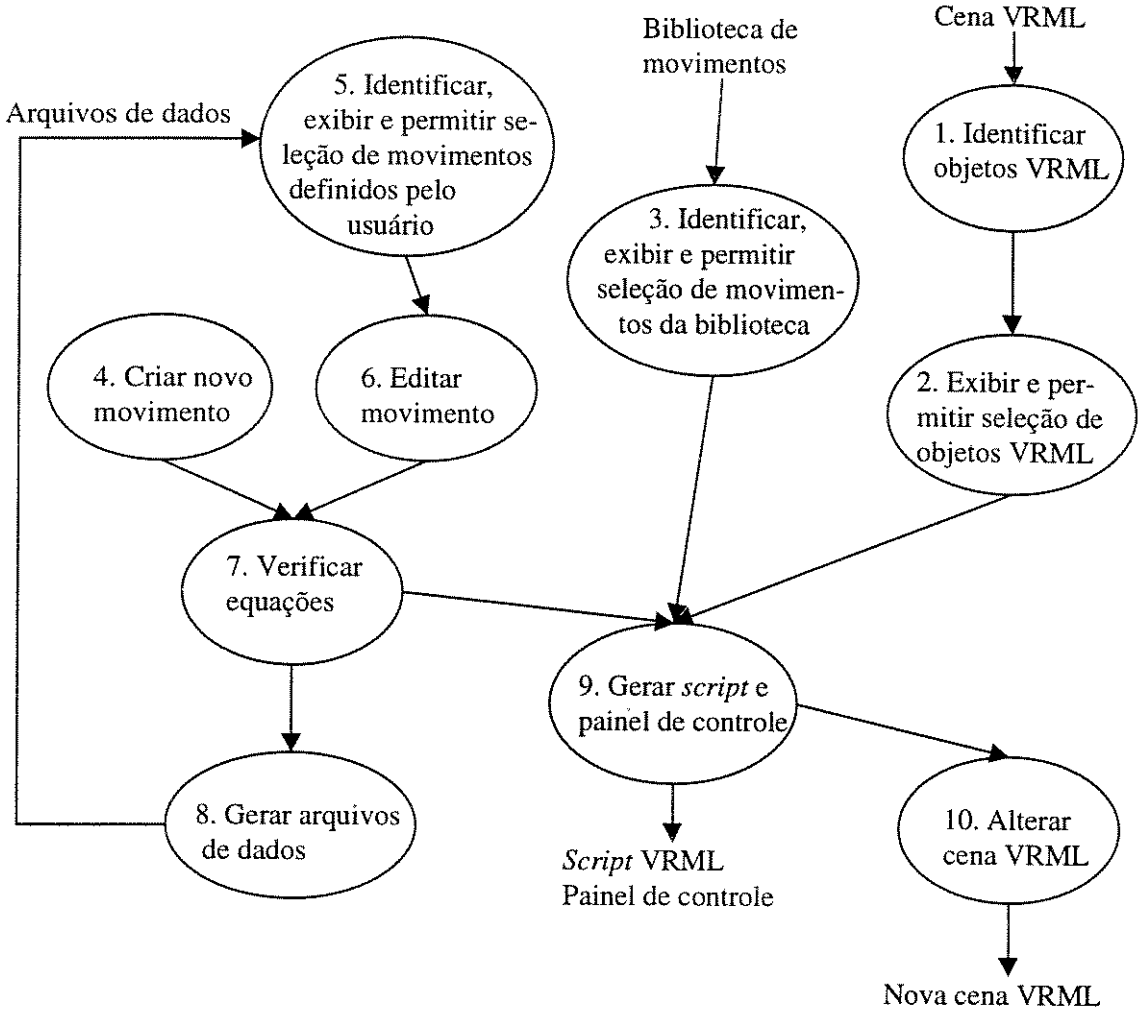

Figura 3.3: Diagrama funcional

O animador deve informar ao ambiente uma cena VRML para dar início ao processo de criação da animação interativa. A animação será criada para um dos objetos da cena informada. O ambiente lê a cena e identifica todos os objetos contidos nela, capazes de receber movimento, como por exemplo, objetos geométricos e nodos de agrupamento. Esta é a função do módulo 1. Para isso, o ambiente deve conhecer a sintaxe de VRML e saber quais objetos são de interesse. Determinando estes objetos, o ambiente deve exibí-los ao animador para possível seleção, o que é feito pelo módulo 2. O animador seleciona um objeto VRML para ser animado.

O movimento a ser atribuído para um objeto da cena VRML pode ser pré-definido ou não. No primeiro caso, o ambiente, através do módulo 3, deve exibir uma lista dos movimentos definidos em uma biblioteca. No segundo (módulo 4), deve prover interfaces para a criação de um novo movimento.

A biblioteca de movimentos faz parte da funcionalidade que o ambiente oferece ao animador. Os usuários que criam movimentos para esta biblioteca devem ter experiência em programação e, ainda, precisam conhecer um pouco da implementação do ambiente. Depois que o movimento é criado, ele precisa ser integrado ao ambiente para que os animadores possam ter acesso. A criação de movimentos na biblioteca é indicada quando forem detectadas muitas aplicações para ele. A atuação deste tipo de usuário, que cria movimentos para a biblioteca, é uma contribuição ao ambiente, pois à medida que um maior conjunto de movimentos é disponibilizado, o ambiente se torna mais versátil.

Com a lista de objetos da cena VRML e a lista de movimentos pré-definidos, basta o animador selecionar um item de cada lista para obter uma animação interativa. Se preferir criar um novo tipo de movimento, uma interação adicional é necessária para definir equações e parâmetros da animação. Esta é uma responsabilidade do módulo 4.

Para definição do movimento o animador deve contar com interfaces facilitadoras, pois as equações podem ser muito complexas, envolvendo muitas variáveis. Neste ambiente o movimento deve ser especificado como função do tempo, utilizando as coordenadas cartesianas x, y e z. O ambiente pode identificar as variáveis informadas e solicitar as definições automaticamente, para que o animador não esqueça de nenhuma variável; assim, a consistência das equações é mantida. Esta é a tarefa do módulo 7. A definição de uma variável consiste na atribuição de um tipo (por exemplo, `int` ou `float`) e valor, que pode ser uma constante, uma outra equação ou um parâmetro da animação que será controlado pelo usuário final. No último caso, o ambiente deve prover interfaces para a definição do que fará parte do painel de controle.

Para criar um painel de controle, o animador pode escolher os componentes gráficos que desejar para o compor, indicando os que corresponderão a parâmetros da animação. O animador deve dispor de um conjunto de componentes gráficos, que proporciona interação amigável e, principalmente, possibilita a entrada de parâmetros para a animação. Exemplos

desses componentes seriam campos de texto, barras de rolagem e botões. O envio de valores de componentes gráficos, determinados na criação do painel de controle, deve acontecer de maneira confiável, rápida e transparente.

No caso de movimento definido pelo animador, este deverá poder reutilizar e modificar o *script* e o painel de controle para a criação de uma nova animação interativa. Para isso, o ambiente deve manter um mecanismo para armazenar essas informações. Este mecanismo poderia ser uma estrutura de arquivos contendo informações chaves para a recriação da animação. Quando uma nova animação é criada pelo animador, o ambiente, através do módulo 8, gera arquivos de dados correspondentes. Na inicialização do ambiente esses arquivos de dados são identificados e exibidos pelo módulo 5 e, se for uma opção do animador, o ambiente deve prover interfaces para edição da animação, cujos arquivos foram selecionados. Esta é a função do módulo 6.

Com a facilidade para reutilizar animações através de pequenas modificações relativas a equações ou painel de controle, o animador pode testar e validar o tipo de animação criado, podendo posteriormente adicioná-lo à biblioteca.

A atribuição de um tipo de movimento interativo a uma cena VRML implica na existência de um painel de controle e de um *script*. O painel de controle é responsável pela interação com o usuário, e o *script*, pelo cálculo do movimento. A função do módulo 9 é gerar estes arquivos com base nas informações dadas pelo animador. No caso de movimentos pré-definidos, o *script* e o painel de controle são gerados automaticamente, sem a necessidade de informações adicionais. Para a utilização de outros tipos de movimento os módulos responsáveis solicitam as informações necessárias do animador.

O ambiente, além de gerar um *script* e um painel de controle, também deve identificar o objeto VRML sendo animado e realizar modificações necessárias no arquivo VRML, tais como inserir nodos e rotear eventos entre eles. Como o movimento é determinado em função do tempo, a cena VRML deve prover um nodo `TimeSensor` que passa eventos para o `Script`, que efetua o cálculo do movimento e repassa a posição para o objeto sendo animado. Gerar a nova cena VRML com estas modificações é a função do módulo 10.

A Figura 3.4 mostra todas as possibilidades para se obter uma animação interativa utilizando a seqüência dos módulos funcionais. A seqüência (a) mostra a criação de uma animação que utiliza a biblioteca de movimentos; em (b) o movimento é definido pelo animador e em (c) a animação é criada a partir da reutilização de outra previamente definida.

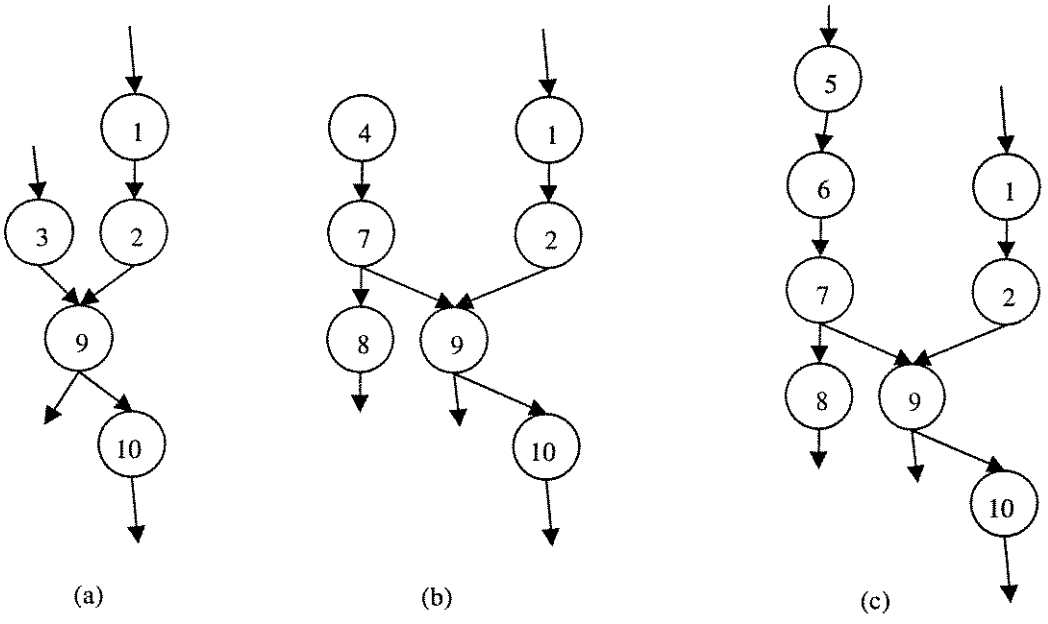

Figura 3.4: Maneiras para se obter animações interativas

As funcionalidades do ambiente podem ser resumidas de acordo com a Tabela 3.1. Esta tabela mostra as funcionalidades essenciais que o ambiente deve prover para atender à necessidade básica de seus usuários: geração e reutilização de animações interativas.

Identificar objetos de uma cena VRML e exibí-los.
Exibir movimentos de uma biblioteca.
Prover interfaces para inserir novo movimento.
Prover mecanismo para reutilização de movimentos.
Gerar script e aplicação Java para cada animação interativa.
Alterar a cena VRML para que contenha a animação criada.

Tabela 3.1: Resumo das funcionalidades

3.5. Arquitetura

A arquitetura do ambiente para geração de animações interativas é apresentada na Figura 3.5, através de módulos definidos de acordo com as funcionalidades descritas anteriormente. A Figura 3.5 detalha o ambiente apresentado na Figura 3.2, identificando todos os módulos necessários para que uma cena VRML estática seja transformada em animada com possibilidade de interação com o usuário final.

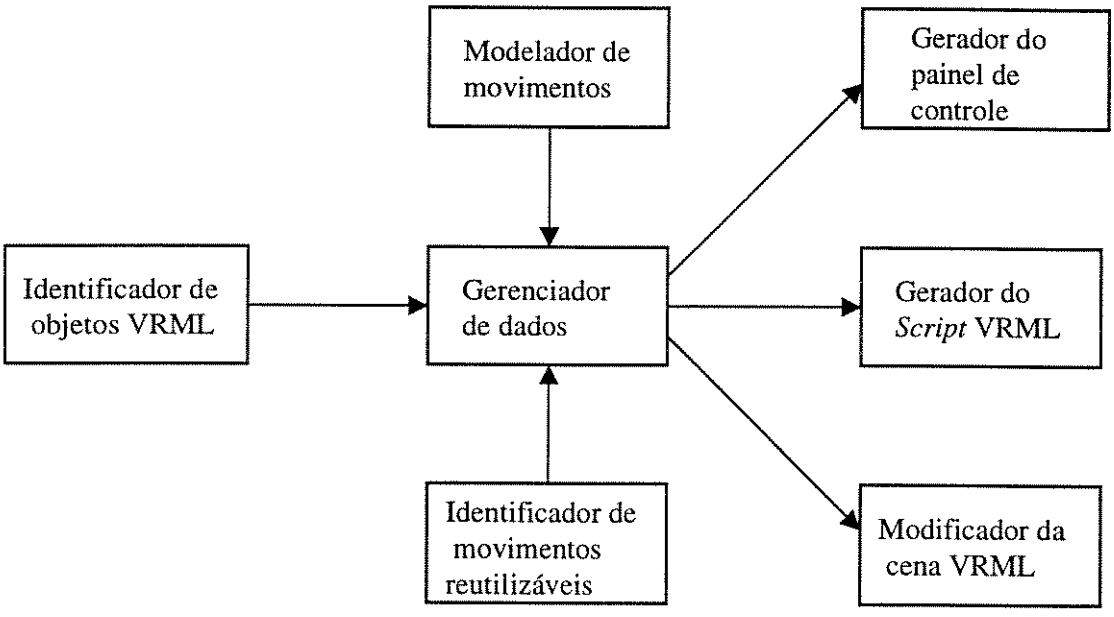

Figura 3.5: Arquitetura do ambiente

Uma cena VRML é a entrada para o módulo "Identificador de objetos VRML", que conhece a linguagem e é capaz de determinar quais nodos existentes na cena são aptos a receber movimento. Identificados estes objetos, eles são passados ao "Gerenciador de dados".

O "Modelador de movimentos" provê ao "Gerenciador de dados" vários tipos de movimentos pré-definidos para animações interativas em uma biblioteca que inclui a definição de painel de controle e *script* para cada um deles.

O "Identificador de movimentos reutilizáveis" também provê ao "Gerenciador de dados" tipos de movimentos para animações interativas. Neste caso são os movimentos definidos previamente pelo animador em que o ambiente gerou arquivos de dados.

O "Gerenciador de dados" é responsável pelo controle do processamento da criação de animações interativas. Recebe uma lista de objetos VRML e uma de movimentos, que deve exibí-las ao animador para seleção. O animador associa um objeto VRML a um tipo de movimento, que pode pertencer à lista ou ser definido. Com estas informações obtidas do animador, o "Gerenciador de dados" ativa os outros módulos.

Se o movimento for escolhido a partir da lista provida pelo "Modelador de movimentos", os módulos "Gerador de painel de controle" e "Gerador de *script* VRML" não precisam de interação com o animador, pois o painel de controle e o *script* são gerados automaticamente. Caso contrário, devem prover interface e mecanismo para armazenar o novo tipo de movimento interativo, possibilitando posterior reutilização. O "Gerador de *script* VRML" deve receber equações de movimento e o "Gerador de painel de controle", componentes de interface gráfica que representarão parâmetros da animação.

Para completar a criação de animações interativas, o módulo "Modificador da cena VRML" deve conhecer e inserir as modificações necessárias para que a cena VRML suporte a animação.

A Tabela 3.2 mostra a associação entre os módulos arquiteturais e os funcionais descritos na seção anterior. Todos os módulos arquiteturais englobam ou estão relacionados com as funcionalidades dos módulos apresentados na Figura 3.3.

Módulos arquiteturais	**Módulos funcionais**
Identificador de objetos VRML	1
Modelador de movimentos	entrada para 3
Identificador de movimentos reutilizáveis	entrada para 5
Gerenciador de dados	2, 3, 5
Gerador de script VRML	4, 6, 7, 8, 9
Gerador de painel de controle	4, 6, 7, 8, 9
Modificador da cena VRML	10

Tabela 3.2: Módulos arquiteturais associados a módulos funcionais

Tendo a arquitetura definida, a implementação do ambiente já pode ser descrita seguindo uma metodologia de desenvolvimento para a criação de animações interativas, o que é assunto para a próxima seção.

3.6. Metodologia de desenvolvimento

O ambiente foi dividido e desenvolvido em níveis para que o processo de geração de animações interativas fosse testado e validado desde o início, tornando a fase de integração das partes desenvolvidas mais simples, pois a maioria dos testes já teria sido feito. Cada nível na Figura 3.6 corresponde a uma implementação que o animador pode interagir para obter uma animação interativa. À medida que os níveis foram implementados, as funcionalidades discutidas na Seção 3.5 foram introduzidas. Com isso, as responsabilidades exigidas do animador, relacionadas ao conhecimento e à interação, foram paulatinamente minimizadas e, assim, foi facilitada a utilização do ambiente.

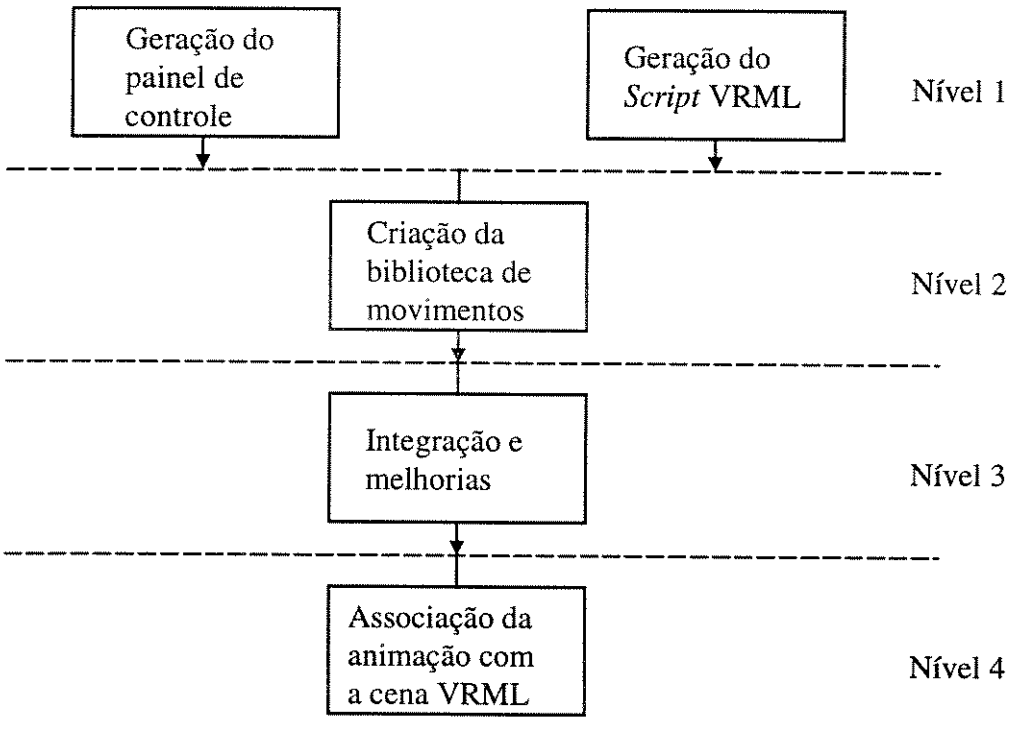

Figura 3.6: Metodologia de desenvolvimento

O primeiro nível foi dedicado à implementação de duas interfaces, que puderam ser feitas em paralelo e sem dependências. A primeira facilita a criação de painéis de controle e proporciona sua conexão a um *script* VRML. Um painel de controle deve ser gerado

segundo um modelo, considerando suas responsabilidades: interação com o usuário final e envio de parâmetros ao *script*. A outra interface deve ser responsável pela entrada de equações do movimento, por receber parâmetros do painel de controle e, por fim, gerar o *script* VRML, também de acordo com um modelo. Como estas duas interfaces são independentes, algumas redundâncias foram encontradas: o animador deve informar, na primeira, quais componentes gráficos são parâmetros da animação, e na segunda, quais variáveis da equação são controladas via painel de controle. A informação dos parâmetros da animação precisa ser dada nas duas interfaces, o que além de ser inconveniente para o animador, pode provocar inconsistências; por exemplo, o número de componentes gráficos enviando valores para o *script* pode ser diferente do número de valores sendo recebidos.

Embora não fosse a melhor forma e ainda incompleta, ao final da implementação do primeiro nível era possível gerar e testar vários tipos de animações interativas. Para isso era preciso interagir com as duas interfaces definidas, de modo que a consistência, relacionada aos parâmetros da animação e à definição de equações, ficasse sob responsabilidade do animador, bem como a geração de um novo arquivo VRML, com as modificações necessárias para inserir a animação interativa criada.

Após a validação dos modelos para criar painel de controle e *script*, obtida com os primeiros testes, alguns tipos de movimento comuns e consagrados poderiam ser definidos, seguindo o modelo, e inseridos em uma biblioteca. Isto consistiu em um segundo nível, onde o animador poderia construir uma animação interativa selecionando um desses movimentos, sem ter o conhecimento do cálculo envolvido no movimento.

O terceiro nível foi dedicado à integração dos níveis anteriores. Uma avaliação também foi feita para a constatação de possíveis melhorias, relacionadas à interação, à reutilização de movimento e ao gerenciamento de consistência associada com a informação de equações e parâmetros da animação. A reutilização de movimentos já era possível através da biblioteca; neste nível foi introduzida a funcionalidade de edição de animações previamente definidas pelo animador. A implantação dessas características teve o objetivo de facilitar a utilização do ambiente, tornando-o mais versátil. Desta forma, ao utilizar este nível para a criação de uma animação interativa, o animador elimina a preocupação com inconsistências e dispõe de uma maneira para reutilizar suas animações.

O quarto e último nível, no desenvolvimento do ambiente para a construção de animações interativas, implementou a integração de arquivos VRML, que consiste na associação da cena com a animação. A animação interativa é gerada, mas não associada a um objeto de uma cena VRML. Para isso, algumas modificações na interface, correspondentes aos primeiros painéis do ambiente, foram feitas. Nestes painéis o animador deve informar uma cena VRML que deseja animar e selecionar qual dos objetos receberá o movimento, que pode ser obtido da biblioteca ou definido. A partir daí, as interfaces já são conhecidas de outros níveis. A responsabilidade de conhecer VRML para gerar o novo arquivo, com as modificações necessárias para inserir a animação interativa criada, é eliminada do animador neste nível.

É importante lembrar que o ambiente, visto na Figura 3.2, contendo todas as funcionalidades descritas, pode ser implementado sob diferentes metodologias para o desenvolvimento de animações interativas. Este modelo em níveis apresentado foi a metodologia adotada neste trabalho.

3.7. Considerações finais

Neste capítulo um ambiente para construção de animações interativas foi descrito de acordo com requisitos e funcionalidades especificadas. Outros ambientes poderiam ser definidos para o mesmo propósito com a variação das especificações. De acordo com as funcionalidades, apresentadas na Seção 3.4, os módulos que compõem a arquitetura do ambiente foram identificados e esquematizados na Seção 3.5. Com base na mesma arquitetura o ambiente pode ser implementado de diversas maneiras; no entanto, uma metodologia de desenvolvimento em níveis foi adotada. Ao final da implementação de cada nível desta metodologia, apresentada na Seção 3.6, o animador dispõe de um subconjunto das funcionalidades, através do qual já é possível criar animações interativas. À medida que os níveis vão sendo implementados, funcionalidades são introduzidas e o esforço exigido do animador é minimizado. Este desenvolvimento em níveis facilita o teste e a validação do ambiente.

O capítulo 4, seguindo a metodologia para o desenvolvimento de animações interativas, apresentada na Seção 3.6, descreverá a implementação do ambiente incluindo

todas as funcionalidades da Seção 3.4. As interfaces gráficas desenvolvidas serão exibidas junto com a descrição do modo de operação.

Capítulo 4.

Aspectos de implementação

Este capítulo descreve como a funcionalidade do ambiente para a construção de animações interativas, apresentada no capítulo anterior, é disponibilizada ao animador e como foi implementada, seguindo a metodologia de desenvolvimento.

A descrição do ambiente implementado será apresentada seguindo um exemplo simples para a criação de uma animação interativa. As interfaces gráficas serão mostradas com a descrição do modo de operação, para o exemplo. A seqüência da apresentação segue a metodologia de desenvolvimento apresentada na Seção 3.6. Desta forma, a aplicação **IULib**, para geração do painel de controle, e **ScrLib** para o *script*, serão descritas individualmente, pois foram implementadas de forma modular e independente (nível 1 da Figura 3.6). Em seguida, será mostrado como as aplicações IULib e ScrLib foram integradas, considerando algumas facilidades introduzidas. A associação com a cena VRML sendo animada é discutida ao longo dessas seções. O processo de criação da animação interativa, usada para exemplificar, será mostrado para as interfaces independentes e após a integração, onde as facilidades introduzidas podem ser verificadas.

As interfaces gráficas do ambiente foram implementadas com o objetivo de proporcionar, ao animador, um sistema de fácil aprendizado e interação, onde através de informações simples o animador pode gerar uma animação interativa a partir de uma cena VRML estática.

Com a validação das interfaces IULib e ScrLib, pode-se utilizar seus produtos, ou seja, os arquivos gerados, como base para a definição de uma biblioteca de movimentos pré-definidos.

Os detalhes de implementação, classes e métodos serão discutidos na apresentação de cada seção. A especificação das classes será apresentada em diagramas de acordo com a UML (*Unified Modeling Language*) [FOWLER, 1997]. Na implementação de todo o ambiente para construção de animações interativas, a linguagem Java foi utilizada; assim, várias referências a ela são encontradas ao longo das seções.

4.1. Exemplo motivador

O exemplo utilizado para auxiliar a descrição do ambiente é de uma cena VRML com uma esfera à qual se deseja associar um movimento retilíneo uniforme. Será considerado que a cena VRML já está definida, com apenas um nodo `Shape`, cujo campo `geometry` define um nodo `Sphere`, sem nenhuma referência à animação ou *script*. Esta cena é descrita a seguir:

```
#VRML V2.0 utf8
Shape {
  geometry Sphere {}
  appearance Appearance {
        material Material {
              diffuseColor 1 0 0
        }
  }
}
```

O movimento retilíneo uniforme a ser associado é representado pela equação:

$$s = s_0 + v.t$$

onde t é um instante de tempo, v a velocidade, s_0 a posição inicial e s a posição a ser gerada pelo *script* e atribuída à esfera. Os parâmetros desta equação que podem ser controlados pelo usuário final são v e s_0.

A utilização das interfaces do ambiente será descrita através da associação do movimento à cena VRML estática. Assim, nas seções seguintes será mostrado como criar o movimento retilíneo uniforme e o painel de controle para a animação da esfera na cena VRML. Primeiramente, o *script* e o painel de controle serão criados separadamente, e a cena VRML deverá ser modificada manualmente para referenciar o *script* criado. Então, com as interfaces integradas compondo o ambiente final, o processo de criação do *script* e painel de controle será repetido, e neste caso, a alteração da cena VRML ficará por conta do ambiente.

4.2. ScrLib: Interface que define *scripts* VRML

Na busca para criar animações com maior complexidade e grau de interação do que proporcionado por VRML, a interface ScrLib foi desenvolvida para a geração automática de *scripts* VRML baseada em informações do animador. Estes *scripts* podem ser definidos com vários campos `field`, `eventIn` e `eventOut` de tipos quaisquer, sendo aptos a se comunicarem com painéis de controle gerados pela IULib (conforme a Seção 4.3).

Utilizando a ScrLib para a geração de um *script*, o animador deve definir o interfaceamento com a cena VRML, que acontece através da definição de `fields` e eventos que o *script* pode receber e gerar. Para *scripts* responsáveis pela animação, equações de movimento e variáveis envolvidas podem ser definidas. Após essas informações, a geração do *script* é automática.

A ScrLib é dedicada a *scripts* que calculam o movimento e, por isso, está preparada para receber um sistema de equações; entretanto, não oferece nenhum mecanismo adicional, como por exemplo, armazenagem de valores ou comparações. No caso de *script* com outro propósito ou que não utiliza apenas um sistema de equações para cálculo de movimento, a ScrLib pode ser utilizada para esquematizar o *script*, deixando o processamento a cargo do animador.

Nesta seção, todo o funcionamento da ScrLib será detalhado. Na Seção 4.4 será mostrado como esta interface foi instanciada a fim de dedicá-la à geração de *scripts* para cálculo de movimento em função do tempo.

É importante notar que um *script* definido com ScrLib não implica na existência de um painel de controle. Portanto, o ambiente também é capaz de gerar animações sem a interatividade oferecida por um painel de controle.

4.2.1. Especificação das classes

As classes que implementam a ScrLib foram divididas em classes de painel, classes de modelo e uma classe de gerenciamento. As classes de painel são responsáveis pela interface gráfica e as de modelo, por armazenar os componentes do *script*, criados pelo animador. A função da classe de gerenciamento é gerar o arquivo correspondente ao *script* VRML com base nos dados das classes de modelo.

As classes de modelo correspondem aos componentes que podem compor um *script* dedicado a cálculo do movimento, ou seja, campos, eventos, variáveis e equações. Essas classes, junto a seus atributos e herança, são vistas no diagrama da Figura 4.1 e apresentadas a seguir.

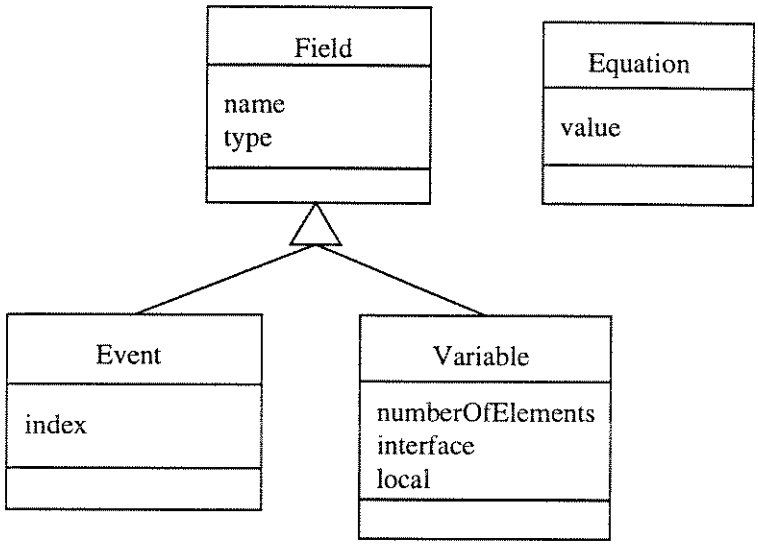

Figura 4.1: Classes de modelo para a ScrLib

A classe Field, que corresponde a um `field` definido para o *script*, tem dois atributos: name e type. As classes Event e Variable são definidas como subclasses de Field e, portanto, herdam seus atributos.

Event, que corresponde a um `eventIn` ou `eventOut`, é definida com o atributo index, que representa uma referência para o componente associado: `field` ou variável. Esta associação entre componentes será detalhada na próxima seção.

Variable, que corresponde a uma variável usada no *script*, apresenta três atributos: numberOfElements, interface e local. Se a variável for definida como cadeia de caracteres, numberOfElements representa o número de elementos; senão é definida com o valor –1. Para que uma variável represente um parâmetro de equações, o atributo interface é definido. O atributo local determina se o escopo da variável será local ao cálculo do movimento ou não.

A classe Equation é definida somente com o atributo value, que representa uma equação definida para compor o *script*.

Para um *script*, o animador pode definir vários objetos das classes mostradas na Figura 4.1; por isso, outras classes são necessárias para a armazenagem de todos os

objetos definidos. Como mostra a Figura 4.2, essas classes são derivadas da classe Vector de Java e, portanto, representam um vetor de objetos. Cada uma delas é responsável pela armazenagem de objetos de uma das classes apresentadas na Figura 4.1.

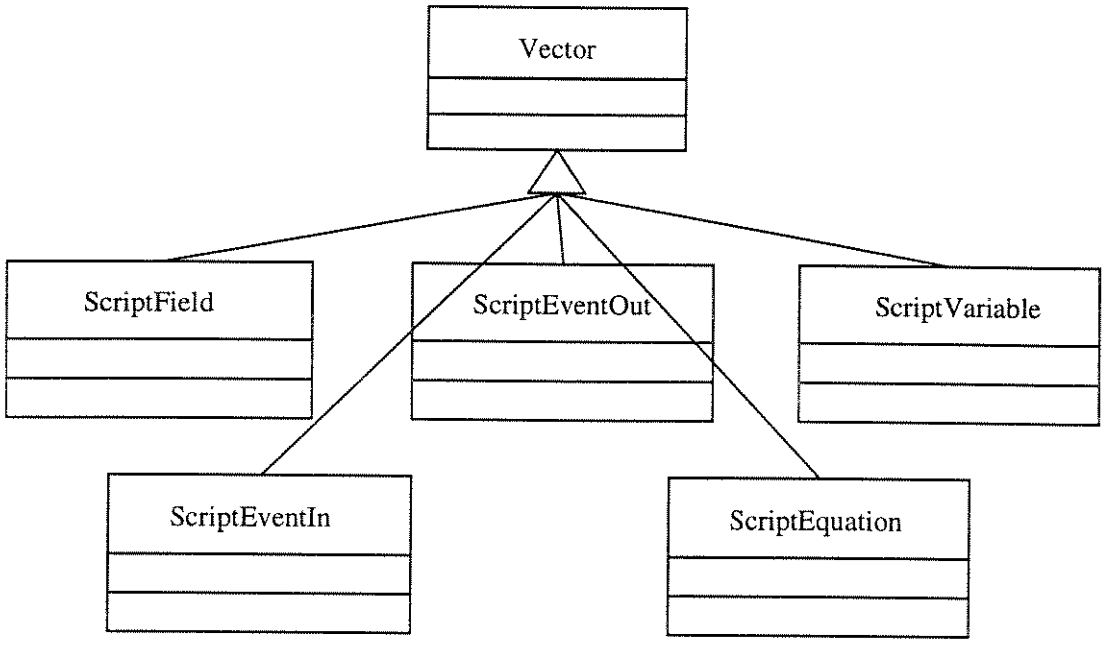

Figura 4.2: Classes de modelo para armazenagem de objetos

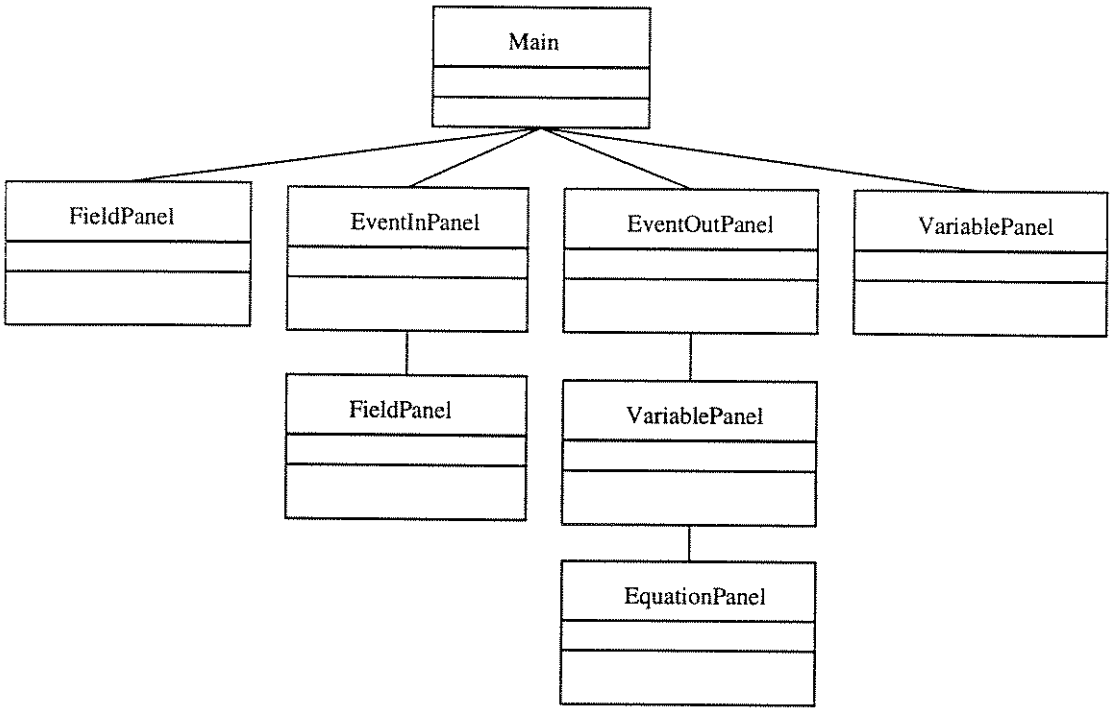

Figura 4.3: Classes de painel que compõem a ScrLib

Os métodos definidos para as classes de modelo se resumem, basicamente, a *gets* e *sets* para os atributos que são privados; outros são os herdados da classe Vector.

As classes de painel que compõe a ScrLib são mostradas, na Figura 4.3, como uma árvore, indicando a seqüência em que os painéis podem ser exibidos, de acordo com ações do animador. O painel principal é exibido logo após a informação do nome com que o *script* será gerado. A partir do painel principal, pode-se obter painéis para incluir `field`, `eventIn`, `eventOut` ou variável. Como o objetivo principal da ScrLib é definir *scripts* para produzir animação, as equações podem ser inseridas somente a partir de um `eventOut` associado com uma variável e responsável por gerar posições.

A classe de gerenciamento é responsável por exibir os painéis e criar instâncias das classes da Figura 4.2 para compor seus atributos. A Figura 4.4 mostra o relacionamento da classe de gerenciamento com as classes de modelo e painel. Quando a classe de gerenciamento é instanciada, os objetos das classes da Figura 4.2 são criados. As classes de painel são instanciadas pela classe de gerenciamento, de acordo com as ações do animador.

Cada classe de painel tem um relacionamento com uma das classes de modelo da Figura 4.1. Todas as classes de painel têm um método definido para processamento a partir da interação com animador. É neste método que, dependendo da ação do animador, objetos de modelo são criados e a classe de gerenciamento é informada de que painel deve ser exibido. Para exemplificar, é considerado um painel para incluir um `field`, contendo um botão para a confirmação. Se este botão é pressionado, o método citado é invocado e uma instância da classe Field da Figura 4.1 é criada. Esta instância é enviada para o objeto de gerenciamento, que adiciona ao vetor correspondente (ScriptField) e decide que painel deve exibir, tendo como base as informações deste objeto recebido.

Quando o animador confirmar a criação do *script*, a classe de gerenciamento, através do método createFile, é responsável por gerar o arquivo. Para isso, deve conhecer o formato de um *script* e utilizar métodos para escrever os objetos de modelo em arquivo.

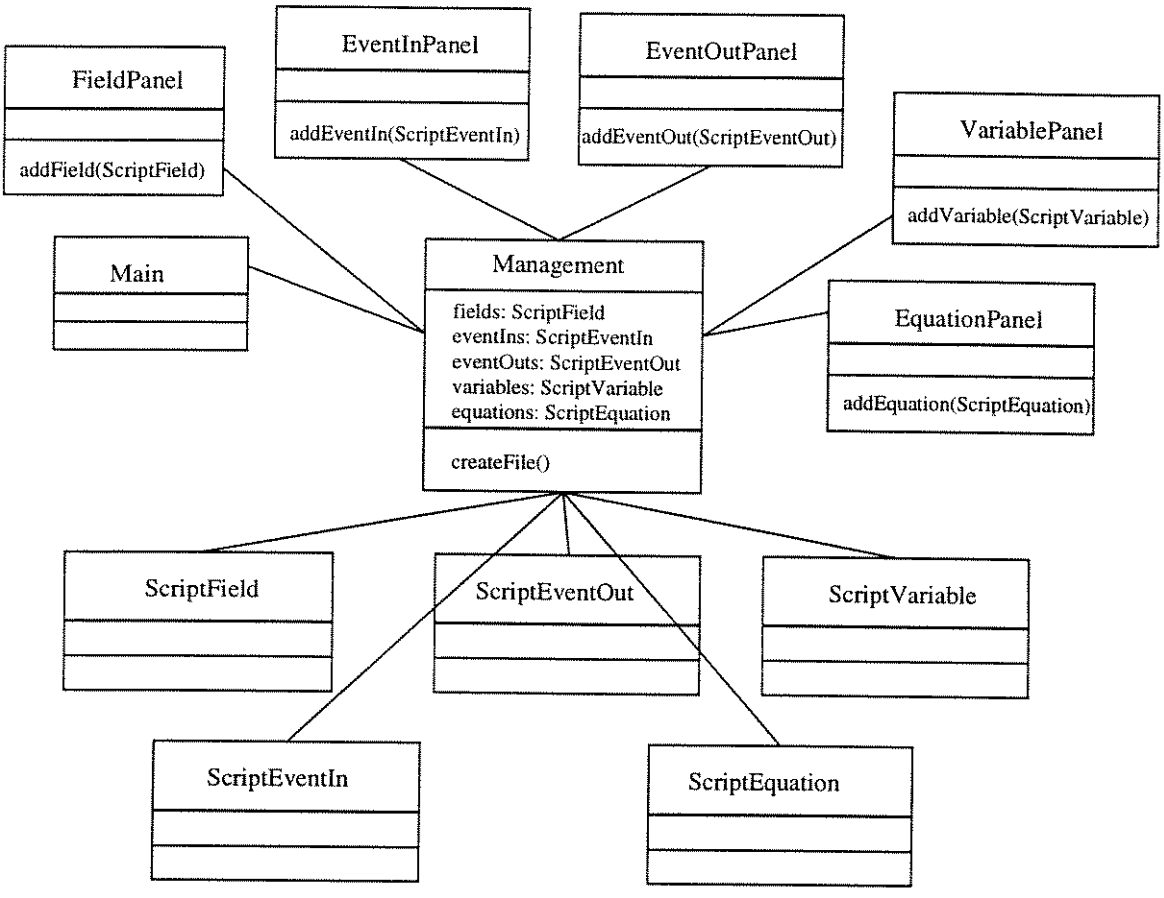

Figura 4.4: Relacionamento entre classes de modelo, painel e gerenciamento

4.2.2. Criando um *script*

A ScrLib terá sua funcionalidade apresentada para a criação de um *script*, que calcula o movimento retilíneo uniforme, para ser associado à cena VRML, que define uma esfera.

A janela principal da interface, com as possíveis opções para definição de um *script* qualquer, pode ser vista na Figura 4.5. Diretamente, pode-se inserir `field`, `eventIn`, `eventOut` e variáveis.

No movimento retilíneo uniforme é necessária a definição de um `eventIn` para o *script* receber instantes de tempo e um `eventOut` para gerar a posição, que é atribuída à esfera da cena VRML.

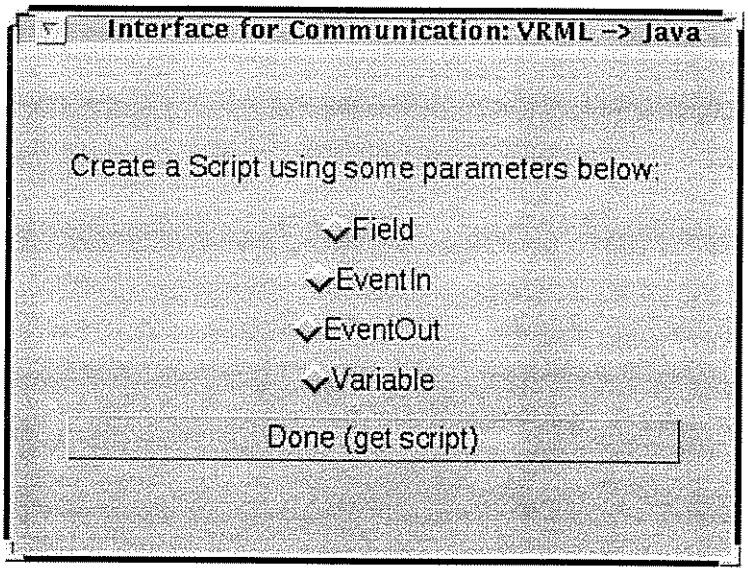

Figura 4.5: Janela principal para geração de *script*

Para incluir um `eventIn` a opção *EventIn* da janela principal deve ser selecionada. Assim, a janela da Figura 4.6 é exibida, solicitando o tipo e o nome do evento. O tipo deve ser um dos pré-definidos pela especificação VRML, e pode ser selecionado de uma lista (Figura 4.7), que é exibida se a opção for ativada. A mesma lista também aparece para o tipo de `field` e `eventOut`, já que ambos devem pertencer ao mesmo conjunto de tipos.

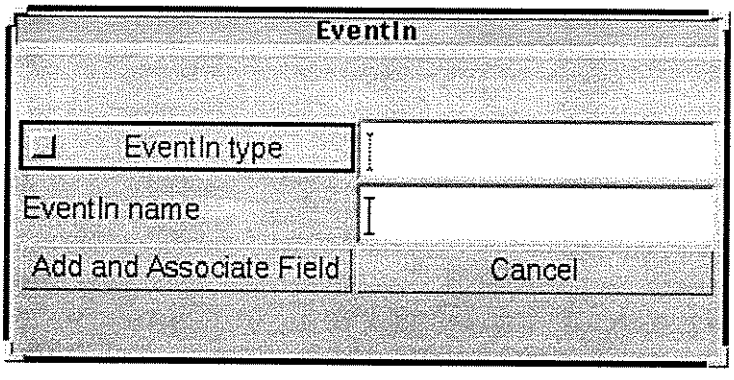

Figura 4.6: Inserindo `eventIn`

Para o evento de tempo, no exemplo, o tipo deve ser `SFFloat` e o nome dado é *currentTime*.

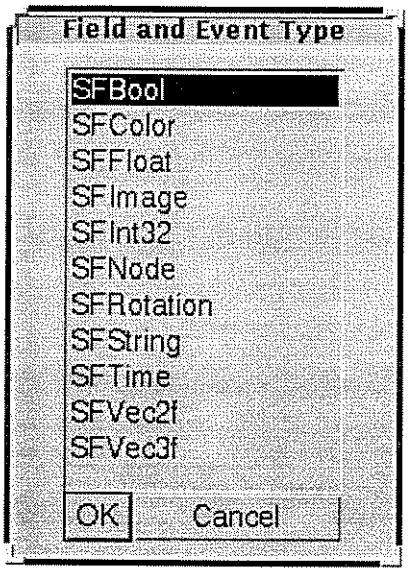

Figura 4.7: Tipos de `eventIn`, `field` e `eventOut`

Cada `eventIn` incluído deve ser associado a um `field`, para que o valor do evento possa ser manipulado dentro do *script* (esta associação significa: `field` = valor do `eventIn`).

Quando a inclusão for confirmada, pelo botão *Add and Associate Field*, a janela para `field`, que aparece na Figura 4.8, é mostrada com o mesmo tipo do evento, neste caso `SFFloat`. No entanto, para um `field` associado a `eventIn`, o botão *Cancel* não é exibido para obrigar que o animador declare um evento com tratamento consistente. O nome do `field`, no exemplo, é *time*.

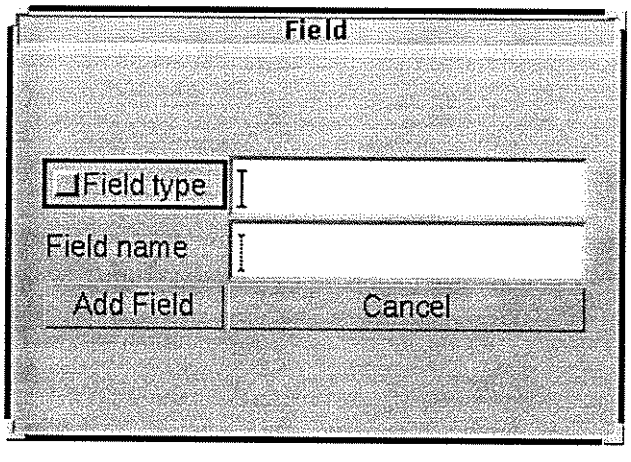

Figura 4.8: Inserindo `field`

Um `field` no *script* não implica em nenhuma outra definição, mas definido sem uma associação com um `eventIn` não faz muito sentido.

A partir da janela principal pode-se inserir um `eventOut` para a geração da posição, variáveis associadas e equações de movimento sucessivamente. Para isso, deve-se interagir com a janela mostrada na Figura 4.9, exibida quando o animador seleciona a opção *EventOut.*

Para um `eventOut`, o tipo e o nome devem ser informados da mesma maneira que para `eventIn`. A diferença do procedimento está no fato de que um `eventOut` deve ser associado a uma variável e não a um `field`. Esta variável é usada para efetuar algum processamento e retornar o resultado, na forma do `eventOut` associado, afetando o comportamento de um objeto da cena VRML.

Para a geração da posição, o evento deve ser do tipo `SFVec3f`, que define um vetor, correspondendo a uma coordenada cartesiana tridimensional, e o nome dado é *translation.*

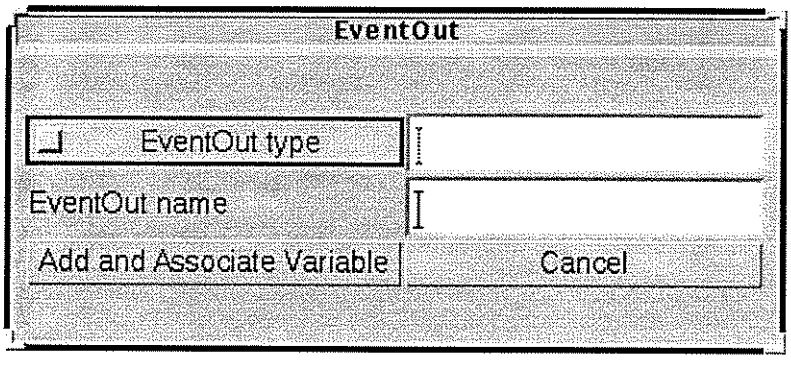

Figura 4.9: Inserindo `eventOut`

Quando a inclusão de um `eventOut` for confirmada, através do botão *Add and Associate Variable*, a janela da Figura 4.10 é exibida para inclusão da variável associada. Nesta janela deve-se informar tipo e nome da variável. Os tipos possíveis são os pré-definidos pela linguagem Java. Como para eventos e `fields`, eles são mostrados em uma lista para seleção única, vista na Figura 4.11, quando a opção for selecionada. Além do nome e tipo, ainda existem três opções. A primeira, *Array*, indica que a variável a ser declarada pode ser uma cadeia de caracteres e, se for, o campo *Number of elements* é considerado. A segunda, *Local to calculate the movement*, é relacionada ao escopo da variável. Ela pode ser local ao método responsável pelo processamento de `eventOuts` ou ser visível por todos os outros. E a última, *Value controlled in the user interface*, indica que o valor dessa variável pode corresponder a um parâmetro, que é controlado pelo usuário final no painel de controle.

A variável, associada com o evento que gera a posição, é responsável pelo cálculo do movimento. Desta forma, na Figura 4.10, a variável deve ser definida como um *array* de três elementos do tipo `int`, onde cada um é responsável por uma coordenada do evento *translation* gerado. As outras opções não são selecionadas.

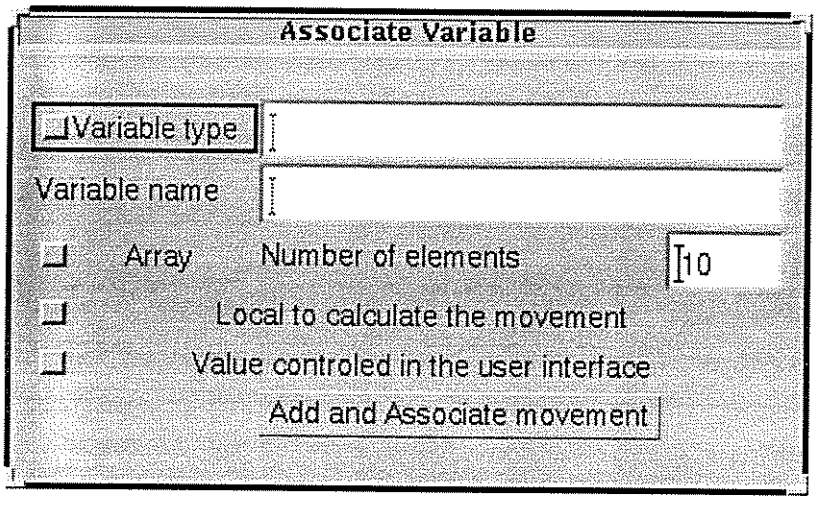

Figura 4.10: Associando variável

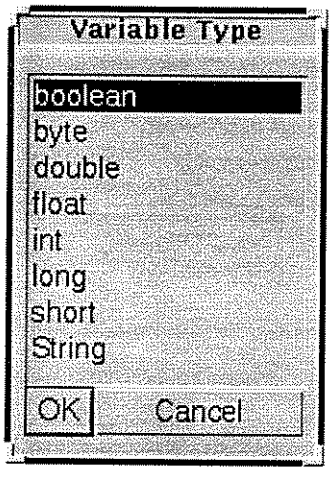

Figura 4.11: Tipo de variável

Se a variável é associada a um `eventOut` e definida como um *array* de no mínimo três elementos, tem-se a possibilidade de incluir equações, logo que o botão *Add and Associate movement* é pressionado, como é o caso do exemplo sendo apresentado. Isto indica que a variável sendo incluída pode ser associada a um tipo de movimento, onde para cada elemento do *array* é associada uma equação que define a posição de um objeto VRML em uma dimensão (são consideradas até três dimensões; por isso, um número de elementos maior do que três não tem efeito).

A Figura 4.12 mostra a janela para a inclusão de equações. Nos campos *x(t)*, *y(t)* e *z(t)* devem ser informadas as equações do movimento em função do tempo (*t* é a variável correspondente ao tempo). Estas equações podem ser formadas por outras, definidas através do campo *Equations*. Neste campo, cada informação deve ser seguida de <*Enter*> e as variáveis utilizadas devem ser definidas através do menu principal.

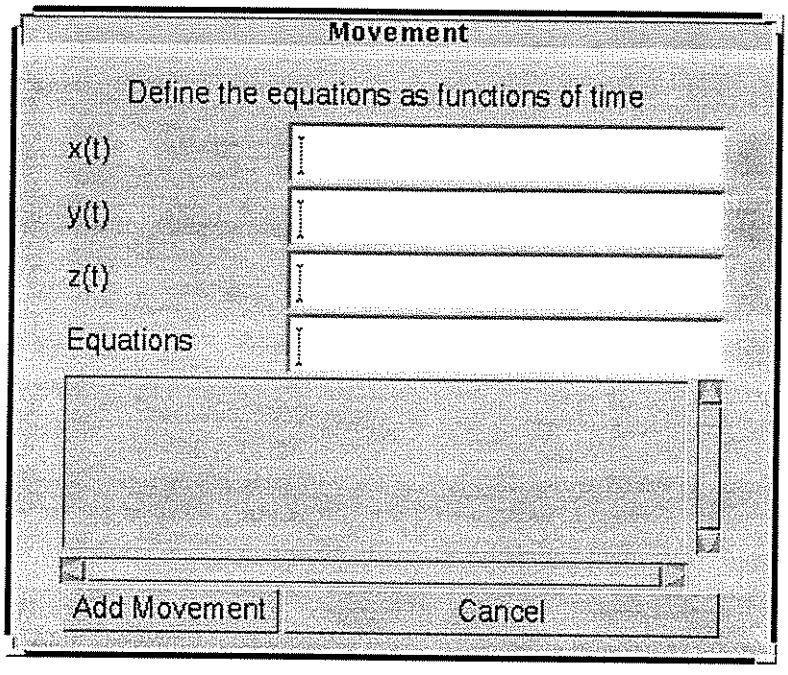

Figura 4.12: Inserindo equações

Para definir o movimento retilíneo uniforme, a equação mostrada na Seção 4.1 deve ser atribuída a cada um dos campos *x(t)*, *y(t)* e *z(t)*. Neste exemplo, o movimento será restrito ao plano *xy*; a velocidade será nula no eixo *z* e única nos eixos *x* e *y*. A posição inicial será definida para cada equação. Assim, as equações são apresentadas:

$$x(t) = x_0 + v.t$$
$$y(t) = y_0 + v.t$$
$$z(t) = z_0$$

Desta forma, existem quatro variáveis que precisam ser definidas: x_0, y_0, z_0 e v. A variável t é pré-definida e corresponde ao evento de tempo recebido pelo *script*; portanto, não precisa ser definida. O campo *Equations* não é utilizado, pois nenhuma outra equação precisa ser definida. Ele seria utilizado quando as equações dos eixos fossem compostas por variáveis correspondentes a outras equações.

Pode-se inserir variáveis envolvidas nas equações diretamente da janela principal; neste caso a janela da Figura 4.10 é exibida com a adição de um botão

Cancel. A funcionalidade da janela é a mesma descrita anteriormente, a diferença está na ação que é executada após a confirmação de inclusão (que é feita através do botão *Add and Associate movement*).

Todas as variáveis definidas podem ser constantes ou parâmetros de controle. Se forem constantes, o *script* não depende de um painel de controle e, portanto, o usuário não pode interagir com o movimento. A opção será por parâmetros de controle. Quando a inclusão dessas variáveis for confirmada, através do botão *Add and Associate movement*, o *script* é preparado para solicitar e receber valores do painel de controle, correspondentes às variáveis definidas.

Quando o botão *Done (get script)* da janela principal (Figura 4.5) for pressionado, o *script* será gerado pela ScrLib. O esquema geral do *script* obtido é mostrado na Figura 4.13. O conteúdo mais completo do arquivo gerado automaticamente é apresentado no Apêndice III.

```
public class Smovement extends Script {
  private SFFloat currentTimer;
  float[] translation;
  private SFVec3f position;
  ...
public void initialize() {
     translation = new float[3];
     currentTimer = (SFFloat) getField("currentTimer");
     position = (SFVec3f) getEventOut("position");
     ...
}
public void processEvent(Event e) {
  if (e.getName().equals("timerIn"))
    currentTimer.setValue((ConstSFFloat)e.getValue());
}

public void eventsProcessed() {
int x0=0, v=0, y0=0, z0=0;
  ... Obtém valores do painel de controle
  translation[0] = x0+v*(float)currentTimer.getValue() ;
  translation[1] = y0+v*(float)currentTimer.getValue() ;
  translation[2] = z0;
  position.setValue(translation);
}
```

Figura 4.13: *Script* para o cálculo do movimento retilíneo

No nível 1 da implementação (Figura 3.6), para associar o *script* à esfera da cena VRML, o animador deve, manualmente, alterar o arquivo VRML. O nodo `Shape` deve ser agrupado por um `Transform`, para que a esfera possa receber o movimento. Um nodo `Script` deve ser inserido, com o campo `url` referenciando o arquivo do *script*

gerado, com `field` *time*, `eventIn` *currentTime* e `eventOut` *translation*. Um nodo `TimeSensor` deve prover ao *script* eventos de tempo. Roteamentos devem ser introduzidos, do `TimeSensor` para o `Script`, e do `Script` para o `Transform`, produzindo a animação da esfera.

Na próxima seção, o painel de controle será definido para enviar os parâmetros utilizados no *script*, correspondendo às variáveis x_0, y_0, z_0 e v.

4.3. IULib: Interface para geração do painel de controle

Esta seção é dedicada à implementação da interface chamada IULib, que procura facilitar a criação de painéis de controle e capacitar sua conexão a um *script* VRML. Um painel de controle é responsável pela interação com o usuário final e envio de parâmetros ao *script*. Esses parâmetros têm influência sobre a animação, pois são utilizados nas equações que calculam o movimento.

Para criar um painel de controle, o animador escolhe os componentes gráficos que desejar para o compor, indicando os que corresponderão a parâmetros da animação. O animador dispõe de um conjunto de componentes gráficos, que proporciona interação amigável e, principalmente, possibilita a entrada de parâmetros para a animação. A IULib gera código para todo o tratamento necessário aos componentes gráficos escolhidos.

Além da interação com o usuário final, a IULib deve ser capaz de gerar código para enviar valores de componentes gráficos, determinados na criação do painel de controle, de maneira confiável e rápida, para o *script* VRML.

4.3.1. Especificação de classes

As classes que implementam a IULib foram divididas em classes de painel, classes de modelo e uma classe de gerenciamento, da mesma forma que para ScrLib (Seção 4.2.1). A idéia do projeto é a mesma, a diferença está no propósito das classes de modelo e gerenciamento. As classes de modelo são responsáveis por armazenar os componentes do painel de controle, criados pelo animador. A função da classe de gerenciamento é criar objetos de modelo e gerar o arquivo correspondente ao painel de controle com base nos dados desses objetos.

As classes de modelo correspondem aos componentes gráficos que podem compor um painel de controle. Os componentes considerados são os oferecidos pela linguagem Java. Essas classes e seus atributos são apresentados na Figura 4.14. De maneira geral:

- alguns atributos correspondem aos atributos dos componentes da linguagem, tais como: name, text, horizontal, maxValue, minValue, current, overall, width, height, editable, size, currentValue e vectorOfCheckbox;
- outros são relacionados à forma de disposição no painel de controle: last e fill;
- create informa se o rótulo está associado a uma barra de rolagem, indicando seu valor;
- associated indica que o componente tem um outro associado, o que pode acontecer entre um grupo de opções e uma barra de rolagem;
- send indica que o componente pode ter seu valor enviado para o *script* VRML. Objetos da classe Label têm seu valor enviado ao *script* somente se representar o valor corrente de uma barra de rolagem.

O animador pode escolher vários objetos das classes da Figura 4.14 para compor um painel de controle; por isso, outras classes, uma para cada tipo de objeto, são definidas para armazená-los. Essas classes são derivadas da classe Vector de Java e, portanto, representam um vetor de objetos. Elas são mostradas no relacionamento com a classe de gerenciamento posteriormente, na Figura 4.16.

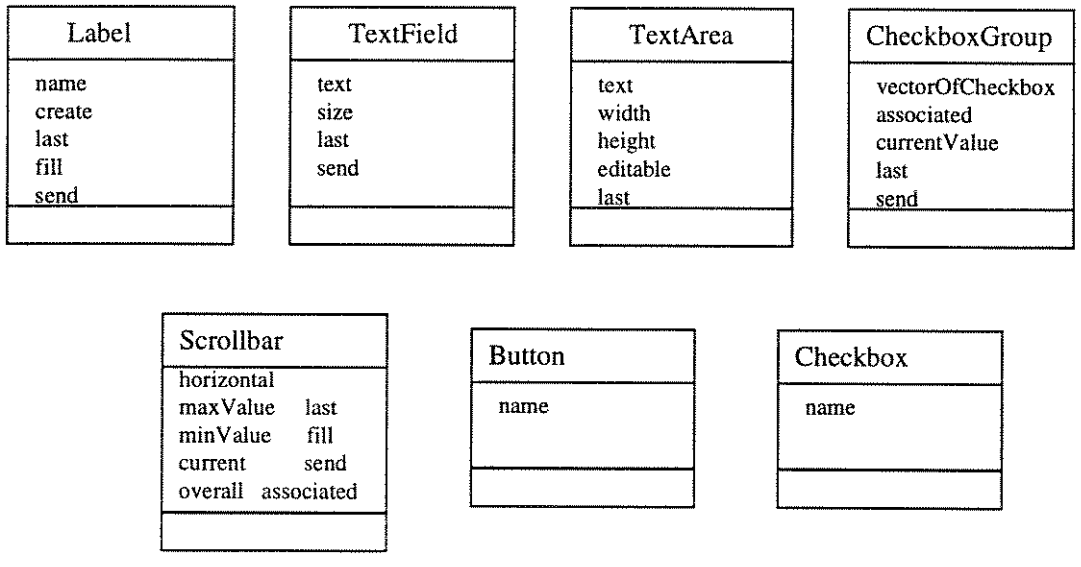

Figura 4.14: Classes de modelo para a IULib

As classes de painel são mostradas na Figura 4.15, onde pode ser verificada a ordem em que os painéis são exibidos, de acordo com ações do animador. Existe uma classe de painel para cada classe de modelo da Figura 4.14.

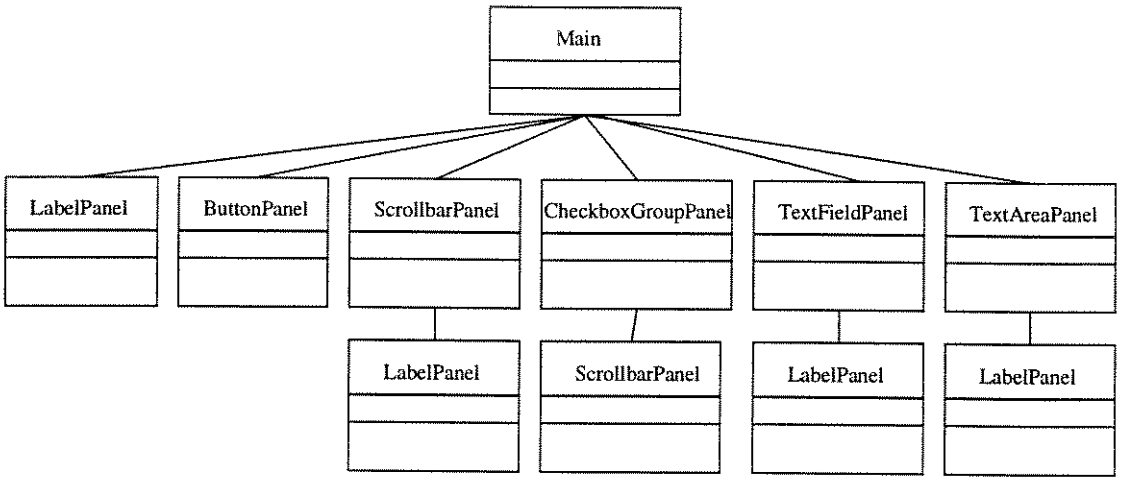

Figura 4.15: Classes de painel para a IULib

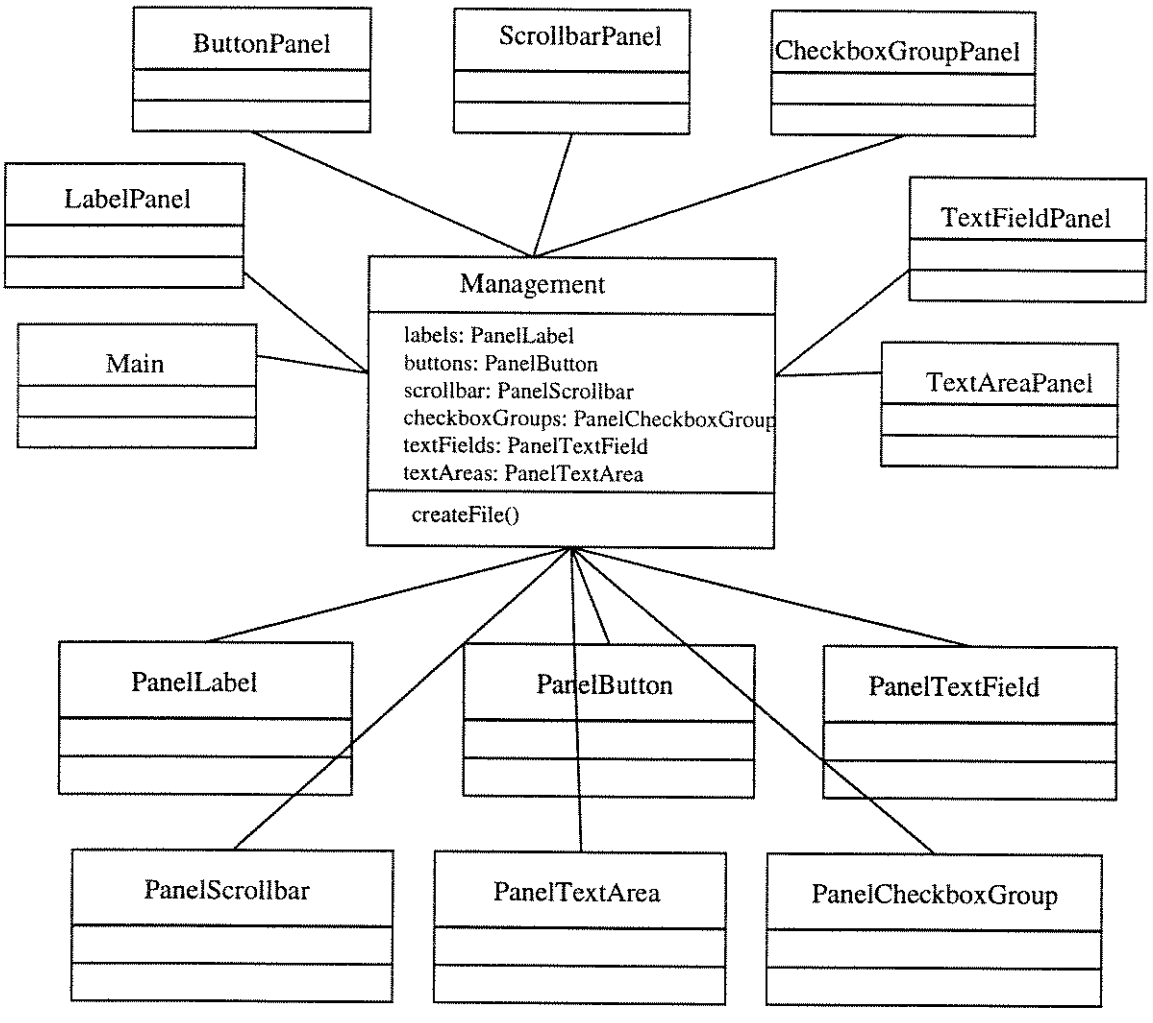

Figura 4.16: Relacionamento entre classes de painel, modelo e gerenciamento

A classe de gerenciamento mantém instâncias dos vetores de objetos de modelo. Ela também deve exibir os painéis. Cada classe de painel, de acordo com a ação do animador, cria um objeto de uma classe de modelo da Figura 4.14 e envia para a classe de gerenciamento, que adiciona ao vetor correspondente. É baseado neste objeto recebido que um outro painel é exibido. O relacionamento das classes de painel e modelo com a classe de gerenciamento pode ser verificado na Figura 4.16.

Quando o animador confirma a criação do painel de controle, a classe de gerenciamento utiliza os objetos de modelo para gerar o arquivo Java. Para isso, esta classe deve conhecer o formato de uma aplicação Java responsável pela interação com o usuário e envio de parâmetros a um *script* VRML.

4.3.2. Criando um painel de controle

A funcionalidade da IULib será apresentada para a definição de um painel de controle (Figura 4.24), usado no exemplo de movimento retilíneo uniforme, cujos os parâmetros foram definidos na Seção 4.2.2: velocidade e posições iniciais para os eixos x, y e z.

Para utilizar esta interface o animador deve atribuir um nome ao arquivo que conterá o painel de controle gerado, na primeira janela exibida (Figura 4.17).

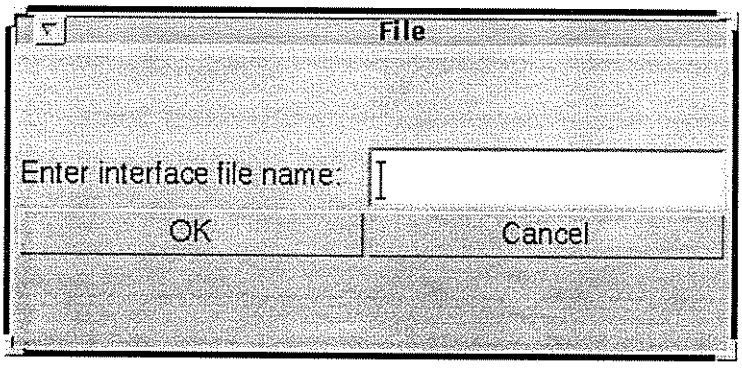

Figura 4.17: Informa nome para o painel de controle a ser gerado

Com a IULib o animador pode construir um painel de controle utilizando e combinando vários componentes gráficos, definidos pela linguagem Java, tais como: rótulos, botões, grupos de opções, barras de rolagem, campos e áreas de texto. A janela principal, mostrada na Figura 4.18, oferece essas opções. Uma nova janela é exibida para cada uma das opções, onde é necessária a informação de dados relacionados ao componente sendo adicionado.

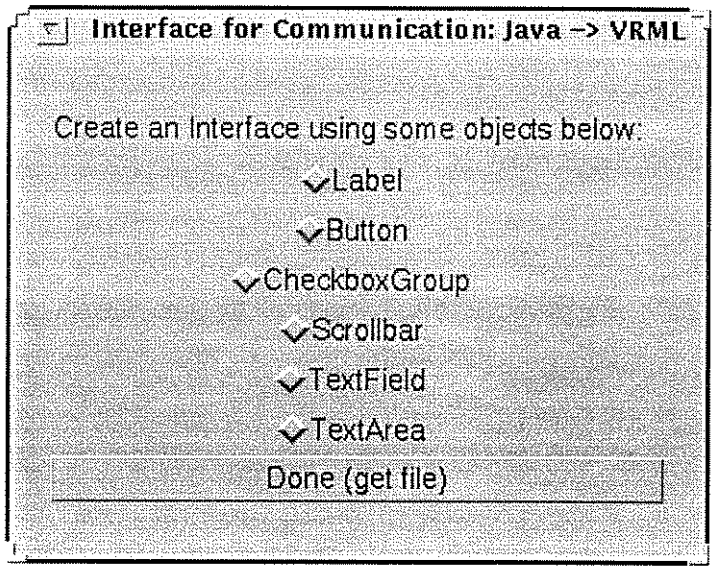

Figura 4.18: Opções da interface

Os parâmetros de controle a serem enviados à animação podem ser associados a valores de grupos de opções, barras de rolagem e campos de texto, através da opção *Send to VRML* (Figura 4.20) existente na janela correspondente. Estes componentes são utilizados porque são associados com valores numéricos de forma mais intuitiva, facilitando a interação do usuário final.

Um painel de controle pode ser definido de várias maneiras para o mesmo *script*, pois os componentes gráficos que o compõem dependem somente da escolha do animador, que deve considerar a melhor forma para interação com o usuário final.

A ordem da definição dos componentes gráficos é a mesma que aparecerá no painel de controle, e que seus valores, representando parâmetros da animação, serão enviados ao *script*.

Para o painel que controlará variáveis do movimento retilíneo uniforme, demonstrando a utilização de vários componentes gráficos, as opções foram as seguintes:

- campo de texto para a posição inicial no eixo x;
- grupo de opções e barra de rolagem para a posição inicial no eixo y e z;
- barra de rolagem para a velocidade;
- inserir rótulos para todos os campos e como título.

Um rótulo pode ser inserido como título informativo ou associado a outro componente, como barras de rolagem, campos e áreas de texto, para descrever o parâmetro que deve ser informado. Para ambos os casos, a janela da Figura 4.19 é exibida.

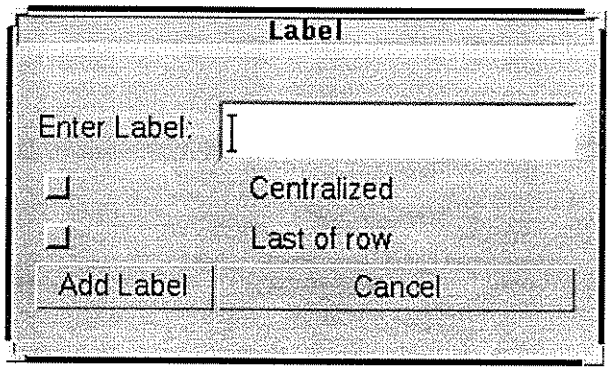

Figura 4.19: Inserindo rótulos

Para inserir o título para o painel, a opção *Label* da janela principal deve ser selecionada. Ela solicita uma descrição, que pode ser "Movimento retilíneo uniforme", e apresenta duas opções relacionadas com o *layout*, que devem ser selecionadas: *Centralized* para centralizar o rótulo, e *Last of row* para o rótulo ser o último componente da linha. Esta última opção estará presente em todas as outras janelas com o mesmo propósito; pode-se ter qualquer componente como último da linha, por isso, será omitido no detalhamento que segue.

Considerando a ordem de apresentação dos componentes no painel de controle, após o título do painel de controle, os parâmetros de controle são inseridos a seguir.

A posição inicial no eixo x será definida como um campo de texto. A Figura 4.20 mostra a janela exibida para a inclusão de campos de texto, obtida a partir da janela principal. São dois os campos requeridos: o conteúdo inicial e o tamanho com quais o campo será apresentado. O conteúdo informado neste campo é definido como parâmetro da animação, habilitando a opção *Send to VRML*. Vale lembrar que somente valores numéricos fazem sentido ao *script* VRML. Pode-se inserir um rótulo explicativo para o campo pressionando-se o botão *Label*, neste caso "Posição inicial (x)".

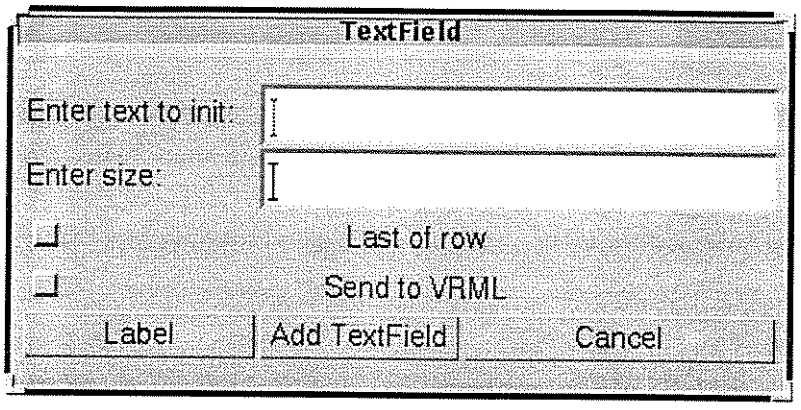

Figura 4.20: Inserido campos de texto

As posições iniciais sobre os eixos y e z serão definidas para assumir o mesmo conjunto de valores, representados por uma mesma barra de rolagem. Isto é possível porque a IULib suporta a definição de um grupo de opções associado com uma barra de rolagem. As opções são variáveis que, quando selecionadas, podem ter seus valores determinados na barra de rolagem.

Esta associação de um grupo de opções com uma barra de rolagem, usada como definição de parâmetros para a animação, é adequada quando há alguma relação ou semelhança entre os tipos de dados; como outro exemplo, o raio maior e o menor de uma elipse. Deste modo, os parâmetros, que podem ser agrupados, são definidos sobre o mesmo intervalo de valores e determinados pela única barra de rolagem.

Na janela da Figura 4.21, que é obtida com seleção da opção *ChekboxGroup* da janela principal, pode-se inserir um grupo de opções. Para isso, deve ser informado, no campo de texto, o conteúdo que estará descrevendo cada opção. Pode-se definir quantos sejam necessários. Uma opção informada deve ser confirmada pela tecla *<Enter>* pois, à medida que o animador procede, os nomes vão aparecendo na lista. Esses dados também podem ser alterados. Para isso, basta selecionar um item da lista e informar o novo conteúdo no campo de texto. As opções informadas nesta janela são: "Posição inicial (y)" e "Posição inicial (z)".

O conteúdo que representa as opções pode ser enviado ao *script* VRML através da opção *Send to VRML*. Para que o conteúdo de uma opção tenha efeito como parâmetro da animação é necessário que seja numérico, pois será o valor de variáveis de equações. Há dois modos para se definir parâmetros da animação usando um grupo de opções: definir números como conteúdo das opções ou associar uma barra de rolagem, como é o caso do exemplo. No primeiro, o grupo de opções corresponde a somente um

parâmetro. Este tipo de definição é indicada para casos onde o parâmetro só pode ser alguns poucos valores pré-definidos. O valor enviado ao *script* é o número da opção selecionada pelo usuário final da animação.

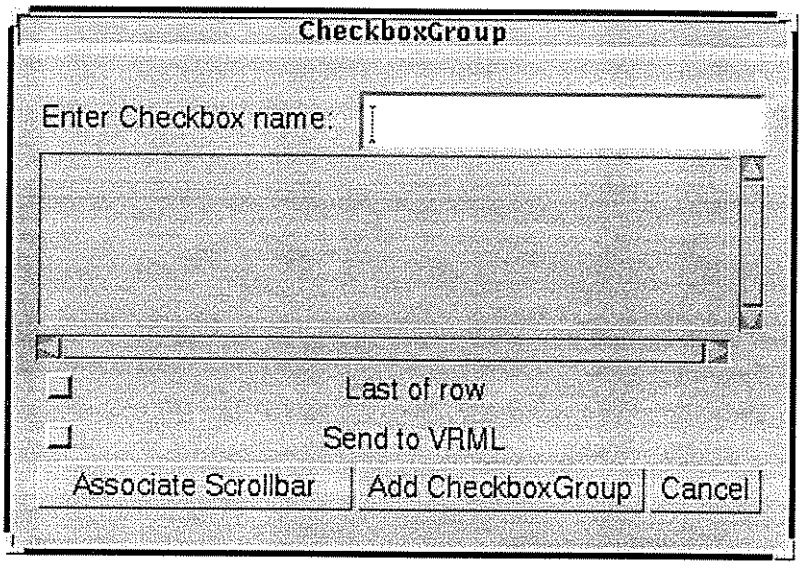

Figura 4.21: Inserindo um grupo de opções

Para associar uma barra de rolagem ao grupo de opções, o botão *Associate Scrollbar* deve ser pressionado e, assim, a janela da Figura 4.22 é exibida. Todas as opções são associadas a uma barra de rolagem, mas cada uma tem seu valor, que é mostrado logo ao lado do nome. O valor alterado na barra de rolagem corresponde à opção selecionada. No caso da opção *Send to VRML* (na Figura 4.21) estar selecionada, o número de parâmetros a ser enviado ao *script* é igual ao número de opções definidas. Os valores enviados são os que ficam ao lado do nome.

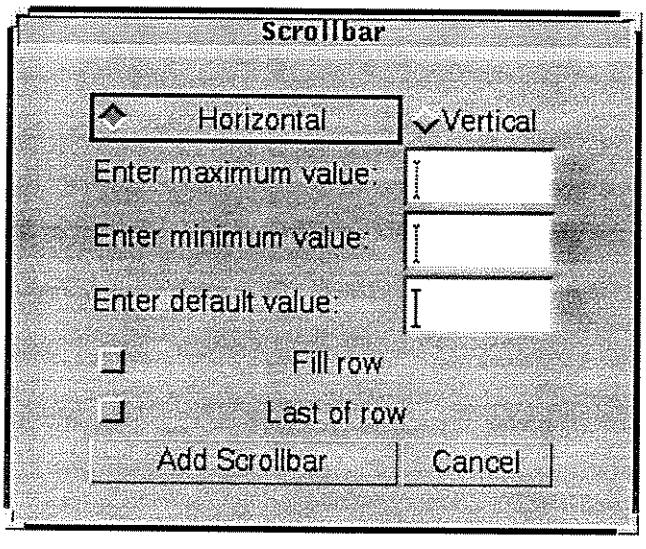

Figura 4.22: Associando barra de rolagem a um grupo de opções

Para inserir uma barra de rolagem é necessário informar o sentido (horizontal ou vertical), um intervalo de valores possíveis (valor máximo e mínimo) e o valor inicial. A opção *Fill row* é relacionada ao *layout*, indica se a barra de rolagem terá seu tamanho proporcional ou ocupará toda a linha.

Uma barra de rolagem também pode ser adicionada ao painel a partir da janela principal, quando a opção *Scrollbar* for selecionada. A janela da Figura 4.23 a ser exibida é semelhante à da Figura 4.22, pois ambas referem-se ao mesmo componente gráfico, só que acrescenta a opção *Send to VRML* e o botão *Label*. Estes dois itens adicionais são redundantes na definição de barra de rolagem associada a um grupo de opções, pois como várias opções referem-se a mesma barra de rolagem, são os valores das opções que devem ser enviados ao *script* e o próprio nome da opção tem a função informativa.

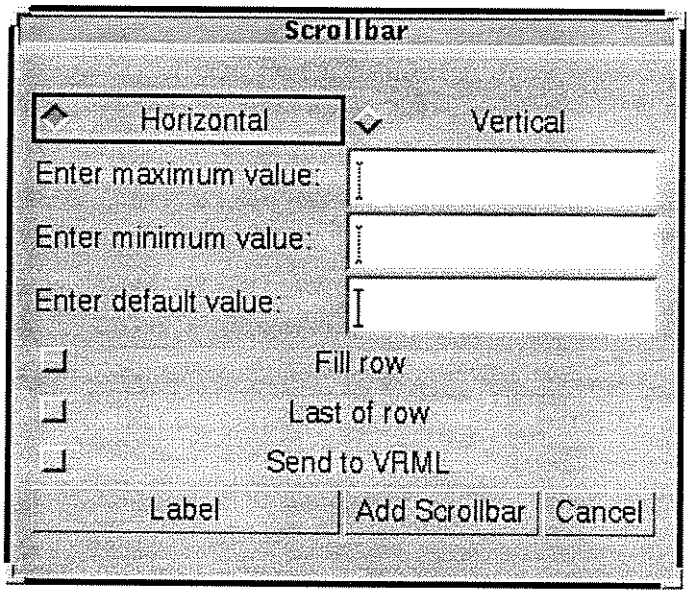

Figura 4.23: Inserindo barra de rolagem

Partindo da janela principal, a definição da barra de rolagem é efetuada para a velocidade. Esta definição é acompanhada por um rótulo que indica o valor corrente. A opção *Send to VRML* é selecionada, e é o valor deste rótulo que será enviado ao *script* VRML. Um outro rótulo "Velocidade" é inserido, através do botão *Label*, com o objetivo de descrever o parâmetro que a barra de rolagem representa. Neste caso, a mesma janela da Figura 4.19 é exibida.

Existem outras opções na janela principal, *Button* e *TextArea*, não exemplificadas nesta seção. Botões não têm muita utilidade em um painel de controle,

sem a intervenção do animador no código. Áreas de texto são úteis no sentido informativo; poderiam ser inseridas no painel de controle para auxiliar o usuário final. Para mais detalhes sobre essas opções, o Apêndice II pode ser consultado.

Em todas as janelas vistas, com exceção da principal, pode-se notar a existência de dois botões: um para confirmar a operação de inclusão e outro para cancelar, sempre nesta ordem. Na janela principal há somente um botão, que depois de incluir todos os componentes gráficos desejados, o animador deve pressionar para concluir e obter o arquivo gerado. O painel de controle obtido para o exemplo do movimento retilíneo uniforme é mostrado na Figura 4.24.

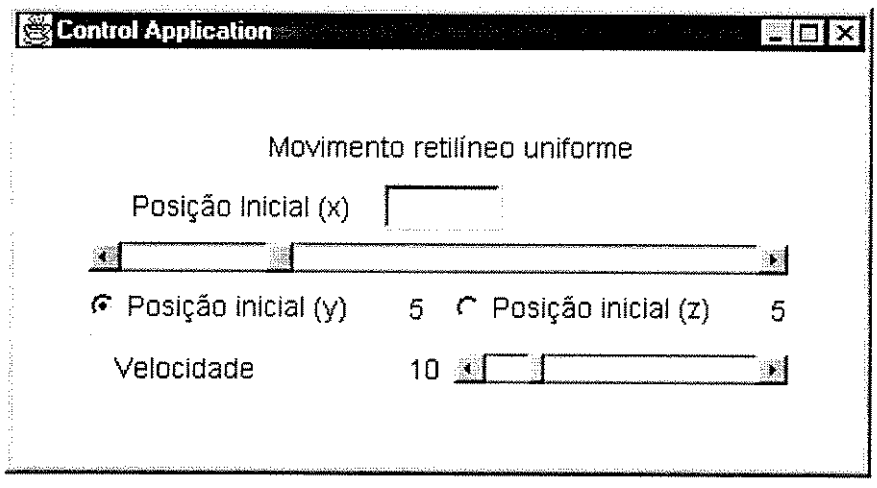

Figura 4.24: Painel de controle para a animação do movimento retilíneo

4.4. Integração IULib e ScrLib

A redundância introduzida pela interdependência entre as interfaces IULib e ScrLib, já comentada na Seção 3.6, é eliminada com a integração das interfaces. Desta forma, o animador, ao invés de informar quais componentes gráficos serão parâmetros da animação e quais variáveis da equação serão controladas via painel de controle, atua de forma única na definição de parâmetros.

As interfaces foram adaptadas para que a informação dos parâmetros da animação seja dada apenas uma vez, tornando mais conveniente para o animador e eliminando inconsistências, ou seja, garantindo que o número de componentes gráficos enviando valores para o *script* seja igual ao número de valores sendo recebidos.

A opção na integração das interfaces foi manter a ScrLib como interface principal, introduzindo a possibilidade de criação de componentes gráficos para o painel

de controle, no menu principal e na definição de variável. Pode-se concluir que, desta forma, o animador pode construir uma animação sem definir um painel de controle, dispensando a facilidade que o ambiente provê para aumentar a interatividade.

A integração da IULib e ScrLib foi uma oportunidade para se tomar decisões, quanto a funcionalidades que poderiam ser adicionadas ao ambiente, a fim de proporcionar maior facilidade ao animador na criação de animações.

Após a validação da criação de *scripts* e painéis de controle, via interfaces, para animações cujos movimentos são dados em função do tempo, algumas ações foram consideradas desnecessárias, tais como a inclusão de eventos e campos para o *script*. Desta forma, foi feita uma especialização da ScrLib, apresentada na Seção 4.2, para que o animador seja responsável somente pela informação das equações de movimento, ficando por conta do ambiente a geração de um *script* com um `eventIn` e um `field` para receber valores de tempo e um `eventOut` para gerar a posição a ser aplicada a um objeto da cena VRML.

Na informação das equações, o animador deveria ser responsável por manter o controle das variáveis definidas. No caso deste controle ser ineficiente, o ambiente gera *scripts* inconsistentes, o que resulta em erros de compilação. O controle de consistência de equações foi introduzido evitando este problema. Para cada equação, todas suas variáveis são identificadas e a informação para suas definições é solicitada. A informação das equações começa pela posição no eixo x, e após todas suas variáveis serem definidas, as posições nos eixos y e z são requeridas, sucessivamente. A implementação consiste em uma pilha de variáveis a serem definidas, inicializada com as variáveis correspondendo às posições nos eixos z, y e x, respectivamente. À medida que variáveis vão sendo informadas, vão sendo adicionadas na pilha. Para a variável no topo da pilha a definição é requerida. Quando a pilha estiver vazia, significa que todas as variáveis foram definidas e o *script* pode ser gerado de forma consistente.

O animador, quando construía uma animação interativa através da ScrLib e IULib, não tinha oportunidade de construir uma nova animação a partir da anterior, ou seja, a reutilização de animações não era possível. Isto, além de dificultar a validação de um tipo de movimento, dificultava a própria utilização do ambiente, pois para cada animação o animador deveria informar todos os dados necessários, tais como: equações e componentes gráficos.

A reutilização de animações é introduzida de duas maneiras: uma biblioteca e o suporte para edição. A biblioteca é constituída por movimentos pré-definidos e validados, que podem ser utilizados para a geração automática de animações interativas. O suporte à edição de animações foi introduzido para que o animador possa validar, de forma mais fácil, o movimento criado. Assim, o animador pode construir uma nova animação modificando uma equação ou trocando um componente gráfico de outra definida anteriormente. Após a validação do movimento, o animador pode tomar a decisão de incluí-lo na biblioteca, como será mostrado na Seção 4.5.

A implementação da funcionalidade de edição é feita através de arquivos de dados. Para toda animação interativa definida, dois arquivos correspondendo a dados do *script* e painel de controle são gerados. Os dados do *script* armazenados são as equações e variáveis; do painel de controle são os componentes gráficos. Os dados são armazenados de uma forma que o ambiente saiba interpretar para que, a partir desses arquivos, novos *scripts* e painéis de controle possam ser construídos.

O ambiente deve, além de permitir definição de movimento, oferecer as opções para reutilização de movimentos via interface gráfica. A próxima seção mostrará a interface final do ambiente, com toda a funcionalidade discutida disponibilizada, para o exemplo do movimento retilíneo uniforme.

4.4.1. Criando uma animação interativa

O ambiente integrado terá sua funcionalidade apresentada, seguindo o mesmo exemplo de animação descrito para IULib e ScrLib, através do qual as facilidades introduzidas poderão ser observadas.

Neste exemplo, as janelas do ambiente são apresentadas na seqüência correta para a criação da animação e com os devidos valores utilizados. Todo o processo de criação será explicado, mas nem todas as janelas serão mostradas. Alguns procedimentos, relacionados à criação de componentes para o painel de controle não serão detalhados, e sim referenciados, pois já foram apresentados na Seção 4.3.2. A única diferença para as janelas apresentadas naquela seção é a eliminação da opção *Send to VRML*, obtida com a integração das interfaces.

A primeira janela do ambiente, mostrada na Figura 4.25, solicita a informação de uma URL correspondente a um arquivo VRML ao qual animação será incorporada. Somente a partir desta informação o ambiente começa o processo de criação da

animação interativa. A URL do arquivo contendo a esfera, apresentado na Seção 4.1, deve ser informada.

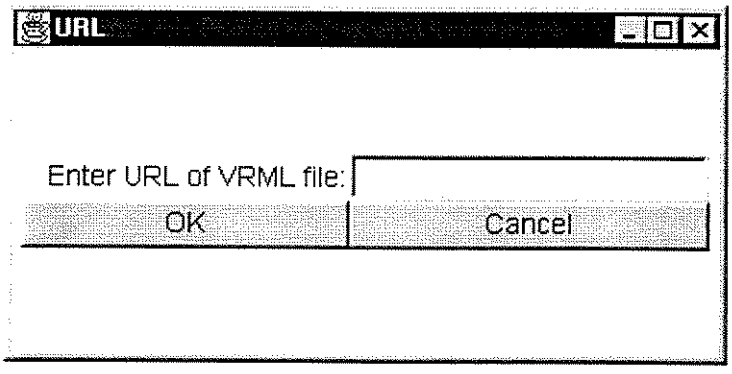

Figura 4.25: Janela para informação de uma URL

O ambiente identifica os objetos da cena e exibe-os em uma lista para seleção, juntamente com os movimentos disponíveis, na janela da Figura 4.26. No caso do exemplo, a cena contém somente a esfera. O animador deve selecionar o objeto para animar e o tipo de movimento que desejar aplicar. Os movimentos da biblioteca são disponibilizados junto com a opção para a definição de um novo. Para definir o movimento retilíneo uniforme a opção *Define movement* deve ser selecionada.

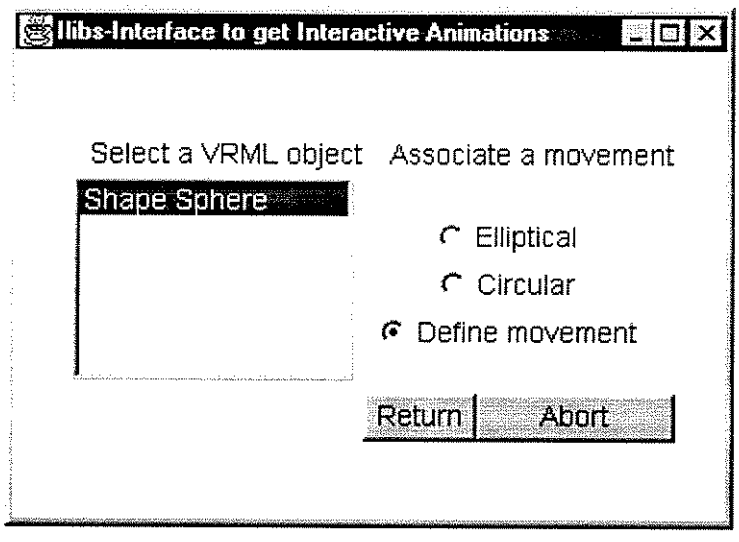

Figura 4.26: Seleção de objeto VRML e tipo de movimento

Quando o animador seleciona a opção para criar novo movimento, informações para a geração de um painel de controle e *script* começam a ser requisitadas. O nome dos arquivos que serão gerados e as opções para edição de um outro painel de controle ou *script* aparecem na janela da Figura 4.27. O ambiente acrescenta prefixos no nome

informado; desta forma, os arquivos para a animação da esfera terão os seguintes nomes: *IUmovement*, para o painel de controle e *Smovement*, para o *script*.

A opção para edição de animações, já criadas através do ambiente, é disponibilizada de duas formas: através da reutilização do painel de controle e do *script* ou apenas do *script*. Se essas opções forem selecionadas na janela da Figura 4.27, uma outra é exibida para a seleção de quais arquivos de dados devem ser carregados pelo ambiente para que a animação seja construída a partir deles. Para o exemplo, é considerada a primeira criação de animação com o movimento retilíneo, por isso essas opções para reutilização não foram selecionadas.

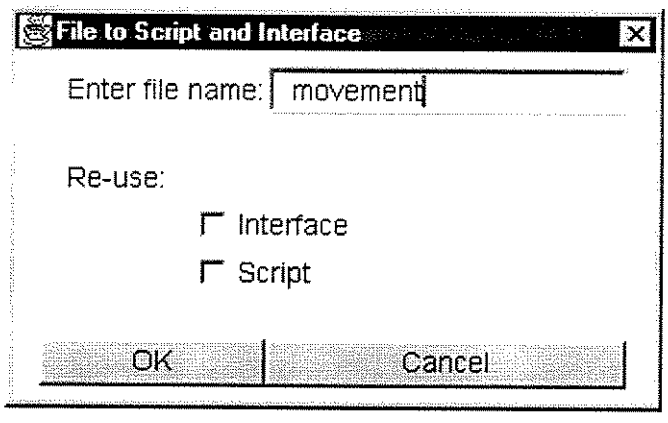

Figura 4.27: Informação do nome do painel de controle e *script* e opção de reutilização

Depois do nome dos arquivos a serem gerados, seus componentes devem ser definidos, através das opções oferecidas pela janela da Figura 4.28. Nesta janela, um subtítulo mostra o objeto VRML sendo animado e o botão para gerar os arquivos aparece desabilitado até que todas as equações para o movimento sejam definidas. O animador pode começar inserindo componentes para o painel de controle ou equações para o movimento. Nenhum dos componentes definidos para o painel de controle através desta janela tem relação com o *script*; os que correspondem a parâmetros da animação são definidos junto com a variável associada.

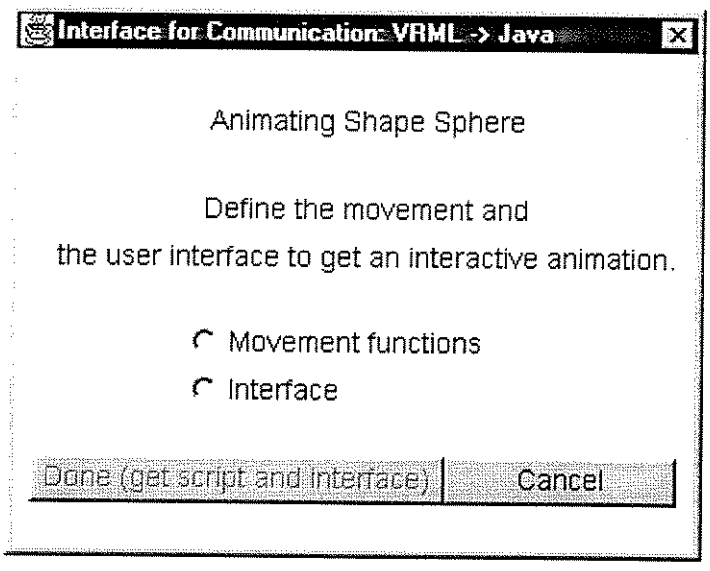

Figura 4.28: Definindo componentes para a animação

A opção *Interface* pode ser selecionada para a inclusão de um título para o painel de controle. Quando esta opção é selecionada e já existem alguns componentes gráficos definidos, o ambiente exibe-os em uma lista, com a possibilidade de inclusão, alteração e remoção. Para inclusão, a mesma janela da Figura 4.18 é exibida e o modo para incluir o título é o mesmo apresentado na Seção 4.3.2.

A opção *Movement functions* deve ser selecionada para a informação das equações do movimento retilíneo uniforme. O ambiente solicita automaticamente a informação das equações para os eixos x, y e z. As outras definições necessárias são solicitadas na ordem em que as variáveis vão sendo informadas. O ambiente possui dois tipos de janelas utilizadas para definir equações, constantes e variáveis: um para os eixos e outro para as variáveis informadas nas equações dos eixos. Essas janelas podem ser observadas a seguir.

Para a equação referente a posição no eixo x a janela da Figura 4.29 é exibida, onde é informada a equação $x_0 + v.t$. As definições para as variáveis x_0 e v serão requeridas a seguir.

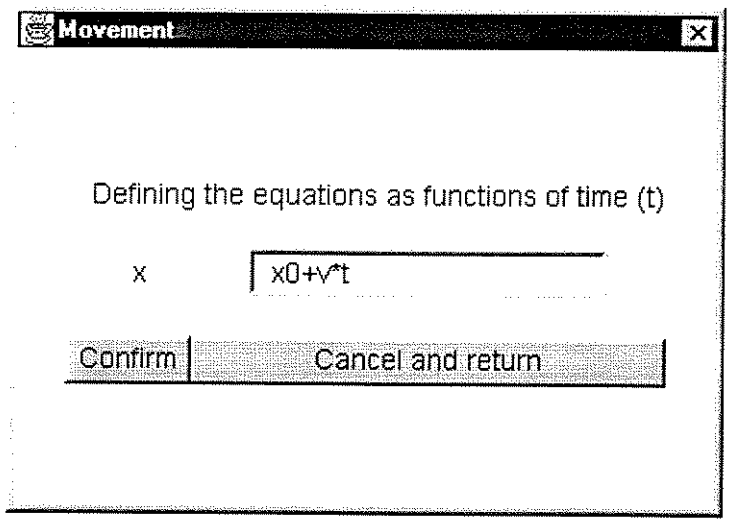

Figura 4.29: Definindo equação para o eixo x

A velocidade `v` é definida na janela da Figura 4.30, com tipo `int` e valor controlado via painel de controle. Quando a opção *Value controled in the user interface* é selecionada, uma outra janela aparece com as opções dos componentes gráficos, para o painel de controle, que podem ser associados a parâmetros da animação. A janela é mostrada na Figura 4.31, onde pode ser observado que as opções são um subconjunto das apresentadas na Figura 4.18. A velocidade é representada por uma barra de rolagem no painel de controle, cuja criação, semelhante à apresentada na Seção 4.3.2, é obtida através da janela da Figura 4.32.

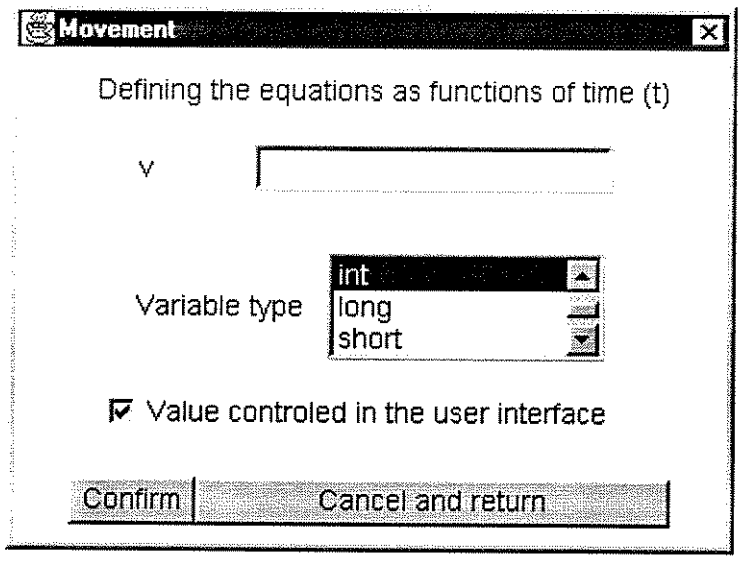

Figura 4.30: Definindo a velocidade

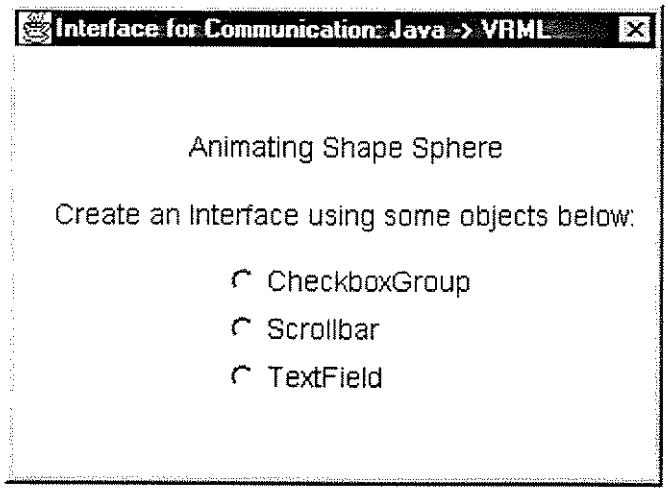

Figura 4.31: Componentes para o painel de controle associados a parâmetros da animação

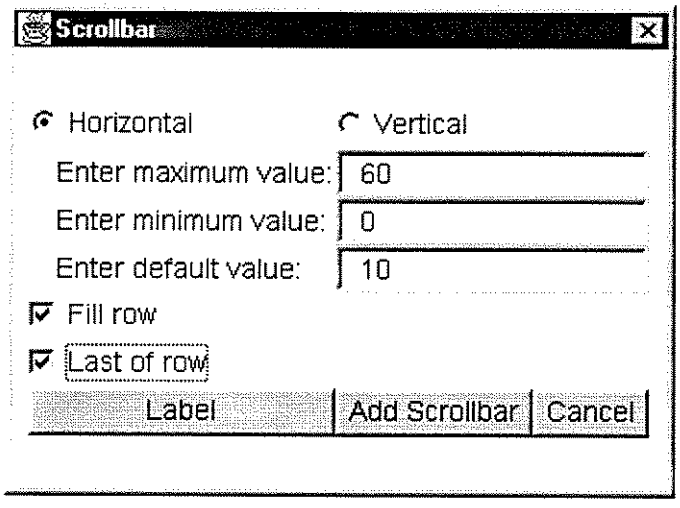

Figura 4.32: Definindo a velocidade no painel de controle

Definida a velocidade, o ambiente exibe uma janela para a definição da posição inicial no eixo x (x_0), do mesmo tipo que para a velocidade (Figura 4.30). A posição inicial também é definida com o tipo `int` e valor controlado via painel de controle, representado por um campo de texto. A opção *TextField* deve ser selecionada na janela da Figura 4.31 e a definição do campo de texto é a mesma apresentada na Seção 4.3.2.

A próxima definição é para o eixo y e, da mesma forma que para x, uma janela do mesmo tipo da Figura 4.29 é exibida, e a equação `y0 + v.t` é informada. A variável `v` já foi definida, portanto, a próxima definição é referente a y_0, que também é do tipo `int` e tem seu valor controlado via painel de controle. O componente gráfico associado com a variável y_0 é uma barra de rolagem, cujo intervalo de valores também é usado para a definição de z_0. Para isso, quando a janela da Figura 4.31 for exibida para a

definição de y_0, a opção *CheckboxGroup* deve ser selecionada. Na janela da Figura 4.33, que é exibida, rótulos referentes às duas variáveis devem ser inseridos e o botão *Associate Scrollbar* deve ser pressionado para a inclusão da barra de rolagem, cujo procedimento é feito através da janela da Figura 4.34, como apresentado na Seção 4.3.2. Desta forma, são inseridos componentes gráficos referentes a dois parâmetros de animação (duas variáveis), onde um é associado com y_0 e o outro é mantido para ser associado à próxima variável definida com valor controlado via painel de controle.

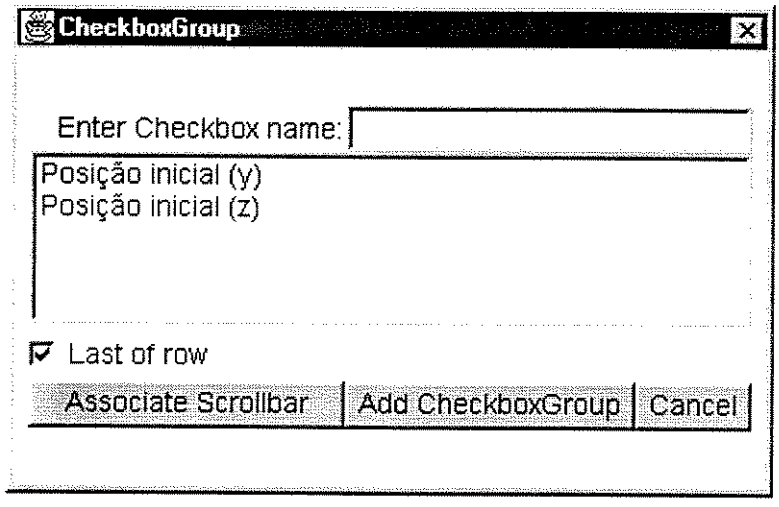

Figura 4.33: Definindo y_0 e z_0 no painel de controle

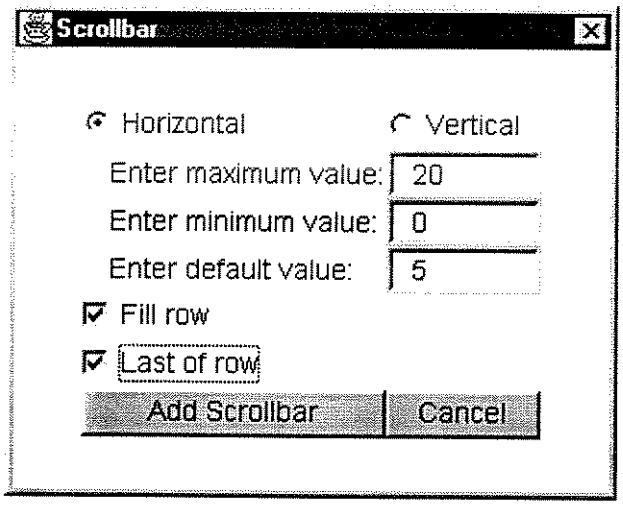

Figura 4.34: Definindo barra de rolagem para y_0 e z_0

A definição para o eixo z é requisitada a seguir, através da janela da Figura 4.35, do mesmo tipo que para x e y. A equação a ser informada consiste na variável z_0, que é definida com o tipo `int` e valor controlado via painel de controle, em uma janela do tipo da Figura 4.30. Neste caso, quando a opção *Value controled in the user interface* é

selecionada, a janela da Figura 4.31 não é exibida, pois já existe um componente gráfico definido para ser associado a uma variável, o que estava "sobrando" na definição de y_0.

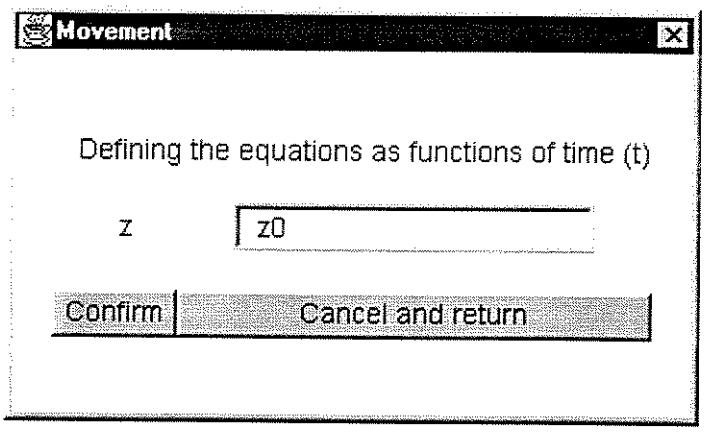

Figura 4.35: Definindo equação para o eixo z

Após todas as variáveis serem definidas, o ambiente exibe as equações em uma lista para que o animador possa conferir e, se necessário, tem a possibilidade de edição. A janela com esta lista é mostrada na Figura 4.36. As variáveis intermediárias, ou seja, as que são definidas nas equações dos eixos, não podem ser editadas diretamente. Para edição, o animador deve selecionar uma das equações dos eixos e confirmar as equações já informadas, até que a janela com o valor de uma variável ou equação que o animador deseja modificar seja exibida.

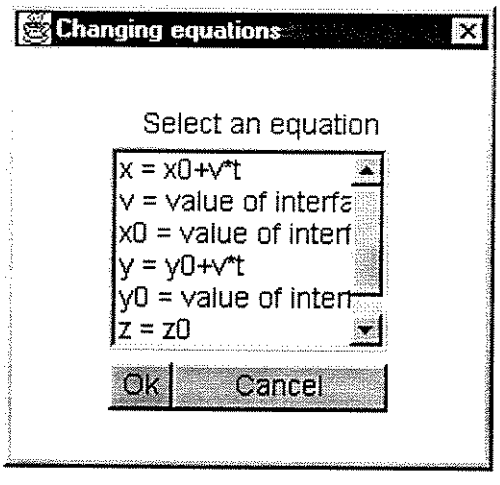

Figura 4.36: Lista de equações informadas

Com todas as variáveis definidas o ambiente volta a exibir a Figura 4.28 com o botão *Done(get script and interface)* habilitado, o qual o animador deve pressionar para obter a animação interativa. O *script* VRML e o painel de controle gerados são praticamente os mesmos já exibidos nas Figuras 4.13 e 4.24, respectivamente. A única

diferença está na disposição dos componentes gráficos do painel de controle, que corresponde a ordem em que as variáveis aparecem nas equações (x_0, v, y_0 e z_0). Além de gerar o *script* e painel de controle, o ambiente modifica o arquivo VRML, incluindo nodos para compor a animação criada. O arquivo VRML da Seção 4.1 modificado é mostrado a seguir:

```
#VRML V2.0 utf8
DEF obj0 Transform {
 children Shape {
 geometry Sphere {}
 appearance Appearance {
       material Material {
             diffuseColor 1 0 0
       }
 }
}
}
DEF XTIMER0 TimeSensor {
 cycleInterval 5
 loop TRUE
}
DEF SCRIPT0 Script {
 url "C:\java\vrml\classes\Smovement.class"
 field SFFloat currentTimer 0.0
 eventIn SFFloat timerIn
 eventOut SFVec3f position
}
ROUTE XTIMER0.fraction_changed TO SCRIPT0.timerIn
ROUTE SCRIPT0.position TO obj0.set_translation
```

4.5. Expansão da biblioteca de movimentos interativos

Esta seção descreve como um movimento pode ser criado e incluído na biblioteca de movimentos interativos. Movimentos consagrados na literatura e/ou que podem ser utilizados em diversas aplicações são indicados para compor a biblioteca de movimentos do ambiente para geração de animações interativas.

O animador pode construir uma animação interativa automaticamente, se optar pela utilização de um dos movimentos pertencentes à biblioteca. Neste caso, basta a informação do nome dos arquivos a serem gerados.

A biblioteca de movimentos interativos, desenvolvida para este ambiente, é composta especialmente por classes que calculam posições em função do tempo e classes que definem painéis de controle e *scripts* automaticamente.

A geração de movimentos para compor a biblioteca pode ser auxiliada pelas interfaces IULib e ScrLib, de modo que um movimento gerado com estas interfaces

pode ser validado antes da decisão de integrá-lo a biblioteca. Uma vez validado, se for interesse de animadores, classes devem ser definidas para que o movimento passe a fazer parte da biblioteca. Isto significa que será disponibilizada, aos animadores, a geração automática do painel de controle e *script* validados.

Para que um movimento interativo seja incluído na biblioteca especificada por este ambiente, três classes devem ser providas para:

- efetuar o cálculo do movimento em função do tempo;
- gerar automaticamente o painel de controle;
- gerar automaticamente o *script* VRML utilizando-se da primeira.

As duas últimas classes podem se beneficiar de métodos utilizados pelas interfaces IULib e ScrLib, pois têm a mesma função: gerar o painel de controle e o *script*, respectivamente; só que neste caso eles são pré-definidos, os componentes gráficos já foram escolhidos e as equações definidas no processo de validação. Para isso, o usuário responsável pela construção da biblioteca deve ter conhecimento dos métodos oferecidos por essas interfaces, além da programação em Java.

O usuário deve se basear no *script* gerado pela ScrLib para definir a classe que calcula o movimento e identificar as modificações necessárias para incluí-la no *script*. As equações devem ser passadas para a classe do movimento e o *script*, no lugar, deve referenciar seus métodos e atributos. Esta é a única diferença entre um *script* gerado pela ScrLib e outro gerado através da biblioteca.

Este esquema, que define classes para o cálculo do movimento separadas do *script*, foi adotado para que estes movimentos, pertencentes à biblioteca, possam ser utilizados em diversas aplicações, inclusive as que fogem ao escopo deste ambiente para criação de animações interativas.

Existem casos em que a validação do *script* através da ScrLib não é possível de forma direta, sem a intervenção do usuário no código gerado. São os casos onde a maneira de implementação do movimento é diferente da esperada pela ScrLib (comentado na Seção 4.2). Nestas situações, o usuário, criador do movimento, deve interferir no código gerado pela ScrLib, inserindo implementação adicional, ou, se preferir incluir o movimento direto na biblioteca, deve criar a classe para o movimento e a classe que gera o *script* utilizando-a, como em situações normais.

4.6. Considerações finais

Neste capítulo, a implementação do ambiente para a criação de animações interativas foi descrita, seguindo a metodologia de desenvolvimento apresentada na Seção 3.6. Um exemplo foi utilizado, ao longo deste capítulo, para facilitar a exposição das interfaces gráficas e do modo de operação do ambiente.

Inicialmente, as aplicações para a criação de animações interativas IULib e ScrLib desenvolvidas de forma independente, foram apresentadas. Através delas, já é possível que o animador crie uma animação interativa. Essas aplicações foram integradas e funcionalidades foram introduzidas a fim de tornar a interação com o animador mais fácil e objetiva. Entre as funcionalidades introduzidas, discutidas na Seção 4.4, estão: a eliminação de redundância, a especialização da ScrLib, a reutilização de movimentos, a consistência de equações e a modificação do arquivo VRML.

Através dos dois mecanismos de reutilização de movimentos, biblioteca e armazenamento em arquivos, é possível criar uma nova animação de forma direta, sem a informação de equações ou componentes gráficos. A diferença é que, através do mecanismo de arquivos, uma animação pode ser editada, ou seja, uma equação pode ser modificada ou um componente gráfico pode ser trocado. Desta forma, uma nova animação pode ser gerada e validada com maior facilidade e rapidez.

Este capítulo mostrou como e porque inserir um movimento na biblioteca. No Capítulo 5, movimentos inseridos na biblioteca deste ambiente serão descritos. O Capítulo 5 vai explorar as funcionalidades do ambiente, onde serão mostrados exemplos para a criação de animações interativas, com a utilização da biblioteca de movimentos interativos.

Capítulo 5.

Utilização da biblioteca de movimentos interativos

Neste capítulo a utilização da biblioteca de movimentos interativos disponibilizada no ambiente para criação de animações interativas será ilustrada. Todos os tipos de movimentos inseridos na biblioteca são constituídos por um *script* VRML e um painel de controle. O *script* define um sistema de equações, expressas como funções do tempo, responsáveis pelo cálculo do movimento. A cena VRML que será associada a este *script* é responsável por atualizar o valor do tempo, que é fornecido como evento ao *script*. O painel de controle define parâmetros para animação, correspondentes a valores de variáveis das equações do *script*. Através de um mecanismo de comunicação o *script* obtém estes valores. Para cada conjunto de parâmetros obtidos, uma nova posição é calculada pelo sistema de equações. Esta posição é enviada como evento ao objeto VRML selecionado para ser animado.

Para a aplicação dos movimentos da biblioteca a cenas VRML uma interação simples com o animador é necessária. A interação consiste na seleção de um dos movimentos e de um dos objetos contidos na cena VRML e na informação do nome com que os arquivos correspondentes ao *script* e painel de controle serão gerados. A geração dos arquivos é feita automaticamente pelo ambiente.

Na biblioteca do ambiente para geração de animações interativas dois tipos de movimentos foram definidos: cinemático e dinâmico. Estes dois tipos de movimento, consagrados na literatura, podem ser utilizados em diversas aplicações e, portanto, foram considerados para compor o conteúdo inicial da biblioteca de movimentos do ambiente. Ambos estão disponíveis para trajetória elíptica e circular. Neste capítulo os movimentos cinemático e dinâmico serão descritos para trajetória elíptica. Os movimentos para trajetória circular são similares, pois são definidos como especializações da trajetória elíptica.

Em seguida, os movimentos serão apresentados para o âmbito de uma aplicação, onde o animador seria um professor que busca apoio para seu curso de física. Os usuários finais seriam os alunos que estão aprendendo cinemática e dinâmica.

Inicialmente será apresentado o cenário em que as animações, geradas com movimentos da biblioteca, serão utilizadas. Na seqüência, os movimentos criados para integrar a biblioteca e envolvidos no cenário serão detalhados; as equações utilizadas no *script* e o painel de controle serão mostrados. Finalmente, para cada um dos tipos de movimento será descrito como obter a animação utilizando o ambiente.

5.1. Descrição do cenário

O cenário considerado como aplicação para a utilização de movimentos da biblioteca consiste em material de apoio a um curso de física, onde o professor ensina tipos de movimento. O professor disponibiliza aos alunos animações que utilizam os movimentos ensinados. Através dessas animações os alunos podem controlar os parâmetros, que correspondem às variáveis das equações do movimento, onde é possível aprender de forma mais intuitiva o significado de cada uma delas. O aluno pode modificar interativamente o valor das variáveis e observar na animação o efeito provocado.

Neste cenário o papel do professor, além de ensinar na sala de aula, é criar as animações e disponibilizá-las aos alunos. O professor não precisa ter grande conhecimento de VRML para obter uma cena inicial a qual a animação será aplicada. Ele pode gerar uma cena simples, que por exemplo contenha somente a definição de uma esfera, como mostrado na Seção 4.1. E ainda, se preferir, pode obtê-la através de outros meios, como por exemplo a Internet. Neste capítulo será considerado que o professor criará as animações para a seguinte cena estática:

```
#VRML V2.0 utf8
DEF small Shape {
    geometry Sphere { radius 2 }
    appearance Appearance {
          material Material {
                diffuseColor 1 0 0
          }
    }
}

Transform {
    translation -6 2 0
    children [
```

```
                DEF big Shape {
                    geometry Sphere { radius 3 }
                    appearance Appearance {
                        material Material {
                            diffuseColor 0 0 1
                        }
                    }
                }
            ]
        }
```

Esta cena define duas esferas: uma menor e outra maior. O movimento será atribuído à esfera menor.

O professor pode criar facilmente diversos tipos de movimento através do ambiente, pois as equações do movimento, que são as principais informações solicitadas, fazem parte de seus conhecimentos. Um exemplo de animação que o professor poderia criar e disponibilizar a seus alunos foi mostrado no capítulo anterior, que se referia ao movimento retilíneo uniforme. Além deste movimento já exemplificado, o professor pode utilizar os movimentos cinemático e dinâmico definidos na biblioteca do ambiente. Neste cenário é considerado que o professor utilizará os movimentos para trajetória elíptica a fim de exemplificar seu curso. Neste capítulo será mostrado como o professor deve proceder para criar animações utilizando os movimentos da biblioteca para trajetória elíptica.

Considerando que o professor tenha ensinado em sala de aula os movimentos elíptico cinemático e dinâmico, o material de apoio criado, referente às animações exemplificando estes movimentos, já pode ser disponibilizado aos alunos. Os alunos, que têm o papel de usuário final das animações criadas através do ambiente, podem utilizar este material de apoio para complementar o aprendizado dos movimentos. Através dessas animações eles têm a oportunidade de manipular, via painel de controle, as variáveis que fazem parte das equações do movimento aprendido em sala de aula. O efeito de cada variável pode ser visualizado diretamente na animação, o que complementa o aprendizado do aluno.

5.2. Movimentos da biblioteca para trajetória elíptica

Nesta seção os movimentos da biblioteca para trajetória elíptica serão descritos. As equações dos movimentos cinemático e dinâmico utilizadas no *script* serão apresentadas, bem como os painéis de controle definidos para o usuário final informar os parâmetros da animação.

Quanto à criação de painéis de controle para os movimentos, apresentados a seguir, é possível verificar a utilização de diversos componentes gráficos e algumas combinações entre eles, feita através dos botões nas respectivas janelas (como mostrado na Seção 4.3.2).

As interfaces IULib e ScrLib, apresentadas no Capítulo 4, foram utilizadas para a validação dos movimentos e da interação associada antes de comporem à biblioteca. O movimento dinâmico é um exemplo em que o usuário deve interferir no código gerado pela ScrLib, como discutido na Seção 4.5. Para incluir os movimentos na biblioteca foram criadas, para cada um, três classes: uma para o cálculo do movimento, outra que gera o *script* utilizando-a e outra para a geração do painel de controle. Na primeira as equações são definidas e as duas últimas utilizam métodos disponibilizados pelo ambiente; são os mesmos métodos utilizados na criação de um movimento qualquer definido pelo animador.

Primeiramente, os parâmetros que podem afetar um movimento elíptico serão apresentados. Depois, os movimentos para trajetória elíptica inseridos na biblioteca serão descritos. Será mostrado como esses movimentos são aplicados na cena VRML apresentada na Seção 5.1. A criação dessas animações compõem o cenário definido na seção anterior.

5.2.1. Parâmetros do movimento elíptico

Existem vários parâmetros que podem gerar diferentes movimentos e trajetórias elípticas. Considerando o movimento em duas dimensões, os parâmetros que afetam a trajetória são:

- coeficiente da elipse no eixo x
- coeficiente da elipse no eixo y
- excentricidade

A excentricidade apresenta relação com os dois eixos. Desta forma, para manipular a trajetória são utilizados somente dois (quaisquer) dos parâmetros apresentados acima. Para os movimentos da biblioteca são utilizados os coeficientes para os eixos x e y.

Os parâmetros que podem afetar o movimento elíptico são:

- velocidade: movimento uniforme
- aceleração: movimento uniforme acelerado
- massas m_1 e m_2: massas de dois corpos utilizadas no movimento dinâmico gravitacional. No exemplo correspondem às massas das esferas maior e menor respectivamente.

Esses fatores são relacionados a um tipo específico de movimento; todos eles são disponibilizados na biblioteca.

Para o movimento elíptico são disponibilizados na biblioteca diversos tipos de movimentos que combinam, através de seus painéis de controle associados, os parâmetros apresentados. Os movimentos definidos são:

- cinemático uniforme
  - definido com velocidade fixa, permitindo a manipulação da trajetória;
  - definido com velocidade variável, que é manipulada junto com a trajetória;
- cinemático acelerado, que permite a manipulação da aceleração e da trajetória;
- dinâmico que permite a manipulação das massas e da trajetória.

5.2.2. Movimento cinemático

Na cinemática, a variação de coeficientes da elipse, velocidade e aceleração, geram diferentes trajetórias. Esses são os parâmetros utilizados nas equações do movimento e informados pelo usuário final através do painel de controle.

O *script* utiliza os parâmetros recebidos do painel de controle e os instantes de tempo para gerar como evento a posição. Esta posição é enviada à esfera menor da cena VRML, produzindo animação cinemática elíptica.

A trajetória elíptica é calculada de acordo com as seguintes equações:

$$\begin{aligned} x &= a * \cos\theta \\ y &= b * \text{sen}\theta \\ z &= 0 \end{aligned} \qquad (5.1)$$

onde x, y e z representam as coordenadas de uma posição, considerando o ângulo $\theta = \theta_0 + \gamma * t + \omega * t^2 / 2$, no tempo t, velocidade γ e aceleração $\omega = (\gamma_{i+1} - \gamma_i)/\Delta t$.

Os movimentos disponíveis na biblioteca do ambiente que utilizam variações desta equação serão apresentados a seguir.

5.2.2.1. Movimento uniforme com controle da trajetória

Para manipular a trajetória elíptica, de acordo com a equação 5.1, os parâmetros definidos no painel de controle são: a e b. A velocidade γ é fixa e constante e a aceleração ω é considerada nula, o que determina o movimento uniforme.

O painel de controle, mostrado na Figura 5.1, é composto por duas barras de rolagem, identificadas por rótulos, correspondendo aos coeficientes da elipse nos eixos *x* e *y*, respectivamente. Os valores obtidos das barras de rolagem, que podem ser visualizados ao lado de cada uma, são fornecidos ao cálculo do movimento, a cada interação. Assim, a trajetória da esfera menor é modificada.

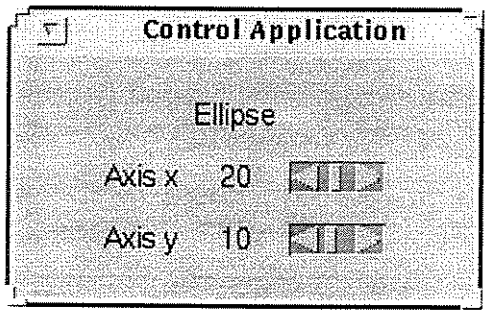

Figura 5.1: Painel de controle para determinar a trajetória elíptica

5.2.2.2. Movimento uniforme com controle da velocidade

Para gerar um movimento uniforme a partir da equação 5.1, a velocidade γ deve ser constante e a aceleração ω nula. Neste caso, além da trajetória, o movimento também pode ser alterado através da velocidade. Os parâmetros definidos no painel de controle são: a, b e γ.

Como visto na Figura 5.2, o painel de controle é composto por rótulos e barras de rolagem. Foram definidos dois títulos que descrevem o tipo de movimento e fazem uma certa classificação dos parâmetros: geométricos e cinemáticos. As barras de rolagem são associadas aos coeficientes da elipse e à velocidade do movimento. Os valores obtidos das barras de rolagem, que podem ser visualizados ao lado de cada

opção, são fornecidos ao cálculo do movimento, que é modificado quando alguma alteração é efetuada.

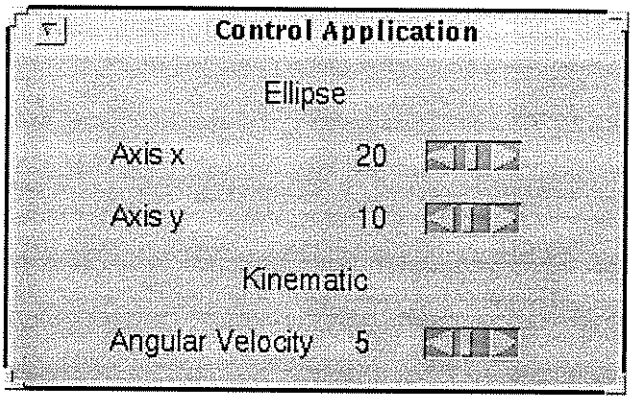

Figura 5.2: Painel de controle para o movimento cinemático uniforme

5.2.2.3. Movimento acelerado

Para gerar o movimento acelerado a partir da equação 5.1, a aceleração ω não deve ser nula e a velocidade γ é obtida da relação que existe entre elas. Neste caso, além da trajetória, o movimento também pode ser alterado através da aceleração. Os parâmetros definidos no painel de controle são: `a`, `b` e ω.

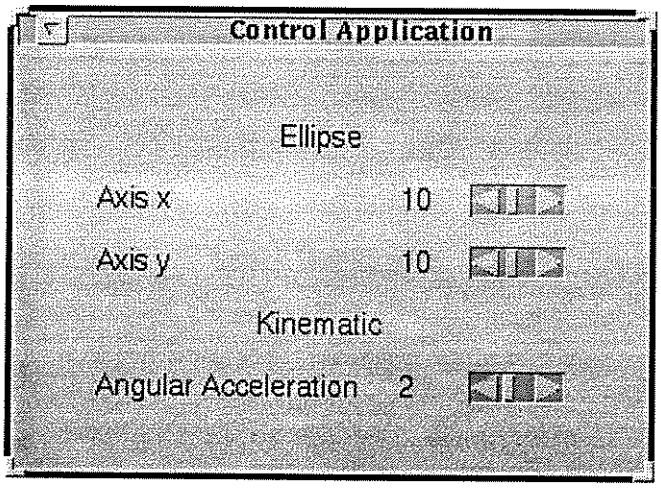

Figura 5.3: Painel de controle para o movimento cinemático acelerado

A Figura 5.3 mostra o painel de controle composto por rótulos e barras de rolagem. Foram definidos dois títulos que descrevem o tipo de movimento e fazem uma certa classificação dos parâmetros: geométricos e cinemáticos. As barras de rolagem são associadas aos coeficientes da elipse e à aceleração do movimento. Os valores obtidos das barras de rolagem, que podem ser visualizados ao lado de cada opção, são

fornecidos ao cálculo do movimento, que é modificado quando alguma alteração é efetuada.

5.2.3. Movimento dinâmico

No movimento dinâmico, aqui exemplificado para forças gravitacionais, os fatores que geram diferentes trajetórias elípticas são: coeficiente da elipse referente ao eixo *x* (a), eixo *y* (b) e massas (m_1 e m_2), que fazem conveniente a definição de expoentes. Para cada conjunto destes valores, obtidos no painel de controle, o *script* determina e armazena o tempo para o ângulo θ variando de 2 em 2 graus, de acordo com Greenwood [GREENWOOD, 1965]:

$$t = \sqrt{a^3 / \mu} * (E - \epsilon * \sin E)$$

$$\epsilon = \sqrt{1 - (b / a)^2}$$

$$E = \arcsin \left((\sqrt{1-\epsilon^2}) * \sin\theta) / (1 + \epsilon * \cos\theta) \right)$$

$$\mu = G * (m_1 + m_2)$$

onde $G = 6,67 \times 10^{-8}$ $cm^3/g\ sec^2$ e ϵ é a excentricidade. Esse procedimento é feito somente quando a esfera se encontra na posição inicial, onde t=0, pois exige muito tempo de processamento quando comparado ao tempo de um ciclo de animação.

O instante de tempo recebido do arquivo VRML é comparado com os valores de tempo armazenados, a fim de obter o ângulo θ correspondente, e assim determinar a posição para a esfera, pois:

$$x = r * \cos\theta + a * \epsilon$$

$$y = r * \sin\theta$$

$$z = 0$$

onde $r = a * (1-\epsilon^2) / (1 + \epsilon * \cos\theta)$.

Este tipo de movimento é definido por um sistema de equações complexo, onde é difícil isolar o valor de θ para obter a posição como função direta do tempo. Deste modo, a maneira de implementação exige um mecanismo adicional para a armazenagem de valores de tempo usados para posterior comparação com o valor recebido. O ambiente está preparado para receber um sistema de equações, através do qual a posição é obtida; no entanto, não oferece nenhum tipo de mecanismo adicional. Portanto, neste tipo de situação, para a criação do movimento através do ambiente, a fim de validá-lo antes de adicioná-lo à biblioteca, o usuário pode usar a ScrLib para gerar o modelo do *script*, mas deve interferir no código gerado, inserindo a implementação adicional.

O painel de controle definido para o movimento dinâmico pode ser visto na Figura 5.4. Os parâmetros são obtidos das barras de rolagem, como para o movimento cinemático. As duas barras de rolagem relativas ao valor da massa e expoente são associadas a um grupo de opções, que determina qual das massas (m_1 ou m_2) está sendo alterada, ou seja, a que está selecionada. Os coeficientes `a` e `b` têm sempre o valor correspondente ao da respectiva barra de rolagem, pois ela é individual. Este valor pode ser visualizado por um rótulo ao lado da barra de rolagem (sua criação é automática). Nestes casos, onde as barras de rolagem são únicas, foram necessárias definições de rótulos adicionais, pois elas precisam informar quais parâmetros estão representando. Rótulos definidos como títulos também foram utilizados.

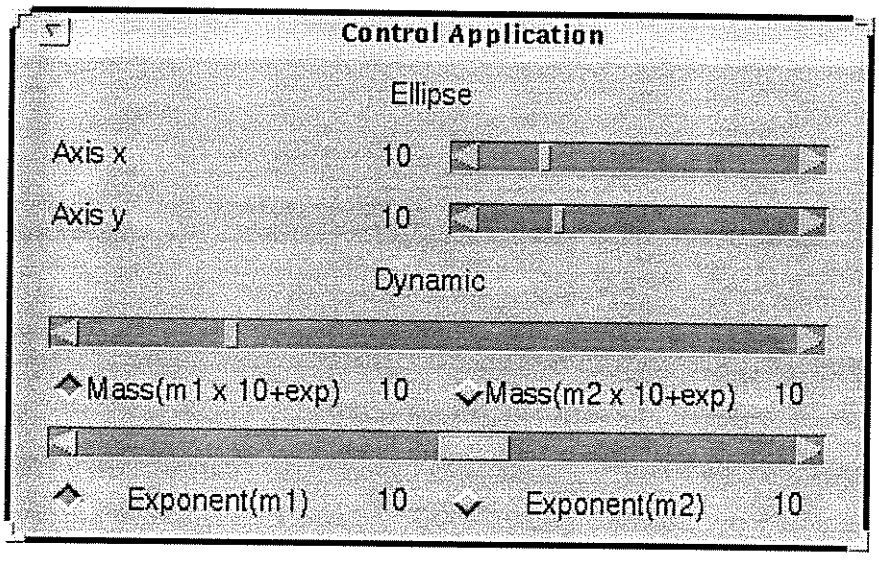

Figura 5.4: Painel de controle para o movimento dinâmico

5.2.4. Criando animações

Para a criação de animações interativas com a utilização de movimentos existentes na biblioteca, uma interação simples com o ambiente é necessária. Considerando o cenário descrito neste capítulo, o professor assume o papel do animador e cria as animações para uma esfera utilizando os movimentos cinemático e dinâmico para trajetória elíptica. As interfaces que o ambiente provê e a atuação do animador para este procedimento serão descritos a seguir.

Na primeira janela do ambiente, mostrada na Figura 5.5, que solicita a informação de uma URL correspondente a um arquivo VRML a ser animado, o arquivo contendo as esferas, apresentado na Seção 5.1, deve ser informado.

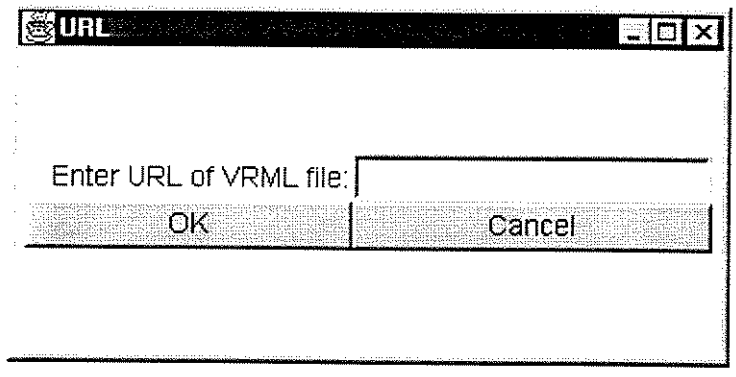

Figura 5.5: Janela para informação de uma URL

O ambiente identifica as esferas contidas na cena e exibe-as para seleção, juntamente com os movimentos disponíveis, na janela da Figura 5.6. O animador deve selecionar a esfera menor para ser animada e o tipo de movimento que deseja aplicar. No caso do exemplo, o movimento para trajetória elíptica deve ser selecionado, através da opção *Elliptical*. Quando esta opção é selecionada, a janela da Figura 5.7 é mostrada para que o animador escolha qual o tipo de movimento para trajetória elíptica deve ser definido. Uma lista oferece as opções existentes na biblioteca relativa aos movimentos cinemático e dinâmico. Para a criação da animação cinemática uniforme com alteração da velocidade, o animador deve selecionar a opção *Kinematic*. Depois que o animador selecionar este tipo de movimento, uma outra janela, mostrada na Figura 5.8, é exibida, solicitando o nome para os arquivos que serão gerados, correspondentes ao *script* VRML e painel de controle. Esta informação finaliza o processo de criação da animação interativa que utiliza um movimento cinemático definido na biblioteca para trajetória elíptica. O processo para criar outros tipos de animações para trajetória elíptica é o mesmo; difere na opção que deve ser selecionada na janela da Figura 5.7. Para a animação cinemática uniforme com controle da trajetória a opção *Trajectory* deve ser selecionada, para a animação cinemática acelerada a opção *Accelerated kinematic* e para a animação dinâmica a opção *Dynamic*.

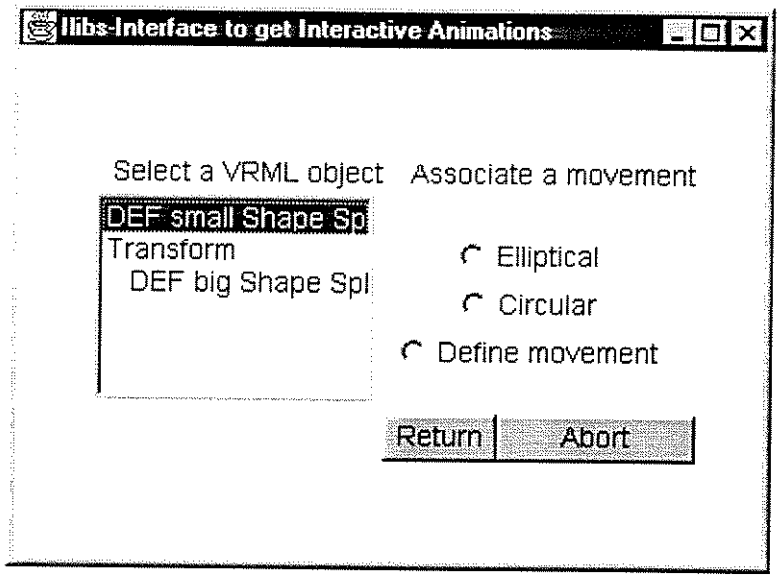

Figura 5.6: Seleção de objeto VRML e tipo de movimento

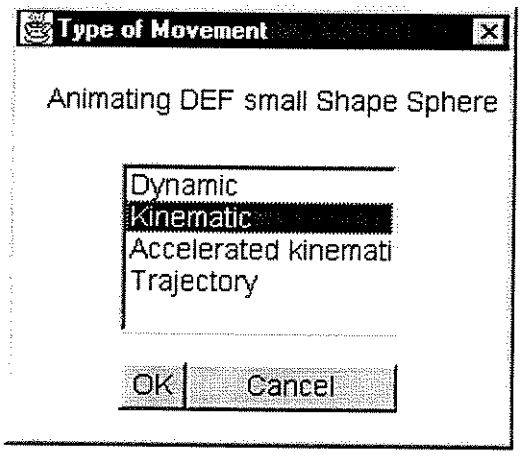

Figura 5.7: Selecionando tipo de movimento cinemático

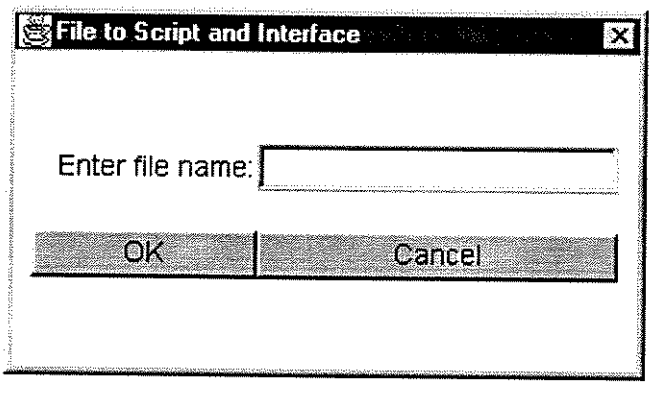

Figura 5.8: Informando o nome do arquivo para o *script* e painel de controle

Com os arquivos gerados o animador pode compilá-los para disponibilizar as animações para o aprendizado dos usuários finais: os alunos. Neste caso, os alunos teriam para a sua aplicação as interfaces definidas nas Seções 5.2.2 e 5.2.3.

5.3. Considerações finais

Este capítulo mostrou a fácil utilização da biblioteca de movimentos do ambiente para criação de animações interativas. Através de informações simples o animador pode obter uma animação interativa.

Os movimentos definidos na biblioteca são consagrados e podem ser utilizados em diversas aplicações como, por exemplo, no apoio a um curso de física, cujo o cenário foi apresentado na Seção 5.1.

Como visto na Seção 5.2.3 existem movimentos complexos que envolvem diferentes técnicas de implementação não oferecidas pelo ambiente. Quando o animador precisa definir animações que utilizam este tipo de movimento, há uma dificuldade na utilização do ambiente, pois o *script* não pode ser gerado apenas com a informação de equações; o animador precisa interferir no código gerado.

Para inserir um movimento na biblioteca o usuário criador do movimento deve definir classes e ter conhecimento de programação Java, especialmente dos métodos definidos pelo ambiente que oferecem suporte à geração de *script* e painel de controle. Esta não é uma tarefa fácil, mas depois que o movimento é integrado na biblioteca, as recompensas são evidentes: sua utilização é ilimitada e a criação de animações é obtida de forma automática; basta ao animador informar o nome dos arquivos que serão gerados.

Capítulo 6.

Conclusões

O objetivo deste trabalho é possibilitar a fácil criação das animações interativas exemplificadas na Seção 2.3.5. Este objetivo foi atingido com o desenvolvimento do ambiente interativo para a geração de animações interativas, apresentado ao longo dos capítulos anteriores. Diversas funcionalidades, apresentadas na Seção 3.4, foram disponibilizadas a fim de oferecer maior versatilidade e mais facilidades na utilização do ambiente. Nos capítulos 4 e 5 pôde ser constatado que através de interação simples com o ambiente, o animador pode obter animações interativas.

Neste capítulo as contribuições do ambiente para criação de animações interativas serão apresentadas. Na primeira seção serão descritas as vantagens na utilização do ambiente oferecidas aos animadores cujo interesse são as animações interativas.

O ambiente desenvolvido é completo para a criação de animações interativas específicas (àquelas exemplificadas na Seção 2.3.5), cujos *scripts* VRML são definidos para o cálculo de movimento em função do tempo. Tentando tornar o ambiente mais versátil e completo, atingindo um maior número de aplicações, trabalhos que podem ser desenvolvidos como aplicações de apoio ao ambiente serão apresentados na segunda seção. A utilização de outras tecnologias de programação que podem beneficiar a implementação do ambiente também serão discutidas.

6.1. Contribuição

O ambiente desenvolvido procura facilitar a construção de animações interativas, cujos pré-requisitos envolvem conhecimentos de programação Java, VRML e tipos de movimento. VRML é a linguagem utilizada para a modelagem da cena e a definição da animação, que inclui a utilização do nodo `Script`, permitindo a associação de um programa Java responsável pelo cálculo do movimento (neste trabalho referenciado

como *script*). No cálculo do movimento equações são informadas envolvendo variáveis cujo controle é interativo com o usuário, através de uma outra aplicação Java, chamada de painel de controle. A linguagem Java é utilizada para a definição do *script* e do painel de controle, o que envolve interface gráfica, mecanismo de comunicação utilizado entre as aplicações e a extensão definida pela especificação VRML que o *script* deve seguir. Conhecimento de linguagens de programação não é comum entre os animadores e, considerando que o mesmo procedimento de codificação é utilizado para diversos tipos de animações, o ambiente torna esta tarefa transparente aos seus usuários. Para obter uma animação VRML interativa, através do ambiente, o animador não precisa ter conhecimento avançado de VRML, como a utilização de nodos interpoladores, sensores ou `Script` que introduz complexidade à cena; basta a modelagem de uma cena estática, o que já é parte do seu escopo de trabalho. A modelagem de movimento também é uma atividade comum no meio dos animadores; portanto, o ambiente permite a definição de movimentos, mas também oferece um conjunto pré-definido, onde a seleção de um movimento isenta o animador desta tarefa. Desta forma os pré-requisitos para se obter uma animação interativa são suportados pelo ambiente, eliminando a responsabilidade do animador ter conhecimentos avançados a sua área de atuação.

O desenvolvimento de um ambiente para construção de animações interativas foi introduzido em uma publicação [TAMIOSSO, 1997a], onde a aplicação para a construção de painéis de controle (IULib) foi apresentada.

O ambiente desenvolvido para a criação de animações interativas apresenta uma interface facilitada com o usuário, possibilitando o rápido aprendizado do modo de operação. A interface gráfica é composta por componentes que são comuns na maioria dos sistemas computacionais, tais como: campos de texto, listas e opções.

Através de uma interface simples, diversas funcionalidades são disponibilizadas para auxiliar na criação de animações interativas. A partir de uma cena VRML estática definida pelo animador, a criação da animação pode seguir três caminhos: a definição de um novo movimento, a reutilização de um movimento definido previamente ou a utilização de um movimento pré-definido. O movimento informado é associado a um dos objetos da cena VRML selecionado pelo animador, concluindo a criação da animação interativa.

Para criar um novo movimento, o animador conta com a verificação das equações para garantir a consistência na definição de variáveis. Interfaces são exibidas solicitando a definição de variáveis à medida que são informadas. Ao final da interação, todas as equações são listadas para que sejam conferidas e talvez modificadas. O ambiente provê um mecanismo para reutilização destes movimentos, onde os dados informados são mantidos em arquivos que podem ser posteriormente interpretados.

A opção de edição de movimentos torna a utilização do ambiente mais versátil, permitindo que movimentos possam ser validados facilmente. A validação pode consistir na investigação de qual o melhor componente gráfico seria associado a um parâmetro da animação ou na variação de alguma variável ou equação do movimento. Com a seleção da opção para reutilizar o movimento, as equações são listadas (Figura 4.36) e, através de mecanismos de edição, a geração de uma nova animação interativa pode ser obtida.

Vários movimentos interativos foram pré-definidos para compor uma biblioteca que é integrada ao ambiente. Foram definidas classes responsáveis unicamente pelo cálculo do movimento que podem ser utilizadas em outras aplicações. Esta biblioteca pode ser estendida por usuários que tenham interesse em disponibilizar outros tipos de movimento. Quando um movimento ainda está em estudo, antes de incluí-lo na biblioteca, o ambiente, apoiado pela funcionalidade de edição, pode ser utilizado para auxiliar na validação. Para incluir um movimento na biblioteca, o usuário deve ter conhecimento de programação Java e da implementação do ambiente, onde os diagramas de classe apresentados no Capítulo 3 e a documentação do código disponibilizada em HTML (gerada com a ferramenta `javadoc`) podem auxiliar. É aconselhável que movimentos de interesse em diversas aplicações passem a integrar a biblioteca devido à maior facilidade para a criação de animações interativas. No ambiente os movimentos da biblioteca são exibidos em uma lista, onde a simples seleção de um deles é necessária para gerar uma animação interativa. Através do ambiente, a utilização da biblioteca é a maneira mais fácil e rápida para se obter uma animação interativa.

Na conclusão do processo para criar animações interativas (seguindo qualquer uma das alternativas), o ambiente altera a cena VRML incluindo os nodos para compor a animação e gera as classes Java referentes ao *script* VRML e ao painel de controle. Comparando com o esforço que um animador dispenderia efetuando todo o

procedimento para construir uma animação interativa, a utilização do ambiente é muito vantajosa, pois o animador é poupado em tempo e conhecimento desnecessário. O tempo que seria utilizado em programação pode ser dedicado, por exemplo, à validação de novos movimentos. Com o ambiente é muito mais fácil a criação de animações interativas: consiste em, no máximo, informar equações do movimento.

6.2. Trabalhos futuros

A seguir serão apresentados trabalhos que podem ser desenvolvidos como aplicações de apoio ao ambiente com o objetivo de torná-lo mais versátil, incentivando sua utilização.

O ambiente foi desenvolvido para gerar animações interativas cujo movimento é calculado com funções diretas do tempo, sem suporte a mecanismos adicionais de implementação, como o exigido pelo movimento dinâmico relatado na Seção 5.2.3. Interfaces de apoio poderiam ser desenvolvidas para possibilitar a entrada e utilização de recursos como, por exemplo, classes `Hashtable`, `Vector` ou definidas pelo usuário, que seriam combinadas com as equações do movimento na implementação do *script* VRML. Um trabalho nesta direção deveria partir de uma análise de possíveis implementações de movimentos, tentando buscar a maneira mais genérica para a entrada dos diferentes mecanismos levantados. Além disso, uma adaptação no ambiente deve ser feita, no que diz respeito ao módulo de geração do *script* VRML.

Uma aplicação de apoio ao ambiente poderia ser desenvolvida para facilitar a expansão da biblioteca de movimentos. Esta aplicação deve ser dedicada à criação e integração de um novo movimento à biblioteca do ambiente. No ambiente desenvolvido, o processo para criar um movimento para a biblioteca, descrito na Seção 4.5, não é simples e envolve conhecimentos de programação para a definição de três classes Java. Como a programação em Java não é comum no meio dos animadores, a expansão da biblioteca é uma atividade restrita a alguns usuários. A aplicação de apoio deve tornar as questões de programação transparentes, provendo um mecanismo automatizado que permita a fácil criação de movimentos para compor a biblioteca do ambiente.

Na interface gráfica desenvolvida para o ambiente, nenhuma sofisticação como a utilização de ícones e componentes mais elaborados como tabelas foram utilizados,

devido à disponibilidade das ferramentas na fase de implementação. O JDK 1.0.2 foi utilizado, onde somente componentes da AWT são disponíveis para construir interfaces com o usuário. Versões mais recentes do JDK já incluem o Swing, uma biblioteca de componentes gráficos, através da qual é possível a construção de interfaces mais sofisticadas. Alguns exemplos de componentes disponibilizados pelo Swing são: tabelas, pastas e a combinação de ícones com vários outros componentes, tais como: menus, rótulos e botões.

Este trabalho poderia ser estendido com uma tradução do código desenvolvido com JDK 1.0.2 para a versão 1.2 que já inclui o Swing. Esta tradução não é direta, pois, devido ao novo modelo de eventos introduzido na versão 1.1, muitas modificações relacionadas à interface gráfica são necessárias. Como a idéia é proporcionar ao usuário uma interface mais sofisticada e atrativa, os componentes da AWT devem ser substituídos pelos do Swing e, assim, a interface gráfica do ambiente é praticamente refeita.

Além da vantagem de melhor interface, com a migração para a versão 1.2 do JDK, outros mecanismos de comunicação podem ser utilizados para estabelecer a conexão entre um *script* VRML e um painel de controle. RMI, introduzido na versão 1.1 do JDK, poderia ser utilizado para posterior análise de vantagens e desvantagens sobre *sockets*. CORBA, incorporado à versão 1.2 do JDK, é outra tecnologia que poderia ser envolvida na implementação. Desta forma, os mecanismos de comunicação relacionados à Java seriam comparados com o enfoque para as animações interativas.

A principal preocupação no desenvolvimento do ambiente era em disponibilizar as funcionalidades para prover a fácil criação de animações interativas. No entanto, nenhuma atenção especial foi dedicada aos caminhos que podem levar a casos de erro. Essas situações podem ser alcançadas devido ao ineficiente tratamento de erros na entrada de dados. A validação na entrada de dados é um trabalho rápido que pode render bons resultados: a eliminação das situações de erro.

Outra aplicação que poderia ser considerada como extensão do ambiente é a implementação de auxílio ao usuário na interação com o ambiente. Essa implementação poderia consistir na criação de um conjunto de arquivos de ajuda que seriam associados às diversas interfaces gráficas do ambiente. Um botão de ajuda poderia ser incluído em cada interface e através dele o usuário poderia obter uma descrição dos dados que deve

informar e do procedimento a ser efetuado. Os arquivos da ajuda podem ser escritos em HTML, pois existem classes Java, dedicadas a construção de ajudas para aplicativos, que definem *browsers* HTML e possuem suporte para navegação.

Apêndices

Apêndice I.

Características da linguagem Java relacionadas ao ambiente

1. Introdução

Java é uma linguagem orientada a objetos desenvolvida para ser simples e com portabilidade para diversas plataformas e sistemas operacionais.

O ambiente para criação de animações interativas foi implementado com a versão 1.0 de JDK (*Java Developer's Kit*) [LEMAY, 1996], embora que, na data em que foi concluído, outras versões já tivessem sido liberadas.

Neste apêndice as principais características de Java serão abordadas, considerando sua utilização neste trabalho. O ambiente desenvolvido é uma aplicação multi-plataforma, onde as interfaces gráficas foram implementadas com a utilização do AWT (*Abstract Windowing Toolkit*). Para aplicações distribuídas, Java oferece mecanismos de comunicação; na implementação das animações interativas *sockets* foram utilizados. Embora RMI seja disponível somente com JDK 1.1 e CORBA seja incorporado em JDK 1.2, uma discussão será apresentada para possível e posterior utilização. As seções a seguir apresentarão as diferenças entre *applets* e aplicações, o ambiente multi-plataforma, o AWT e os mecanismos de comunicação.

2. Programas Java: applets e aplicações

Um *applet* é um programa dinâmico e interativo que pode ser executado dentro de uma página *web*, mostrada por um *browser*, como Netscape 2.0 ou versão posterior, que forneça suporte à Java. [LEMAY, 1996]

Para criar um *applet*, basta escrevê-lo na linguagem Java, compilá-lo usando um compilador Java e referenciá-lo em uma página HTML. Então, os arquivos Java e HTML devem ser colocados em um *site* da *web* para torná-los acessíveis. Assim,

quando alguém usa um *browser*, que suporta Java, para visualizar a página com o *applet*, o *browser* carrega esse *applet* para o sistema local e executá-o, promovendo a interação com o usuário.

Com Java, além de criar *applets*, pode-se alcançar objetivos e resolver problemas como seria feito com a utilização de outras linguagens de programação, como C ou C++.

Aplicações são programas mais gerais, que são executados apenas com o interpretador Java, por exemplo, através de uma linha de comando. Diferente de *applets*, não precisam de um *browser*. Java pode ser usada para criar todo tipo de aplicação, incluindo, por exemplo, gerenciamento de rede e elementos de interface com o usuário.

Como *applets* executam dentro de um *browser*, eles têm a vantagem de sua estrutura: janela existente, tratamento de eventos, contexto gráfico e suporte à interface com usuário. Aplicações podem criar toda esta estrutura, mas não a requerem.

Embora *applets* tenham uma certa conveniência sobre aplicações, existem algumas restrições relacionadas ao que eles podem fazer. Essas restrições previnem que *applets* causem danos ou comprometam a segurança do sistema, já que eles podem ser carregados de qualquer lugar e executados sobre o sistema local. *Applets*:

- não podem ler ou escrever no sistema de arquivos local, exceto em alguns diretórios pré-definidos;
- geralmente, não podem se comunicar com outro servidor senão aquele onde está armazenado;
- não podem executar programas do sistema local;
- não podem carregar programas nativos à plataforma local, incluindo bibliotecas compartilhadas, como DLL.

3. Ambiente de desenvolvimento: multi-plataforma

Independência de plataforma é uma das mais importantes vantagens que Java tem sobre outras linguagens de programação, principalmente quando são considerados sistemas que precisam trabalhar em diversas plataformas.

Java é independente de plataforma no nível binário e de fonte. No nível de fonte, tipos de dados primitivos têm tamanhos consistentes ao longo de todas as plataformas

de desenvolvimento. A construção da biblioteca de classes facilita a escrita de código, que pode ser movido de uma plataforma a outra sem precisar reescrevê-lo para funionar.

Arquivos compilados, por sua vez, podem ser executados em múltiplas plataformas sem precisar recompilar o fonte. Esses arquivos são formados por *bytecodes*, que são um conjunto de instruções que se assemelham muito com código de máquina, mas não são específicos a um processador.

O ambiente de desenvolvimento é constituído por um compilador e um interpretador Java. O compilador, a partir de um programa Java, gera *bytecodes*, ao invés de código de máquina, gerado por outras linguagens convencionais. Para executar um programa Java, um interpretador de *bytecodes* deve ser usado. Para *applets*, há um interpretador construído dentro do próprio *browser* que o executa.

Tendo o programa Java em *bytecodes* basta ter o interpretador para executá-lo em diversas plataformas.

A desvantagem de se usar *bytecodes* está na velocidade da execução. Programas específicos a um sistema, que executam diretamente sobre o *hardware* para o qual foram compilados, executam mais rápido que *bytecodes* Java, que devem ser processados pelo interpretador. Se o desempenho for um ponto crítico no sistema, existem algumas soluções, como, ligar código nativo ao programa ou usar ferramentas para converter *bytecodes* em código nativo. Mas é importante notar que, com essas soluções, a portabilidade dos *bytecodes* é perdida.

4. AWT – *Abstract Windowing Toolkit*

AWT foi projetado pela necessidade de se criar aplicações de interface gráfica mais complexa, comparada ao uso de texto ou interações básicas, que por exemplo, envolvem a posição do *mouse*. AWT pode ser usado para trabalhar tanto com *applets* como aplicações.

A idéia de AWT é que uma janela Java é formada por um conjunto de componentes, que incluem, dentre vários, janelas, barras de menu, botões, campos de texto e *containers*, que por sua vez, podem conter outros componentes, e assim formar uma hierarquia. A hierarquia define a ordem que os componentes são postos na tela e afeta o tratamento de eventos. [LEMAY, 1996]

Os eventos são recebidos pelos componentes mais internos da hierarquia. Se o evento não for de interesse, ou seja, não pode ser tratado pelo componente, ele é passado a um nível superior.

Na maioria dos sistemas de janelas, os componentes de interface gráfica são dispostos na tela através de posições de *pixels*. No AWT, uma janela deve ser mostrada em diversos sistemas, telas, fontes ou métricas. Neste panorama surgem os gerenciadores de *layout*, proporcionando métodos mais flexíveis para dispor os componentes na tela.

Os principais componentes definidos no AWT serão descritos a seguir.

Containers são componentes genéricos, que podem conter outros, incluindo *containers*. O mais comum é o painel (*applet* é um tipo de painel). A aparência dos componentes na tela é determinada pela ordem em que são adicionados ao painel, e pelo gerenciador de *layout* pré-definido que está em uso.

Canvases são superfícies para desenho.

Componentes de interface com o usuário são os mais simples do AWT; podem incluir botões, listas, menus e opções, entre outros. Os mais relevantes para este projeto serão comentados.

Label é um componente textual que pode ser usado como texto descritivo para outro componente.

Buttons podem disparar algumas ações quando são pressionados na interface construída.

Checkboxes podem ser selecionados ou não, promovendo opções. São usados para indicar características opcionais de alguma outra ação.

Choice menus são menus *popup* que possibilitam a seleção de um de seus itens. O item selecionado é mostrado na tela.

Text fields permitem que o usuário informe algum valor. Geralmente, é necessário o uso de um rótulo junto ao campo editável, para informar ao usuário do que se trata. Campos são diferentes de áreas de texto porque têm tamanho limitado; já áreas apresentam barras de rolagem e são mais apropriadas para grandes janelas de texto.

Scrolling list é semelhante a *choice menus*, só que possibilita a seleção de mais de um item da lista e todos dentro de um intervalo são visíveis. Se o número de itens é maior que a caixa da lista, uma barra de rolagem é automaticamente providenciada, para a visualização dos outros itens.

Scrollbars podem gerenciar o conteúdo de uma área de texto ou lista, e ainda, ser usadas para manipular um intervalo de valores, selecionando um desses valores.

Componentes de interface com o usuário produzem eventos e, para interceptá-los, deve-se definir um método action ou handleEvent. Todos os componentes geram eventos a partir de ações do usuário; por exemplo, os botões geram eventos quando são selecionados e opções, quando são marcadas.

Componentes para construção de janelas incluem janelas, *frames*, barras de menu e diálogos.

Existe uma classe, chamada Window, que cria janelas independentes de *browser*. Normalmente, suas subclasses, Frame e Dialog, que são utilizadas. Frame cria toda funcionalidade de uma janela, incluindo barra de menu. Dialog é mais limitada a caixa de diálogo e pode ser usada para mostrar avisos ou solicitar informações especifícas. Vários componentes podem ser adicionados a qualquer uma delas.

Um *browser* que suporta Java provê a janela principal e a barra de menus, por isso, *applets* não precisam definir um Frame. Componentes de interface com o usuário podem ser criados e adicionados a um *applet* sem que um *container* seja criado.

5. Mecanismos de comunicação

Aqui serão descritos os mecanismos de comunicação utilizados em aplicações cliente-servidor implementadas em Java: *sockets*, RMI e CORBA.

5.1. *Sockets*

Até recentemente *sockets* eram o único caminho para escrever aplicações cliente-servidor em Java. O pacote *java.net* contém um conjunto de classes que permitem carregar e manipular objetos URL, além de implementarem *sockets*. Esses serviços correspondem à reputação de Java como linguagem de programação para a Internet.

A utilização de *sockets* é combinada com *streams* Java, que permitem mover dados de e para *sockets* usando a metáfora de arquivos. Com *streams* é possível enviar e receber dados através da rede como ler e escrever dados em arquivos. Para isso utiliza-se o pacote *java.io*.

Sockets têm um nome e um endereço de rede. O endereço em uma rede TCP/IP consiste de um endereço IP (número de 32 bits representado por uma cadeia de caracteres ou por um nome de domínio) e um de porta (número de 16 bits). A porta é um ponto de entrada para uma aplicação que reside em um *host*.

Java suporta dois tipos de *sockets*: datagrama e *stream*. A atenção será concentrada no segundo.

Sockets datagrama provêem interface para UDP (*User Data Protocol*), que trata transmissões como pacotes independentes e sem garantias. Datagramas têm um limite de 32 Kbytes sobre os dados transmitidos. São usados em situações onde é importante o envio de mensagens rápidas sem a necessidade de confirmação de recebimento.

Sockets stream provêem interface para TCP. Este tipo de *socket* garante que pacotes sejam enviados sem erros ou duplicação e recebidos na mesma ordem que foram enviados. Nenhum limite é imposto sobre os dados, que são considerados como *stream* de bytes. O preço de uma classe segura, oferecido por este tipo de *socket* com protocolo orientado à conexão, é o *overhead* associado com a criação e o gerenciamento da sessão.

Há três classes Java que implementam *sockets stream*. Clientes e servidores usam a classe **Socket** para criarem um *socket*, passando a porta e o endereço destino. Também há métodos para obter o objeto *stream* de entrada e saída usado para ler e escrever dados. **ServerSocket** é a classe que servidores usam para "ouvir" conexões requeridas pelos clientes. Pode ser criado um *socket* no lado do servidor que "ouve" conexões de clientes em uma porta local específica. **SocketImpl** é uma classe abstrata que define os métodos que as licenças de JDK devem implementar para suas respectivas plataformas.

A classe Socket provê métodos que associam *sockets* com *streams* (getOutputStream e getInputStream retornam objetos *stream*). Através de *streams* ocorre a transferência de dados, que pode ser entre programas, programas e arquivos, conexões

de rede ou impressoras. Para enviar dados, um *stream* de saída é criado, para receber, um *stream* de entrada.

O pacote *java.io* tem um grande número de classes, a maioria derivada das classes abstratas **InputStream** ou **OutputStream**, definidas quase sempre em pares (uma classe derivada de InputStream com correspondente derivada de OutputStream). Essas classes permitem a definição de *streams "buferizados"* e de dados tipados.

OutputStream define métodos básicos para saída de dados: escrever bytes, forçar a enviar dados *"buferizados"* e fechar *stream*. Todos seus métodos são redefinidos pela classe FilterOutputStream, que permite a filtragem de *streams*, produzindo *streams* compostos e de maior poder. Esta classe apresenta as subclasses: DataOutputStream, que permite escrever tipos de dados primitivos de Java em suas representações binárias e BufferedOutputStream, que armazena o dado em um *buffer* e o escreve quando sua capacidade é alcançada ou quando é forçado a enviar.

InputStream define métodos básicos como: ler próximo byte do *stream*, obter o número de bytes e fechar *stream*, liberando os recursos do sistema que estava consumindo. A hierarquia corresponde a OutputStream. DataInputStream, que permite ler tipos de dados primitivos, e BufferedInputStream, que lê os dados e os armazena em *buffers*, são *streams* de filtragem derivadas de FilterInputStream.

Streams "buferizados" permitem aumentar o desempenho, pois trabalham com (escreve-se ou lê-se) uma porção de dados ao invés de um byte por vez.

Quando se tem várias interfaces oferecidas pelo servidor a seus clientes, e estas consistem em vários métodos que requerem vários parâmetros, é difícil implementar estes serviços usando *sockets*. Nestes casos, ORBs (*Object Request Broker*) são mais adequados, pois detalhes da distribuição são escondidos e a programação ocorre sobre um alto nível de abstração.

5.2. RMI – *Remote Method Invocation*

RMI é parte de JDK 1.1 e foi desenvolvido para suportar invocações de métodos remotos sobre objetos localizados em máquinas virtuais Java. RMI integra um modelo de objetos distribuídos, tornando o ORB quase transparente ao cliente. [ORFALI, 1997]

Um objeto RMI é um objeto Java remoto, cujos métodos podem ser invocados de uma outra máquina virtual Java, mesmo através da rede. Esta invocação é feita como se o objeto fosse local.

Objetos RMI podem interagir somente com outros objetos RMI e não suportam invocações dinâmicas. Há a possibilidade da passagem de um objeto remoto por referência, como argumento ou retorno de um método.

Clientes RMI interagem com objetos remotos via suas interfaces, nunca diretamente com as classes que as implementam.

Uma invocação RMI passa objetos locais como argumento por cópia (ou valor), porque a referência a estes objetos é usada somente em uma única máquina virtual.

Em um sistema distribuído um coletor de lixo deve ser capaz de deletar, automaticamente, objetos remotos não referenciados por algum cliente. RMI usa um contador de referência, que mantém a informação de uso externo, para objetos servidores de uma máquina virtual Java.

RMI provê novas interfaces e classes que possibilitam encontrar, carregar e executar objetos remotos com segurança. RMI estende classes de exceção para tratamento de falhas remotas.

O processo de desenvolvimento de uma aplicação cliente-servidor RMI consiste nos seguintes passos:

1. Definir uma interface remota: o objeto servidor deve declarar seus serviços via uma interface remota.
2. Implementar a interface remota: uma classe servidora Java deve implementar a interface publicada.
3. Compilar a classe servidora.
4. Executar o compilador de *stub*, chamado *rmic*: gera *stubs* cliente e *skeletons* servidor para as classes remotas. Os *stubs* ordenam e enviam chamadas remotas ao servidor. Os *skeletons* recebem as chamadas e direcionam às suas classes de implementação.
5. Iniciar o registro RMI do servidor: RMI define interfaces para um serviço de nomes não-persistente. Isto permite o registro de objetos servidores usando simples nomes.

6. Iniciar objetos servidores: carregar classes e criar instâncias de objetos remotos.
7. Registrar objetos remotos: registrar todas as instâncias de objetos remotos, assim clientes podem acessá-los. Após este passo, objetos remotos estão prontos para serem invocados.
8. Escrever código cliente: consiste em código Java. Para usar objetos remotos basta localizá-los e invocar seus métodos via *stub*.
9. Compilar código cliente.
10. Iniciar o cliente: carregar as classes clientes e seus *stubs*.

RMI é um tipo de ORB construído sobre o modelo de objetos Java, que introduziu algumas inovações: permite mover código além de dados; garante que o código carregado é seguro; permite passagem de objetos por valor; usa Java para definir interface e implementação; usa um esquema de nomes baseado em URL.

RMI pode ser atrativo para programadores Java, mas o limite da infra-estrutura, onde objetos RMI só interagem entre si, deve ser considerado, quando se busca um mecanismo de comunicação distribuído.

5.3. CORBA – *Common Object Request Broker Architecture*

CORBA é o produto do consórcio formado pela OMG (*Object Management Group*), que define um modelo para objetos distribuídos. Como alternativa à utilização de RMI, CORBA apresenta a vantagem de independência de linguagem de programação. Serviços remotos são definidos através de interfaces portáveis a linguagens de programação, sistemas operacionais e redes. O suporte a CORBA foi introduzido na versão 1.2 de JDK.

Apêndice II.

Implementação das interfaces gráficas do ambiente

O ambiente para criação de animações interativas é definido como uma aplicação Java. Todas as interfaces gráficas, desenvolvidas nos diversos níveis de aplicações do ambiente apresentados na Seção 3.6, foram implementadas com a utilização do AWT. As janelas principais são derivadas da classe Frame, e as outras, exibidas a partir dessas, são janelas de diálogo, cada uma correspondendo a uma classe derivada de Dialog.

As janelas (das classes Frame ou Dialog) são compostas de painéis e/ou componentes de interface gráfica, básicos do AWT. Painéis agrupam outros componentes, e tornam-se necessários por razões de *layout* ou pela reusabilidade que podem proporcionar.

O *layout* utilizado em todo o sistema foi *GridBagLayout*, pela maior versatilidade que apresenta. Seguindo este *layout*, uma classe foi disponibilizada para prover uma forma mais fácil de criação e disposição dos componentes gráficos em *containers*. Esta classe, chamada MakeObjects, é independente e pode ser usada pelo programador para desenvolver qualquer outro tipo de interface que utilize o *layout GridBagLayout*.

Cada classe derivada de Frame ou Dialog abrange todo o tratamento de eventos que possa ser gerado por seus componentes gráficos. A captação dos eventos é feita pelos métodos handleEvent e action.

Quando uma opção, da janela principal de qualquer uma das aplicações, é selecionada, uma instância de uma classe, derivada de Dialog, é criada e exibida. Quando o botão "*Add...*" das janelas de diálogo é pressionado, um método da classe de gerenciamento é chamado para gerenciar a inclusão de um objeto (componente gráfico, no caso da aplicação IULib, e campo, variável ou evento, no caso da ScrLib). As janelas

são fechadas quando o processo de inclusão acaba ou quando o botão "*Cancel*" é pressionado.

1. Interfaces gráficas do ambiente

A seguir serão apresentadas todas as janelas que podem ser obtidas no ambiente para a construção de animações interativas. Algumas dessas janelas podem ser exibidas com pequenas variações, como botões habilitados ou desabilitados, dependendo da operação. Em seguida esquemas mostram a seqüência em que as janelas são exibidas para operações específicas.

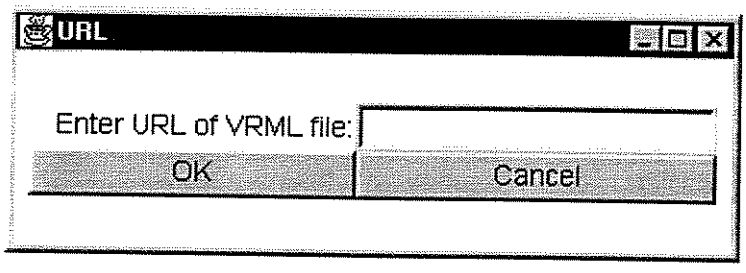

Figura II.1: Informação de URL correspondendo a uma cena VRML

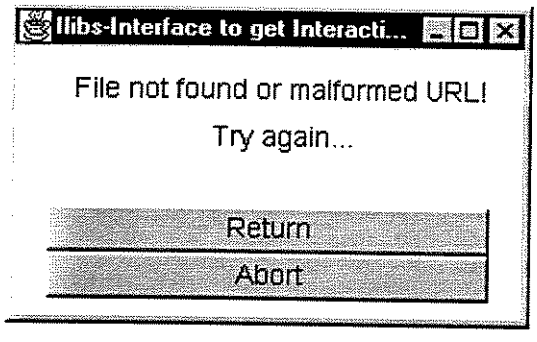

Figura II.2: Mensagem de URL errada

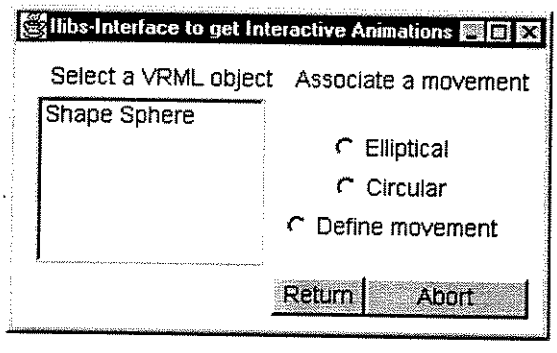

Figura II.3: Objetos VRML e movimentos

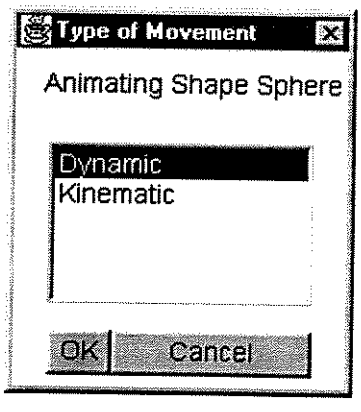

Figura II.4: Movimento circular

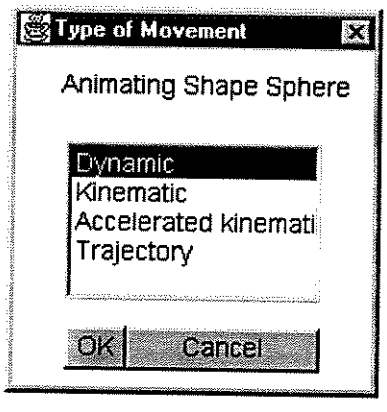

Figura II.5: Movimento elíptico

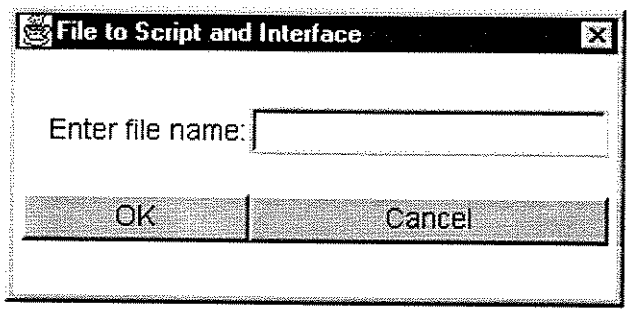

Figura II.6: Informação do nome dos arquivos a serem gerados

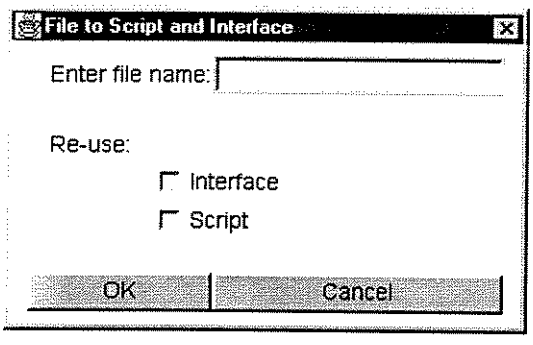

Figura II.7: Informação do nome dos arquivos a serem gerados e opção para reutilizar movimentos

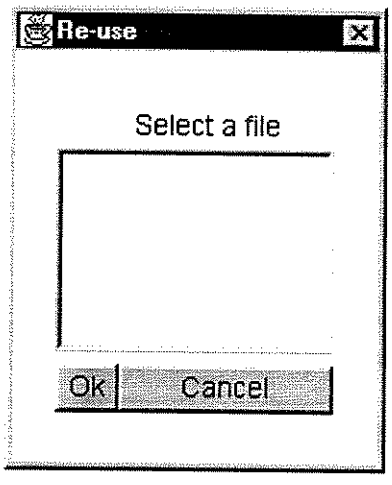

Figura II.8: Reutilizando movimento

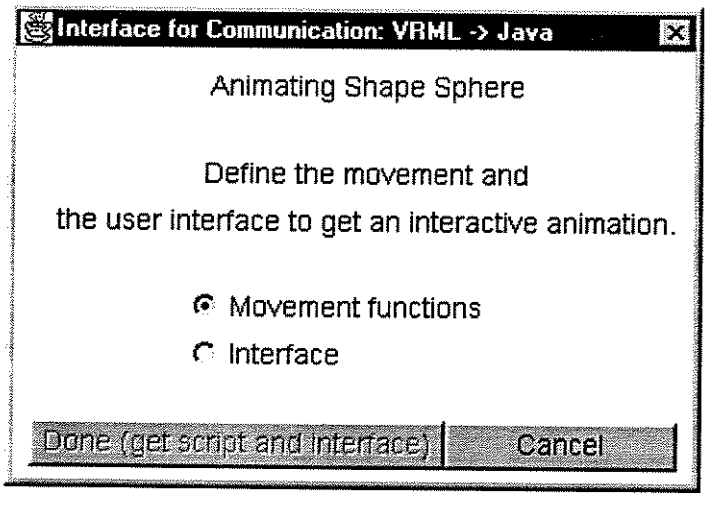

Figura II.9: Definindo movimento

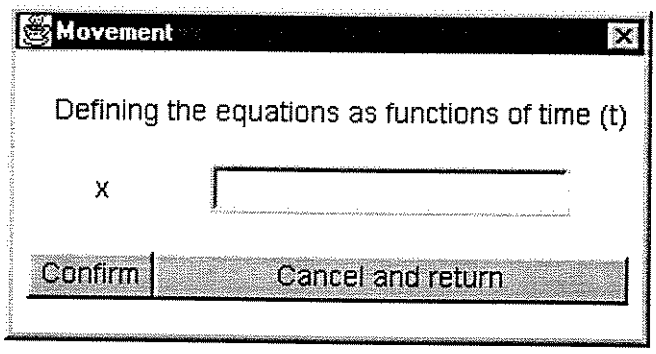

Figura II.10: Definindo equações do movimento: x, y e z

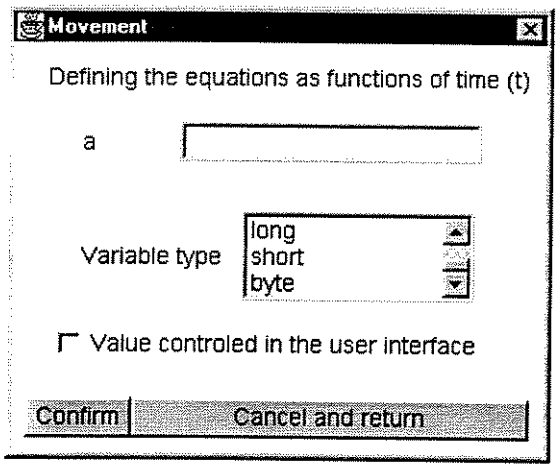

Figura II.11: Definido variáveis

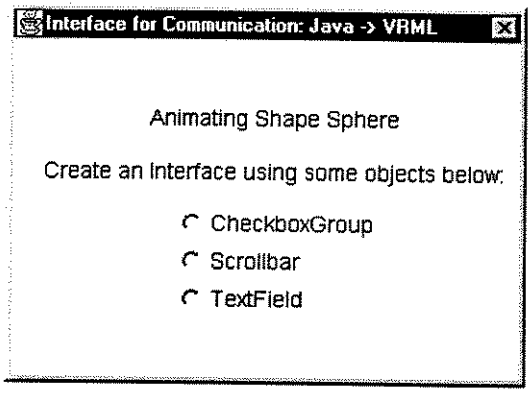

Figura II.12: Definindo variável como parâmetro da animação

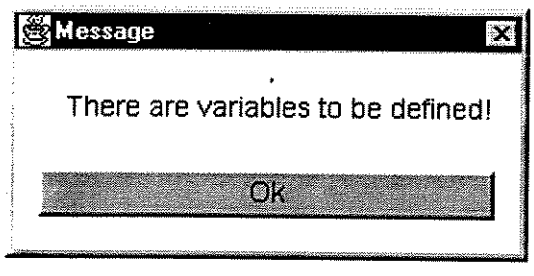

Figura II.13: Definição de variáveis incompleta

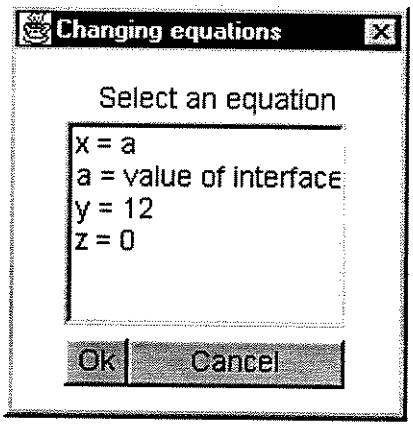

Figura II.14: Definição de variáveis concluídas

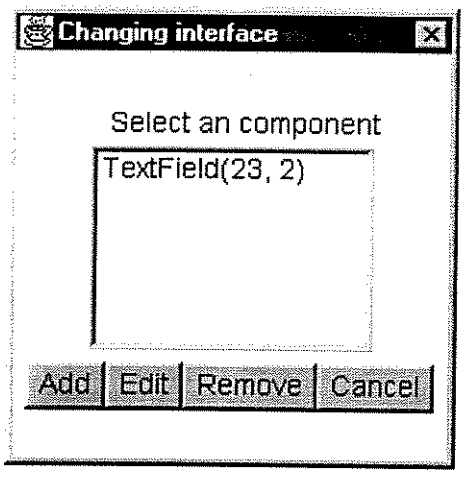

Figura II.15: Exibição dos componentes gráficos no painel de controle

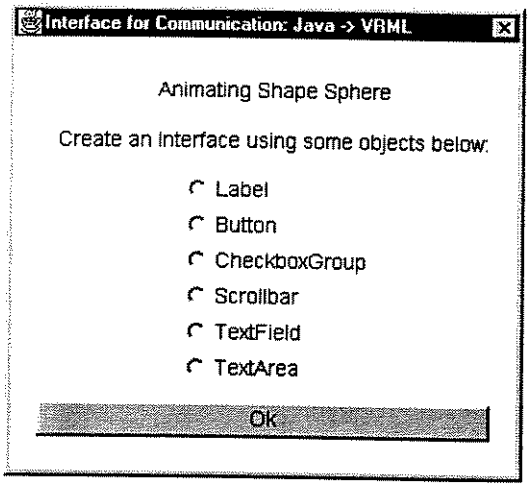

Figura II.16: Adicionando componentes gráficos no painel de controle

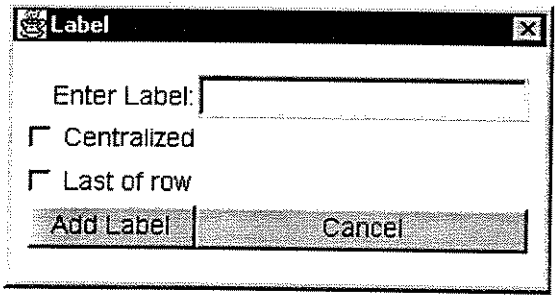

Figura II.17: Inserindo rótulo

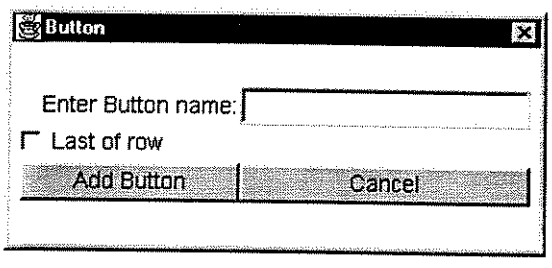

Figura II.18: Inserindo botão

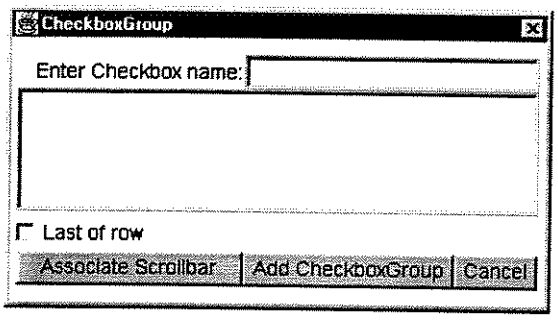

Figura II.19: Inserindo grupo de opções

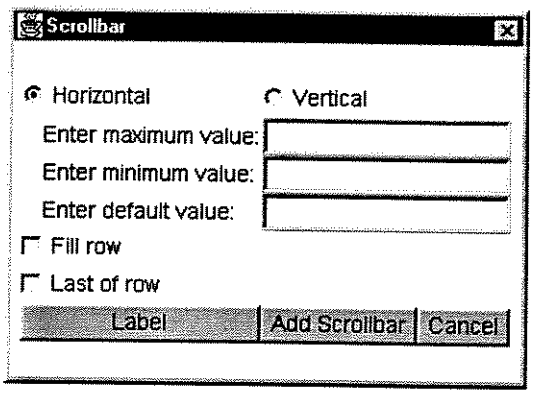

Figura II.20: Inserindo barra de rolagem

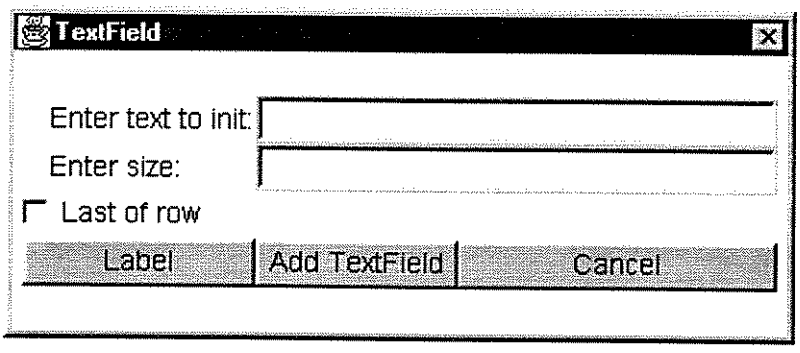

Figura II.21: Inserindo campo de texto

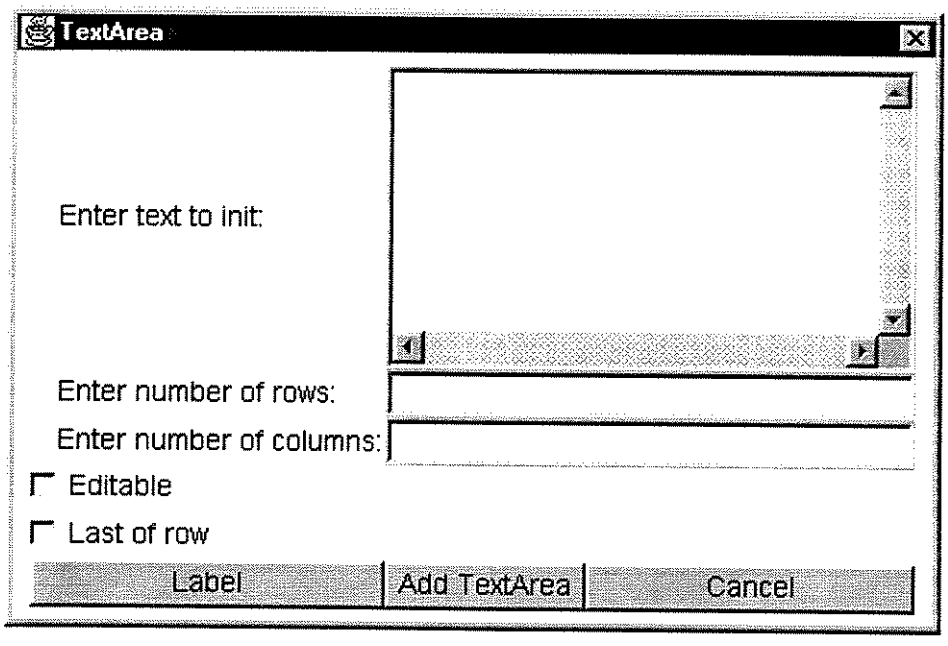

Figura II.22: Inserindo área de texto

As figuras mostradas a seguir definem, na forma de árvore, a seqüência que as janelas apresentadas anteriormente são exibidas ao animador durante o processo de utilização do ambiente desenvolvido. Os nodos dessas árvores são definidos com um número que representa o número da figura em que a janela foi apresentada, excluindo o número deste apêndice. Os caminhos a serem seguidos, ou seja, as janelas que são exibidas, dependem da ação do animador. O nodo ⊙ é definido como final, onde o animador pode sair do sistema de janelas, após ter construído uma animação interativa ou não. A seqüência definida a partir de um nodo, geralmente não é repetida; portanto, quando um nodo (diferente do final) não apresentar derivações, a sua seqüência já definida pode ser considerada. A Figura II.23 mostra as possíveis janelas exibidas na construção de animações interativas com a utilização da biblioteca.

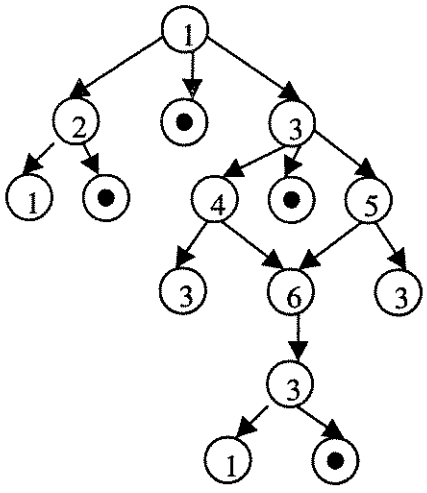

Figura II.23: Utilizando movimentos da biblioteca

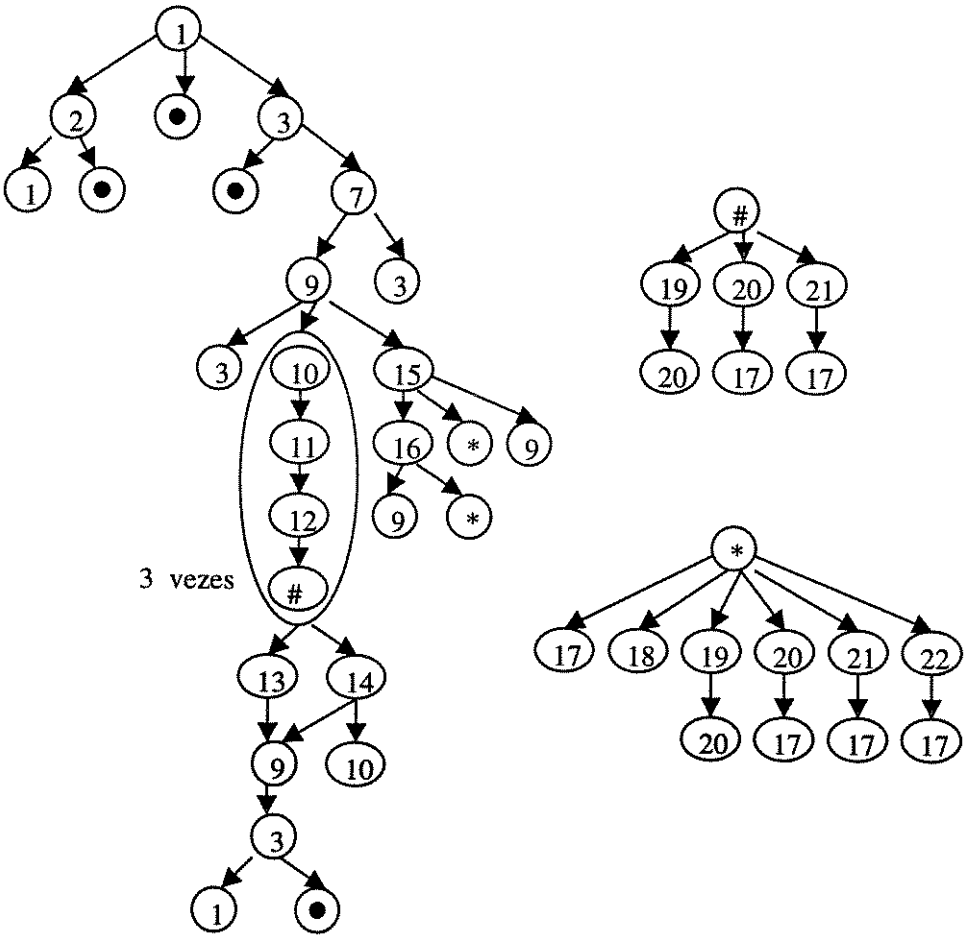

Figura II.24: Definindo movimento

Nas Figuras II.24 e II.25 são mostradas as árvores com as janelas exibidas para a construção de animações interativas com definição e reutilização de movimento,

respectivamente. Nestas figuras dois nodos (*) e (#) foram utilizados para auxiliar no desenho. Também um agrupamento de nodos (10, 11, 12 e #) pode ser observado, indicando que a seqüência destas janelas pode ser exibida até três vezes, uma para cada equação correspondendo aos eixos *x*, *y* e *z*.

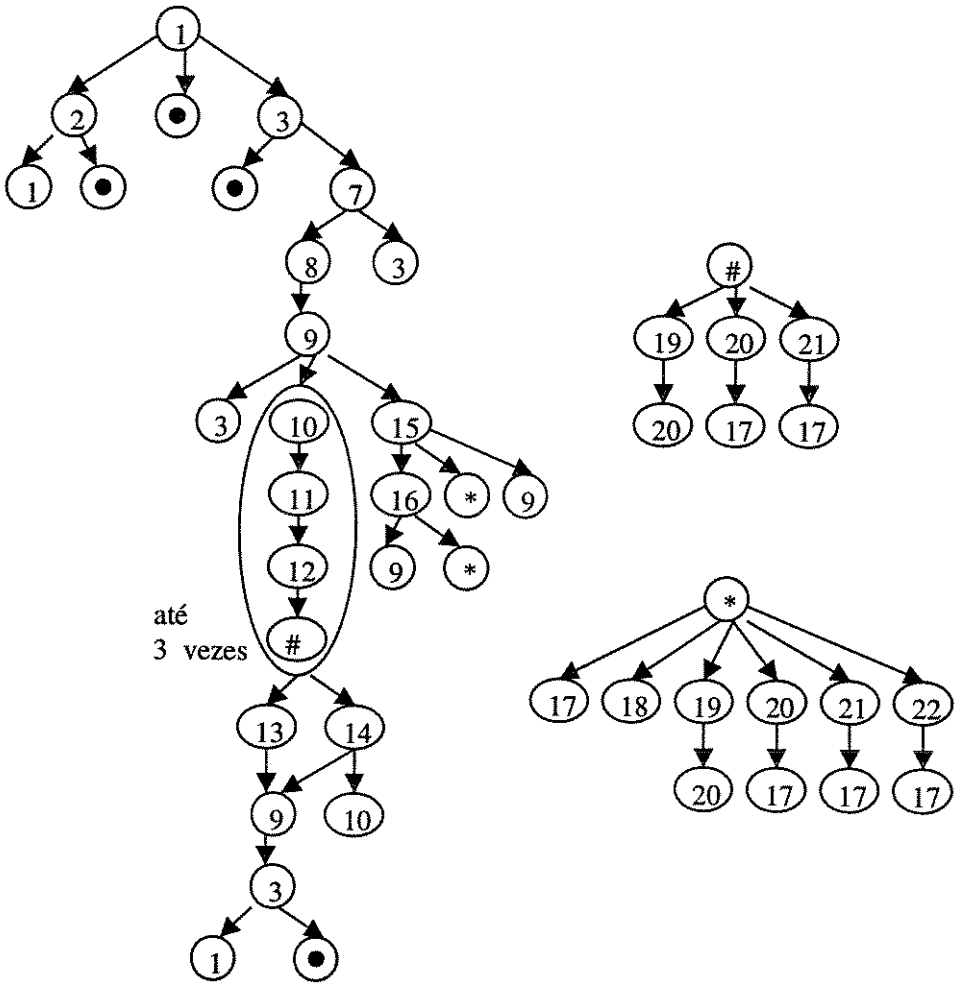

Figura II.25: Reutilizando movimento

Apêndice III.

Implementação da animação interativa

Neste apêndice as animações interativas geradas pelo ambiente terão as questões de implementação abordadas. O modelo de comunicação estabelecido entre o painel de controle e *script* VRML será apresentado com detalhes de codificação.

Os arquivos Java gerados pelo ambiente na criação de uma animação interativa, correspondentes ao painel de controle e *script* VRML são apresentados, na forma de modelo, excluindo informações cuja responsabilidade é do animador, tais como, componentes gráficos e equações.

1. Comunicação entre os arquivos gerados pelo ambiente

No cenário de uma animação interativa o *script* VRML recebe dados do painel de controle, informados pelo usuário. A conexão entre o painel de controle e o *script* é responsável pela transmissão dos parâmetros de controle da animação definidos pelo usuário.

A comunicação entre o painel de controle e o *script* é baseada no modelo cliente-servidor. O painel de controle definido pelo animador na interação com o ambiente é o servidor e o *script*, o cliente. O arquivo VRML dispara o *script*, que obtém conexão com o painel de controle e requisita dados. O painel de controle, então, envia os dados requisitados, o *script* processa-os e retorna o resultado ao arquivo VRML. Esses passos são esquematizados na Figura III.1.

A comunicação é feita com a utilização de *sockets*. Os métodos de transmissão são baseados nos pacotes padrões de Java: *java.net* e *java.io*. O *script* também é capaz de utilizar os mesmos pacotes para receber os dados.

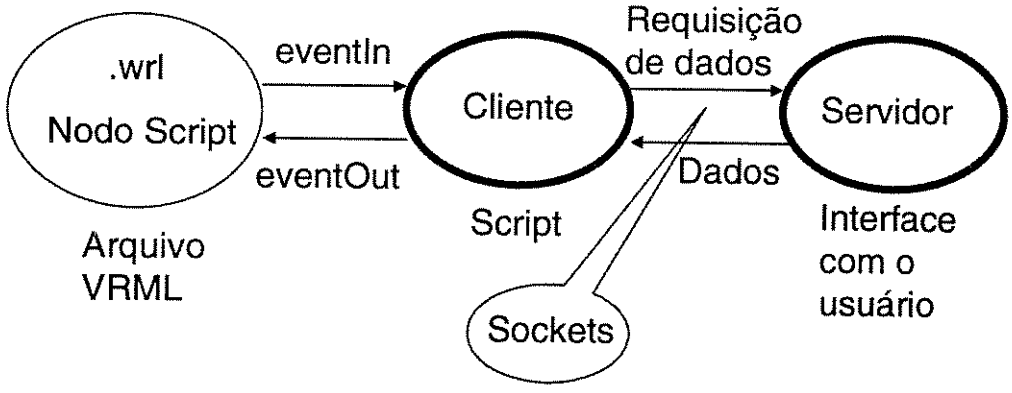

Figura III.1: Comunicação entre painel de controle, *script* e arquivo VRML

O servidor e o cliente são definidos para executarem na máquina *host* do usuário sobre uma porta especificada. Estas definições são pré-definidas e podem ser incluídas em um arquivo de lote, tornando transparente aos usuários.

A comunicação entre o arquivo VRML e o *script* seguem o padrão proposto pela especificação VRML, por esta razão não será detalhada.

2. Arquivo Java gerado para um painel de controle

Para um painel de controle é gerado um arquivo fonte Java (.java) incluindo duas classes. Uma é responsável por todo tratamento que os componentes gráficos necessitam e a outra, pela transmissão de dados ao *script* VRML. A classe responsável pela transmissão de dados do usuário final usa métodos da outra classe (responsável pela interface gráfica) para obter tais dados, já que esta segunda tem a informação de quais componentes gráficos são associados a parâmetros de controle da animação, e assim, gera os métodos de acesso necessários. Desta forma, o arquivo tem a seguinte estrutura, onde os principais pontos estão comentados com destaque:

```
//Imports necessários
import java.awt.*;
import java.util.*;
import java.io.*;
import java.net.*;
import makeobjects;

//Classe responsável pela interação com o usuário
public class IUnome extends Frame {
public IUnome() {
      //Criação do Servidor
      serv = new ControlServer(this);
      //Definição do layout
      GridBagLayout gridbag = new GridBagLayout();
      GridBagConstraints c = new GridBagConstraints();
      setFont(new Font("Helvetica", Font.PLAIN, 14));
```

```
        setLayout(gridbag);
        //Criação dos componentes gráficos
        makeobjects m = new makeobjects();
        c.fill = GridBagConstraints.HORIZONTAL;
        cbg0 = new CheckboxGroup();
        ...
}
//Tratamento de eventos
public boolean handleEvent(Event evt) {
        if (evt.target instanceof Checkbox) {
        ...
        }
        ...
        return super.handleEvent(evt);
}
//Definição do main
public static void main (String args[]) {
        IUnome win = new IUnome();
        win.setTitle("Control Application");
        win.resize(500, 270);
        win.show();
}
//Definição dos componentes gráficos
private makeobjects m;
...
//Métodos para retornar valores de componentes gráficos
public String returncbg0() {
        Checkbox cb = new Checkbox();
        cb = cbg0.getCurrent();
        if (cb.equals(c1))
              return c1.getLabel();
        ...
}
//Finaliza servidor
public void stop () {
        serv.closeEverything();
        serv.stop();
}
}

//Classe responsável pela transmissão de dados
class ControlServer extends Thread {
boolean clientConnected = false;
Socket incoming;
IUnome app;
DataInputStream in = new DataInputStream (System.in);
PrintStream out = System.out;
//Construtor
ControlServer(IUnome pc) {
        app = pc;
        this.start();
}
//Roda a thread
public void run() {
        try {
              ServerSocket ss = new ServerSocket
(Integer.parseInt(System.getProperty("port")));
              incoming = ss.accept();
              in = new DataInputStream(incoming.getInputStream());
              out = new PrintStream(incoming.getOutputStream());
              clientConnected = true;
```

```
        } catch (Exception e) { System.out.println(e); }
        while(true) {
              if (clientConnected) {
                    try {
                          String s = in.readLine();
                          sendToClient(app.returncbg0());
                          ...
                    } catch (Exception e) { System.out.println(e); }
              }
              try {
                    sleep(5);
              } catch (Exception e) { System.out.println(e); }
        }
}
//Envia dados
public void sendToClient(String value) {
        out.println(value);
}
//Finaliza comunicação
public void closeEverything() {
        try {
              incoming.close();
        } catch (Exception e) { System.out.println(e); }
}
}
```

O painel de controle é uma aplicação Java, estendida da classe Frame. A classe terá o nome dado pelo animador, precedido por "IU". O construtor desta classe usa basicamente componentes de interface gráfica oferecidos pelo ambiente (e conseqüentemente pelo AWT) e escolhidos pelo animador. Os componentes são dispostos sobre o *layout GridBagLayout*, com a utilização da classe makeobjects. O tratamento de eventos é pré-determinado e feito pelo método handleEvent. Esta classe consta com métodos de retorno de valores, correspondendo a componentes associados à animação que devem enviar seus valores ao *script* VRML. A relação é um método para cada componente. Esses métodos são chamados da classe responsável pela transmissão dos valores.

Seguindo o modelo cliente-servidor, a classe ControlServer é responsável pela comunicação no lado do servidor, ou seja, envia dados ao cliente. Esta classe, que estende a Thread, é instanciada no construtor, para começar a receber requisições do cliente e fica executando durante toda a vida da aplicação, ou seja, envia valores até que a aplicação seja encerrada. Esta *thread* cria um *socket* e *streams* para estabelecer a comunicação. A chegada de uma requisição faz com que o método sendToClient seja chamado diversas vezes, dependendo do número de valores a serem enviados. Neste ponto, os métodos de retorno de valores de componentes gráficos da outra classe são invocados.

Todos os painéis de controle gerados apresentarão a classe ControlServer da mesma forma, diferindo somente na quantidade de valores enviados (número de chamadas ao método sendToClient).

3. Arquivo Java gerado para um *script* VRML

A seguir, em um exemplo, a estrutura do *script* VRML gerado pelo ambiente na criação de animações interativas, é apresentada com os principais pontos comentados em destaque.

```
//Imports necessários
import vrml.*;
import vrml.field.*;
import vrml.node.*;
import java.io.*;
import java.net.*;

public class Snome extends Script {
//Declara fields, eventos, variáveis, socket e streams.
private SFFloat currentTimer;
float[] translation;
private SFVec3f position;
Socket soc;
DataInputStream in;
PrintStream out;
//Inicializa fields, eventos, variáveis. Cria sockets e streams.
public void initialize() {
      translation = new float[3];
      currentTimer = (SFFloat) getField("currentTimer");
      position = (SFVec3f) getEventOut("position");
      try {
            soc = new Socket(host, port);
            in = new DataInputStream(soc.getInputStream());
            out = new PrintStream(soc.getOutputStream());
      } catch (Exception e) { System.out.println(e); }
}
//Recebe eventos de entrada
public void processEvent(Event e) {
      if (e.getName().equals("timerIn"))
            currentTimer.setValue((ConstSFFloat)e.getValue());
}

public void eventsProcessed() {
int a=0;
...
      //Requisita valores
      out.print("\n");
      try {
            //Recebe valores
            a = Integer.parseInt(in.readLine());
            ...
      } catch (IOException ioe) {
            System.out.println("IOException: " + ioe);
      }
      //Calcula movimento
      translation[0] = a*(float)currentTimer.getValue();
```

```
        ...
        //Determina eventos de saída
        position.setValue(translation);
}
}
```

Esta classe estende a Script e deve implementar muitos de seus métodos. Para que isso seja possível, este arquivo deve importar pacotes Java oferecidos pelos *browsers* VRML que seguem a especificação, no que diz respeito a linguagens de *script*.

A classe derivada de Script é nomeada pelo animador, com um "S" precedendo. Os campos, eventos e variáveis são definidos como atributos da classe e utilizados nos métodos apropriados. Um *socket* e *streams* são definidos para estabelecerem comunicação do lado cliente, quando o *script* é inicializado. No método eventsProcessed, dados são requisitados e valores inteiros são recebidos na mesma ordem que foram enviados do servidor. Com esses dados, o *script* calcula o movimento e gera eventos que podem alterar a cena VRML associada.

Referências Bibliográficas

[CAMARGO, 1995] J. T. F. de Camargo, L. P. Magalhães e A. B. Raposo. Tutorial: *Fundamentos da Animação Modelada por Computador*. Simpósio Brasileiro de Computação Gráfica e Processamento de Imagens, SIBGRAPI'95. São Carlos, SP. 1995.

[FOWLER, 1997] M. Fowler e K. Scott. Foreward by Grady Booch, Ivar Jacobson, James Rumbaugh. *UML distilled – Applying the standard object modeling language*. Addison-Wesley. 1997.

[GREENWOOD, 1965] D. T. Greenwood. *Principles of dynamics*. 1965.

[ISAACS, 1987] P. M. Isaacs, et al. *Controlling dynamic simulation with kinematic constraints, behaviour functions and inverse dynamics*. ACM Computer Graphics, v.21, n.4, pp.215-224. Julho, 1987.

[KALRA, 1992] D. Kalra e A. H. Barr. *Modeling with Time and Events in Computer Animation*. Eurographics'92. Volume 11, número 3, pp. 45-48. 1992.

[LEMAY, 1996] L. Lemay e C. L. Perkins. *Teach yourself Java in 21 days*. Sams net. 1996.

[MAGALHÃES, 1997] L. P. Magalhães, A. B. Raposo e F. S. Tamiosso. *VRML 2.0 – An Introductory view by examples*. 1997. (http://www.dca.fee.unicamp.br/~leopini/tutorial/vrml-tut.html)

[ORFALI, 1997] R. Orfali e D. Harkey. *Client/Server Programming with Java and CORBA*. John Wiley & Sons, Inc. 1997.

[RAPOSO, 1997a] A. B. Raposo e L. P. Magalhães. *Uma linguagem para desenvolvimento de roteiros de animação*. Relatório técnico DCA 97-003. 1997. (ftp://ftp.dca.fee.unicamp.br/pub/docs/techrep/1997/DCA97-003.ps.gz)

[RAPOSO, 1997b] A. B. Raposo e L. P. Magalhães. *Using Petri Nets for Animation Modeling and Analysis*. Relatório técnico DCA 97-005. 1997. (ftp://ftp.dca.fee.unicamp.br/pub/docs/techrep/1997/DCA97-005.ps.gz)

[SGI, 1997] Silicon Graphics, Inc. Cosmo Player, 1997. (http://vrml.sgi.com/cosmoplayer/)

[SONY, 1997] Sony Corporation. Community Place, 1997. (http://vs.spiw.com/vs/)

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

[TAMIOSSO, 1997a] 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.

[TAMIOSSO, 1997b] F. S. Tamiosso. *VRML 2.0 – Um resumo baseado em exemplos*. 1997. (http://www.dca.fee.unicamp.br/~fabiana/vrml/tutorial.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/).