**UNIVERSIDADE REGIONAL DE BLUMENAU**

**CENTRO DE CIÊNCIAS EXATAS E NATURAIS**

**CURSO DE CIÊNCIAS DA COMPUTAÇÃO**

(Bacharelado)

# PROTÓTIPO DE UM MUNDO VIRTUAL DISTRIBUÍDO UTILIZANDO DIS-JAVA-VRML

TRABALHO DE CONCLUSÃO DE CURSO SUBMETIDO À UNIVERSIDADE REGIONAL DE BLUMENAU PARA A OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA COM NOME EQUIVALENTE NO CURSO DE CIÊNCIAS DA COMPUTAÇÃO — BACHARELADO

**EDISON WEISE**

BLUMENAU, JUNHO/2002.

2002/1-25

# PROTÓTIPO DE UM MUNDO VIRTUAL DISTRIBUÍDO UTILIZANDO DIS-JAVA-VRML

**EDISON WEISE**

ESTE TRABALHO DE CONCLUSÃO DE CURSO FOI JULGADO ADEQUADO PARA OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA DE TRABALHO DE CONCLUSÃO DE CURSO OBRIGATÓRIA PARA OBTENÇÃO DO TÍTULO DE:

**BACHAREL EM CIÊNCIAS DA COMPUTAÇÃO**

Prof. Dalton Solano dos Reis - Orientador na FURB

Prof. José Roque Voltolini da Silva - Coordenador do TCC

**BANCA EXAMINADORA**

Prof. Dalton Solano dos Reis

Prof. Paulo César Rodacki Gomes

Prof. Marcel Hugo

# AGRADECIMENTOS

Agradeço primeiramente a Deus que me deu a oportunidade de realizar este trabalho, dando-me esperança e força para seguir em frente nos momentos difíceis e também alegrias ao superar os desafios. Um agradecimento especial a aqueles que estiveram juntos comigo e me apoiaram em todos os momentos:  meus pais Nilson e Inge, minha irmã Christiane e minha noiva Suzana.

Agradecimentos merecidos ao meu orientador professor Dalton Solano dos Reis, cujo conhecimento na área e atenção para comigo, foram de grande importância para o desenvolvimento desse trabalho.

# SUMÁRIO

# LISTA DE FIGURAS

# LISTA DE QUADROS

# LISTA DE TABELAS

# RESUMO

Este trabalho apresenta um estudo sobre os ambientes virtuais distribuídos, abordando desde conceitos básicos sobre a realidade virtual e ambientes virtuais até a construção e comunicação entre estes mundos num ambiente distribuído. O trabalho ainda apresenta fatores estudados sobre o protocolo DIS (*Distributed Interactive Simulation*), o qual teve suas classes escritas na linguagem Java, também utilizada neste trabalho juntamente com a linguagem de modelagem VRML (*Virtual Reality Modeling Language*). Concentrando-se no estudo da tecnologia DIS-Java-VRML, este trabalho tem como resultado a implementação de um protótipo de ambiente virtual distribuído em uma rede local utilizando a tecnologia anteriormente citada.

# ABSTRACT

This work presents a study about networked virtual environments, approaching since basics concepts about the virtual reality and virtual environments to the construction and communication between these worlds in a networked environment. The work still presents a study about the DIS (*Distributed Interactive Simulation*) protocol, that has classes written in Java language, too exploit in this work with the VRML (*Virtual Reality Modeling Language*) modeling language. Focusing DIS-Java-VRML technology, this work has as result the implementation of prototype of a networked virtual environment at a local area network applying the technology previously quoting.

# 1 INTRODUÇÃO

## 1.1 CONTEXTUALIZAÇÃO / JUSTIFICATIVA

Cada vez mais a computação gráfica tem sido utilizada para a criação de imagens que representam modelos do mundo real. Segundo Gradecki (1994), é nos mundos virtuais que se experimenta novas realidades. Estes respectivos mundos podem ser compostos por vários ambientes virtuais. O termo "ambiente virtual" é utilizado para descrever o ambiente com o qual se interage quando se utiliza um sistema de realidade virtual. De acordo com Pinho (1999) um ambiente virtual pode ser projetado para simular tanto um ambiente imaginário quanto um ambiente real, podendo estar classificado em ambiente virtual imersivo ou ambiente virtual não imersivo.

Os ambientes virtuais imersivos são aqueles onde as imagens são exibidas ao usuário utilizando dispositivos especiais de tal forma que o faça acreditar que está imerso neste ambiente. Já nos ambientes virtuais não imersivos as imagens são exibidas diretamente em um monitor de vídeo ou computador (Pinho, 2000).

Recentemente os ambientes virtuais têm sido expandidos para permitir a participação de diversos usuários, conectados através de um sistema distribuído, chamados de ambientes virtuais distribuídos (Sementille, 2000). Os sistemas de realidade virtual multi-usuários em ambiente distribuído vêm crescendo e apresentam elevado potencial de aplicação. Esse tipo de sistema permite que os usuários geograficamente dispersos atuem em mundos virtuais compartilhados, usando a rede para melhorar o desempenho coletivo, através da troca de informações (Kirner, 2000).

Existe uma preocupação com os ambientes virtuais distribuídos, segundo Sementille (2000), em grande parte justificada pelo enquadramento dos mesmos como sistemas distribuídos de tempo real, e devido a isto, os esforços de pesquisa têm se concentrado, especialmente, na minimização da latência[1] de comunicação. Várias são as soluções atualmente em uso e em pesquisa, podendo-se citar, principalmente, o protocolo *Distributed Interactive Simulation* (DIS).

[1] No contexto das redes de computadores, pode-se dizer que latência é o período de tempo decorrido, medido em unidades de tempo, entre envio de um pacote e o seu recebimento no computador de destino.

Segundo Macedonia (1995), o protocolo DIS é um grupo de padrões, definido pelo Departamento de Defesa dos Estados Unidos e indústrias interessadas, preocupados em determinar uma arquitetura de comunicação comum, definindo "o formato e o conteúdo dos dados comunicados; informações a respeito dos objetos do mundo virtual e sua interação; gerenciamento da simulação; medidas de desempenho; comunicações de rádio; segurança; fidelidade; controle de exercícios, etc".

Outro ponto importante para a construção de um mundo virtual é a linguagem a ser utilizada. Neste trabalho utilizou-se a linguagem de programação Java que de acordo com Flanagan (2000) é uma linguagem avançada, orientada para o objeto, com uma sintaxe semelhante à da linguagem C, que além de permitir a utilização de primitivas gráficas é uma linguagem propícia para a *World Wide Web* (WEB).

Em conjunto com a linguagem de programação Java também foi utilizada a linguagem VRML que é o acrônimo de *Virtual Reality Modeling Language*. Esta linguagem permite que se apresentem objetos e mundos tridimensionais através da WEB, não somente através de cenas estáticas, mas também permitindo que o visitante de um "Mundo Virtual", interaja com os objetos deste mundo. Isto é importante, pois segundo Gradecki (1994) a realidade virtual deve permitir ao usuário interagir com os objetos no ambiente. Para isto, foi necessário a utilização do avatar, que é a representação de um usuário no ambiente virtual, podendo ser tipicamente um usuário comum (uma pessoa) ou uma entidade automatizada que representa um processo (entidade limitada que executa ações dentro de um mundo virtual) (Sementille, 2000).

Contudo, este trabalho teve como principal ponto estudar a tecnologia DIS-Java-VRML. De acordo com Web3D (2000a), o DIS, Java e VRML podem fornecer todos os recursos pertinentes e necessárias para a implementação de ambientes virtuais. DIS é essencialmente um protocolo direcionado ao comportamento físico - baseado em várias interações. Pode-se dizer que é utilizado para comunicar informações de estado como: posição, orientação, velocidade e aceleração dentre entidades múltiplas participando em um ambiente de rede compartilhado. Java é a linguagem de programação utilizada para implementar o protocolo DIS, desempenha cálculos matemáticos, comunica com a rede e com a cena em VRML. Já a VRML é utilizada para modelar as entidades locais e remotas em mundos virtuais distribuídos.

Dessa forma, este trabalho permite que se tenha uma visão geral de algumas características importantes que precisam ser observadas na construção de ambientes virtuais

distribuídos e como o estudo da tecnologia DIS-Java-VRML foi utilizado para a implementação do protótipo.

## 1.2 OBJETIVOS

O objetivo do trabalho é implementar um protótipo de um mundo virtual distribuído e não imersivo sobre uma rede local, utilizando a tecnologia DIS-Java-VRML.

Os objetivos específicos do trabalho são:

a) representar objetos por primitivas gráficas simples;
b) permitir a interação com o mundo virtual através de um avatar.

## 1.3 ORGANIZAÇÃO DO TRABALHO

O trabalho está organizado conforme descrito abaixo.

O capítulo dois apresenta conceitos sobre a realidade virtual, concentrando-se na abordagem dos ambientes virtuais e sua evolução, até o surgimento dos ambientes virtuais distribuídos.

O capítulo três apresenta as principais características da linguagem VRML, bem como o seu histórico, situação atual, algumas aplicações, e um tutorial básico da linguagem. Já o capítulo quatro apresenta conceitos sobre o protocolo DIS, a linguagem de programação Java trabalhando com a VRML, e a tecnologia DIS-Java-VRML.

O capítulo cinco apresenta a especificação, implementação e o funcionamento do protótipo desenvolvido utilizando a tecnologia DIS-Java-VRML. Por fim, no capítulo seis são apresentadas as conclusões provenientes da execução desse trabalho, bem como as possíveis extensões que dele podem ser desenvolvidas.

# 2 AMBIENTES VIRTUAIS DISTRIBUÍDOS

## 2.1 REALIDADE VIRTUAL

Nos últimos anos, a realidade virtual deixou de ser um assunto restrito aos centros de pesquisa e chegou bem próximo do usuário. Um dos fatores que contribuiu para que isto acontecesse foi a possibilidade de gerar aplicações em PC, que há pouco tempo eram possíveis somente em *workstations* (Peruzza, 2000). Outro fator muito importante e decisivo na popularização da realidade virtual, segundo Peruzza (2000), foi a criação da linguagem VRML que colocou a realidade virtual na internet, despertando o interesse de muitos usuários.

Com isso, a utilização da realidade virtual começou a crescer e hoje, com o seu elevado potencial nas mais variadas áreas de conhecimento, pode ser utilizada em várias atividades da sociedade, como educação, pesquisa, treinamento, negócios, lazer, etc (BIT, 1999). Segundo Gradecki (1994), a realidade virtual pode ser aplicada a muitas situações, desde capacitar executivos a participarem em conferências televisionadas através do globo, até ajudar estudantes de medicina a praticarem cirurgia.

A realidade virtual pode ser definida como sendo a forma mais avançada de interface do usuário de computador até agora disponível, isto é, uma forma das pessoas visualizarem, manipularem e interagirem com computadores e dados extremamente complexos (Kirner, 2000). De acordo com Machado (2001), o termo realidade virtual é bastante abrangente, e por isto acadêmicos, desenvolvedores de software e principalmente pesquisadores procuram defini-lo baseados em suas próprias experiências, porém, em geral é definido como uma experiência imersiva e interativa baseada em imagens 3D geradas em tempo real por computador.

Além disso, Machado (2001), aponta dois fatores bastante importantes em sistemas de realidade virtual que são imersão e interatividade. A imersão pelo seu poder de prender a atenção do usuário, e a interatividade no que diz respeito à comunicação usuário-sistema. Pinho (1996) em outras palavras, afirma que a interação e o envolvimento são as grandes virtudes da realidade virtual. Segundo Pinho (1996), a realidade virtual permite entrar em lugares onde jamais poder-se-ía na vida real, talvez por estes lugares serem muito pequenos, ou ainda muito distantes. Ela permite que se mova coisas leves e pesadas, permite visitar lugares em períodos diferentes de tempo e com uma rapidez tão grande que sem ela seria impossível.

Porém, existe outra questão importante ligada aos sistemas de realidade virtual que é o fato de ela poder ser imersiva ou não imersiva. Segundo Kirner (2000), do ponto de vista da visualização, a realidade virtual imersiva é baseada no uso de capacetes ou de salas de projeção nas paredes, enquanto a realidade virtual não imersiva baseia-se no uso de monitores.

Como já foi colocado anteriormente que um dos fatores mais importantes em um sistema de realidade virtual é a imersão, subentende-se que a realidade virtual não imersiva não poderia ser considerada um sistema de realidade virtual, porém, de acordo com Kirner (2000) os dispositivos baseados nos outros sentidos acabam dando algum grau de imersão à realidade virtual com o uso de monitores, mantendo sua característica e importância. Ele afirma ainda que embora a realidade virtual com o uso de capacetes tenha evoluído e seja considerada típica, a realidade virtual com o uso de monitor apresenta pontos positivos como: utilizar todas as vantagens da evolução dos computadores, evitar as limitações técnicas e problemas decorrentes do uso de capacetes e facilidade de uso. Já segundo Pasqualotti (2000) a principal vantagem dos sistemas de realidade virtual não imersivos é seu baixo custo, que permite assim, atingir uma quantidade maior de interessados em desenvolver este tipo de sistema.

## 2.2 AMBIENTES VIRTUAIS

É nos mundos virtuais que se experimenta novas realidades (Gradecki, 1994). Com eles é possível representar várias formas, como prédios, automóveis, ou em outros casos, representar um mundo sem nenhuma referência física, isto é, constituindo um modelo abstrato (Kirner, 1999). Através dos mundos virtuais é possível representar tanto coisas simples que comumente são feitas no dia a dia das pessoas, quanto experimentar ou interagir com coisas que talvez não seriam possíveis no mundo real. Por isso, não se deve esquecer a energia e criatividade que pode ser trazida à vida num mundo virtual (Gradecki, 1994).

O ambiente virtual, também conhecido como mundo virtual, aborda questões como construção do modelo tridimensional, características dinâmicas do ambiente, características da iluminação e detecção de colisão (Kirner, 1999). Além disso, segundo Kirner (1999), os ambientes virtuais também devem conter objetos virtuais, que de um modo geral são classificados como estáticos ou dinâmicos, dependendo da capacidade de movimentação de cada um, podendo ainda ter outros atributos associados como: geometria, cores, texturas,

características dinâmicas e atributos acústicos. Por outro lado, esses objetos também podem ter restrições físicas associadas, como limite de translação ou de rotação.

Assim, com as características que foram expostas, um mundo virtual pode ser definido como sendo um cenário dinâmico em três dimensões modelado computacionalmente através de técnicas de computação gráfica e usado para representar a parte visual de um sistema de realidade virtual (Pinho, 1999). Uma definição mais direta, segundo Pinho (1999), é a de que um ambiente virtual é um cenário onde os usuários de um sistema de realidade virtual podem interagir. Como meio de interação com as aplicações, muitos pesquisadores e desenvolvedores vêm utilizando a internet, isto devido a sua explosão de popularidade através da *World Wide Web* aliada às tecnologias que permitem maior dinamismo e flexibilidade de interação com a *web*, como por exemplo a linguagem Java (Raposo, 2000).

De acordo com Raposo (2000), as grandes vantagens em se desenvolver aplicações disponibilizadas via *web*, estão associadas à fácil acessibilidade: as aplicações ficam disponíveis a uma ampla gama de usuários da *web*, e podem ser acessadas de praticamente qualquer lugar, sem contar que ainda existe a independência de plataforma das aplicações *web*. No entanto, ainda existem aplicações que impõe algumas limitações no que diz respeito à interação. É o chamado "isolamento" do usuário, com as aplicações que não permitem vários usuários interagirem entre si no mesmo mundo virtual. Porém, estas limitações já estão sendo vencidas através da evolução dos mundos virtuais, com o surgimento dos ambientes virtuais distribuídos. Pesquisas relacionadas aos ambientes virtuais distribuídos já existem há duas décadas, mas só o advento da internet os tornaram disponíveis para o grande público.

## 2.3 A EVOLUÇÃO DOS AMBIENTES VIRTUAIS

A *World Wide Web* e sua estrutura hipermídia oferece a ilusão de uma grande base de informações. No entanto, ainda existem barreiras significativas à colaboração efetiva entre os usuários. Apesar da informação ser um recurso compartilhado, os *web browsers* ainda são ferramentas para um único usuário, mantendo os usuários separados uns dos outros, já que oferecem pouco suporte para que um grupo de usuários trabalhe de maneira colaborativa sobre a informação compartilhada (Raposo, 2000).

Porém, com a evolução dos ambientes virtuais, ou seja, com o surgimento dos ambientes virtuais distribuídos, viu-se a possibilidade da participação de diversos usuários, conectados através de um único sistema distribuído.

Segundo Singhal (1999), os primeiros ambientes virtuais distribuídos surgiram no início dos anos 80, através do Departamento de Defesa dos Estados Unidos, o qual, além de ser um dos pioneiros no desenvolvimento de sistemas relacionados a ambientes virtuais distribuídos, como por exemplo o SIMNET, também é considerado o primeiro no desenvolvimento em grande escala. O SIMNET (*simulator networking*) é um ambiente virtual distribuído de uso militar, desenvolvido para o exército dos Estados Unidos. O objetivo do projeto era desenvolver um ambiente virtual distribuído de baixo custo para treinamento de pequenas unidades simulando lutas com um time. Os dois principais desafios do projeto foram: desenvolver um sistema de alta qualidade através de simuladores de baixo custo e como colocá-los juntos em rede de forma a criar uma batalha virtual consistente. Atualmente, o Departamento de Defesa dos Estados Unidos é considerado um dos maiores desenvolvedores de ambientes virtuais distribuídos para uso de sistemas de simulação.

Além disso, nos dias atuais, as pesquisas na área dos ambientes virtuais distribuídos vêm sendo conduzidas por duas comunidades separadas: a comunidade da internet e a comunidade militar. Estas duas comunidades tomaram caminhos diferentes no desenvolvimento de ambientes virtuais distribuídos, discordando sobre quais aspectos são mais importantes hoje em dia. A comunidade da internet, liderada por empresas comerciais e universidades, enfoca a disponibilidade para a grande massa de usuários. A comunidade militar, por sua vez, segue a área de simulações realistas, imersivas e que suportam milhares de usuários simultaneamente, enfocando o treinamento para situações de combate (Raposo, 2000). Singhal (1999) também destaca o fato dos ambientes virtuais distribuídos começarem a ser utilizados em áreas como: educação, treinamento, engenharia, projetos, comércio e entretenimento, no entanto, segundo Raposo (2000), limitações de banda de passagem e latência da internet ainda são empecilhos ao desenvolvimento de ambientes virtuais distribuídos mais efetivos.

Outros possíveis cenários de aplicações dos ambientes virtuais distribuídos são citados a seguir de acordo com Waters[2], apud Raposo (2000):

- arquitetos em locais diferentes podem “visitar” um prédio virtual que eles estão projetando;
- turistas podem “visitar” o local de destino antes de comprar as passagens;
- simulações militares para situações de combate;

[2] WATERS, R. C; BARRUS, J. The rise of shared virtual environments. **IEEE spectrum**, Los Alamitos, v. 34, n. 3, mar. 1997.

- lojas virtuais simulando as compras do mundo real: carrinho de compras, produtos nas prateleiras e possível interação com vendedores e outros clientes da loja;
- educação: para aprender uma nova língua, pode-se criar por exemplo uma Paris virtual, onde os alunos e professores assumiriam papéis nessa sociedade virtual. Aprender-se-ía francês e algo sobre a vida na França.

## 2.4 DEFINIÇÕES E TIPOS DE AMBIENTES VIRTUAIS DISTRIBUÍDOS

Segundo Waters, apud Raposo (2000), um ambiente virtual distribuído também pode ser definido de acordo com as seguintes características:

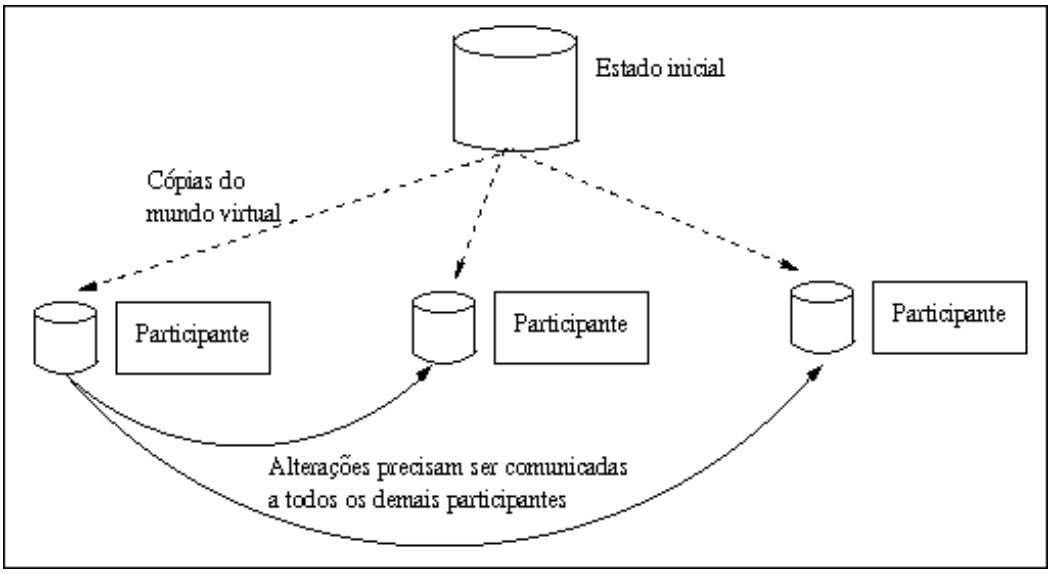
- permite que um grupo de usuários separados geograficamente interaja em tempo real;
- permite número elevado de usuários;
- é tridimensional para os olhos e ouvidos, ou seja, movimentos no ambiente mudam a perspectiva visual e auditiva do usuário;
- usuários são representados como objetos do ambiente, onde a representação do usuário é o avatar;
- muda continuamente em todos os aspectos, ou seja, novos usuários entrando, saindo, se movendo, mudando o estado dos objetos, etc. Isto faz com que seja passado o estado atual do ambiente para os usuários que entram e não uma representação inicial armazenada em algum lugar da *web*;
- usuários, além de interagirem entre si, podem interagir com simulações computacionais, portanto, podem ir além da imitação da realidade, permitindo coisas que não existem no mundo real;
- comunicação verbal é a base da interação entre usuários;
- executado em *hardware* "barato" e nas redes padrões, atingindo assim um grande público.

Tendo a definição de um ambiente virtual distribuído, pode-se agora verificar os tipos de ambiente que podem ser utilizados no mesmo. Segundo Macedonia (1997), os tipos de ambiente são definidos de acordo com o local onde são armazenados os dados relevantes ao estado do ambiente e de seus objetos. Os tipos de ambiente são: ambiente homogêneo replicado; ambiente com dados compartilhados centralizados; ambiente com dados

compartilhados distribuídos e atualização ponto a ponto; e ambiente com dados compartilhados distribuídos cliente-servidor.

A seguir tem-se uma definição de cada tipo de ambiente segundo Macedonia (1997). No ambiente homogêneo replicado, o estado de cada participante é inicializado a partir de uma base de dados homogênea contendo todas as informações do mundo virtual. Cada participante passa a ter cópia do estado inicial e só precisa ser comunicado das mudanças de estado, como por exemplo a locomoção de objetos. A grande vantagem deste tipo de ambiente é que as mensagens transmitidas são pequenas. No entanto, este tipo de ambiente é relativamente pouco flexível e, com o tempo, tem uma forte tendência ao aparecimento de inconsistências entre os participantes devido à perda de mensagens. Este tipo de ambiente pode ser visualizado na fig. 2.1.

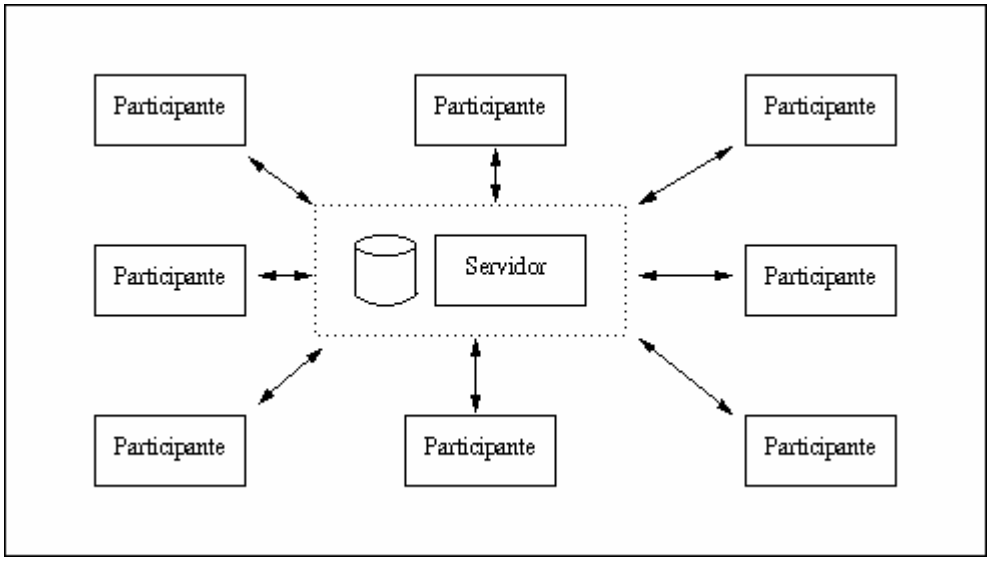
Figura 2.1 – Ambiente homogêneo replicado




Fonte: Raposo, 2000

Já no ambiente com dados compartilhados centralizados existe uma base de dados localizada em um servidor central, compartilhada por todos os usuários. Todas as alterações de estado realizadas pelos usuários devem ser comunicadas apenas ao servidor, que redistribui a mensagem a todos os participantes. Como a base de dados é compartilhada, apenas um usuário pode alterá-la de cada vez. A vantagem com relação ao ambiente homogêneo replicado é que estes estão menos sujeitos a inconsistências devido à base de dados ser centralizada. No entanto, o servidor é muito sobrecarregado e tende a se tornar o gargalo do sistema. Além disso, o tráfego de mensagens é muito grande, pois as mensagens são enviadas primeiramente ao servidor e então distribuídas para os participantes, como é mostrado na fig. 2.2.

Figura 2.2 – Ambiente com dados compartilhados centralizados




Fonte: Raposo, 2000

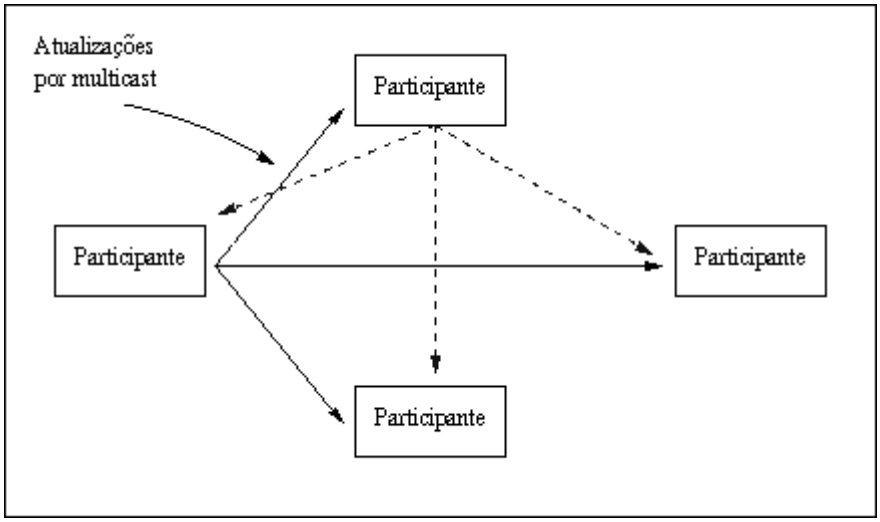
No ambiente com dados compartilhados distribuídos utilizando atualização ponto a ponto, os dados são replicados parcial ou totalmente em cada participante deste tipo de ambiente. Não existe um servidor central e as alterações são comunicadas aos participantes via *multicast*[3]. A vantagem é não ter o servidor como gargalo do sistema, entretanto, existem problemas de escalabilidade[4] devido aos custos para a manutenção da confiabilidade e consistência de dados, além de a comunicação ponto a ponto permitir os mesmos privilégios a todos os participantes sobrecarregando os mesmos com mensagens. Este tipo de ambiente é mostrado na fig. 2.3.

O ambiente com dados compartilhados distribuídos cliente-servidor é uma variante do modelo cliente-servidor, onde os dados estão divididos entre os participantes e a comunicação é mediada por um servidor central. Este tipo de ambiente supera os problemas característicos da comunicação ponto a ponto, porém, a presença de um servidor pode servir como gargalo do sistema, especialmente em ambientes com número elevado de usuários. A fig. 2.4 ilustra este tipo de ambiente.

[3] Através do transporte *multicast* é possível enviar dados a vários destinos fazendo apenas uma requisição de transporte (Gaspary, 1996).

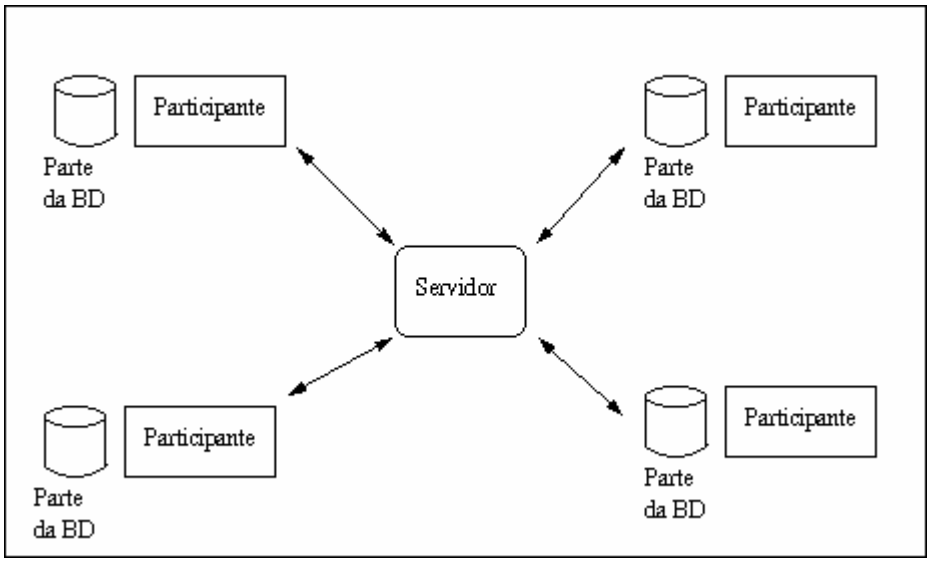
[4] Capacidade de expandir o número de usuários (Aspetus, 2001).

Figura 2.3 – Ambiente com dados compartilhados distribuídos com atualização ponto a ponto




Fonte: Raposo, 2000

Figura 2.4 – Ambiente com dados compartilhados distribuídos cliente-servidor




Fonte: Raposo, 2000

Para finalizar este capítulo, é interessante ressaltar a questão da percepção do usuários. Segundo Benford (1995), os participantes de ambientes virtuais distribuídos devem ser diretamente visíveis a si mesmos e aos demais participantes. A percepção dos usuários deve englobar os seguintes aspectos:

- presença: o objetivo principal em se representar o usuário no ambiente é indicar sua presença no ambiente, de modo que todos os usuários possam saber quem está presente;
- localização: é importante saber a posição dos usuários no ambiente. A orientação do usuário também é importante, principalmente no que diz respeito à interação com outros usuários, indicando, por exemplo, se ele está de frente ou de costas;

- identidade: é necessário diferenciar os usuários de outros objetos do ambiente e diferenciar os usuários entre si, de modo que se possa reconhecer um usuário quando ele for visto novamente;
- atividade: a representação do usuário deve permitir um reconhecimento da atividade em andamento, o que é ainda mais importante em atividades colaborativas;
- disponibilidade: é preciso indicar se o usuário está disponível para a interação, pois há situações em que a representação do usuário está no ambiente, mas ele pode não responder a uma interação. Isto evita a frustração de outros usuários que tentarem interagir com ele;
- gestos e expressões faciais: importantes na conversação e, portanto, devem ser representados no ambiente virtual;
- história de atividades: indicação de quem esteve presente no ambiente e o que eles fizeram;
- manipulação da visão dos outros participantes: o usuário deve ter alguma influência na determinação de como os outros usuários aparecem para ele. Em ambientes heterogêneos isso é muito importante, pois usuários com recursos mais limitados podem optar por representações mais simples dos participantes;
- representação em várias mídias: não apenas a imagem, mas também o áudio e texto;
- presença em vários locais: ambientes virtuais podem extrapolar a realidade e permitir a uma pessoa estar em mais de um lugar ao mesmo tempo;
- verdade: neste aspecto, é levantada a questão da representação ser coerente com a pessoa do mundo real ou ser criada de acordo com o desejo do usuário;
- eficiência: como priorizar os aspectos acima mencionados de acordo com necessidades específicas de cada aplicação e recursos computacionais limitados.

O próximo capítulo apresenta uma visão geral da linguagem VRML, a qual será utilizada para a construção do ambiente virtual.

# 3 A LINGUAGEM VRML

Este capítulo aborda a linguagem VRML, apresentando conceitos, definições, história e situação atual. Também são apresentadas algumas aplicações disponíveis criadas com a linguagem VRML, bem como um breve tutorial para criação de mundos e objetos virtuais utilizando esta linguagem.

## 3.1 CONCEITOS GERAIS

A linguagem VRML surgiu da idéia de se levar a realidade virtual para a internet. Abreviação de *Virtual Reality Modeling Language*, ou Linguagem para Modelagem em Realidade Virtual, ela é uma linguagem independente de plataforma que permite a criação de ambientes virtuais por onde se pode passear, visualizar objetos por diferentes ângulos e até interagir com eles (Ipolito, 1997). Segundo Pollo (1997), a VRML é uma linguagem padrão utilizada para descrever objetos 3D, combinando os mesmos em cenas ou mundos virtuais. Com ela, pode-se criar simulações interativas que incorporam animação, física do movimento e participação multiusuária em tempo real. Assim como a maioria dos autores, Pollo (1997) também destaca o fato da linguagem VRML ter sido projetada para a *World Wide Web*, ou seja, a interação dentro dos ambientes tridimensionais é realizada através da internet. Com isso, a VRML permite a seus usuários em qualquer parte do mundo criar e compartilhar seus objetos e espaços 3D (Kay, 1997). Além de possibilitar a criação destes mundos virtuais em três dimensões na internet, pode-se também construir salas virtuais, edifícios, cidades, montanhas e planetas. Pode-se ainda preencher estes mundos virtuais com móveis, carros, pessoas, naves espaciais, ou qualquer coisa que se possa sonhar, ou seja, a imaginação é o único limite (Ames, 1997).

A linguagem VRML contribuiu para o desenvolvimento da computação gráfica em 3D, cujo objetivo principal sempre foi o de conseguir apresentar cenários que realmente parecessem reais em uma tela de computador, embora novas características estejam constantemente sendo adicionadas à especificação e a evolução esteja longe de terminar. A partir da década de 80, com a explosão da Internet norte-americana, um novo objetivo passou a direcionar os trabalhos em computação gráfica: o desenvolvimento não apenas de um ambiente simulado visualmente convincente, mas de um *cyberspace*, uma verdadeira experiência virtual interativa entre usuários de uma rede. A primeira grande revolução nessa busca foi a criação da *World Wide Web*, por Tim Berners-Lee, em 1989. Embora fosse

essencialmente uma interface gráfica para a internet e envolvesse apenas gráficos 2D, a *web* foi o primeiro passo na direção da criação da linguagem VRML (Pollo, 1997).

A criação da *web* foi mais um dos esforços da comunidade de computação visando melhorar a interface entre os usuários e as tarefas executadas pelos computadores. Até a introdução dos URL (*Universal Resource Locators*), utilizados para resolver a maior parte dos endereços da internet, mesmo usuários avançados de computadores tinham grande dificuldade para lembrar "onde estava o quê", pois é necessário, por exemplo, lembrar que http://www.inf.furb.br/ é o endereço da homepage dos cursos de informática da FURB. Nem sempre os endereços são tão reveladores, e isso levou a uma constatação básica: se a informação não é representada sensorialmente, de maneira semelhante ao que se está acostumado no mundo real, ela não faz muito sentido, e fica difícil fazer associações. A partir desta constatação, uma grande variedade de tecnologias, que vieram a ser coletivamente conhecidas como realidade virtual, deram início a uma mudança radical na natureza das chamadas "interfaces com o usuário", tornando o usuário o centro do processo de *design* destas interfaces. A idéia era a imersão do usuário em um espaço tridimensional interativo que representasse o ambiente computacional e no qual as informações pudessem ser absorvidas através de todos os sentidos (Pollo, 1997).

Decorrente de pesquisas nesta área, o segundo passo foi a introdução, em 1992, do *toolkit* gráfico Inventor, da Silicon Graphics. Embora na época nem se cogitasse a utilização do novo produto como base para o desenvolvimento de gráficos 3D para a Internet, o formato de arquivo do Inventor formaria a base sobre a qual toda a especificação de VRML seria fundamentada. Os conceitos chaves no Inventor eram a estrutura da cena e a descrição de objetos, o que permitia aos programadores gráficos desenvolver aplicações 3D interativas de maneira rápida e eficiente, duas qualidades imprescindíveis em aplicações internet (Pollo, 1997).

Mas foi o terceiro acontecimento que realmente impulsionou a criação da primeira especificação de VRML. Depois de meses trabalhando em uma interface de realidade virtual para a *web*, Mark Pesce e Tony Parisi foram convidados por Berners-Lee para apresentar suas idéias durante a primeira conferência anual sobre *World Wide Web* em Genebra, Suíça, numa sessão de discussão sobre realidade virtual. Um ano e meio depois do lançamento do Inventor pela Silicon Graphics, uma entusiasmada platéia decidia iniciar o desenvolvimento de uma linguagem que combinasse descrição de cenas tridimensionais com os *hyperlinks* da internet, algo análogo ao HTML, mas para realidade virtual. A sigla VRML, que inicialmente

significava *Virtual Reality Markup Language*, era cogitada como designativo para a linguagem pela primeira vez. Mais tarde, o termo *Markup* seria substituído por *Modeling*, procurando coerência com os objetivos da linguagem (Pollo, 1997).

Imediatamente após o término da conferência, o grupo que havia participado da discussão, iniciou os trabalhos na primeira especificação da VRML. Em um mês, havia se estabelecido uma lista de discussão por *e-mail* reunindo pessoas interessadas em desenvolver a especificação. Durante sua primeira semana de existência, a lista cresceu assustadoramente, chegando a conter mais de mil membros. Ao mesmo tempo, na lista de discussão, Mark Pesce havia requisitado que os membros concluíssem um esboço da primeira especificação em apenas cinco meses, para apresentá-lo em outubro na segunda conferência sobre *World Wide Web*. Era pouco tempo para se construir uma especificação inteiramente nova para uma linguagem, e todos concordaram que seria mais fácil adaptar uma linguagem de modelagem já existente (Pollo, 1997).

Em algumas semanas, Gravin Bell, um dos engenheiros do *Inventor* na Silicon Graphics, apresentaria a proposta da empresa: um subconjunto modificado do formato *Open Inventor 3D Metafile*, com a adição de algumas funcionalidades para a utilização em rede. A empresa ainda daria uma contribuição decisiva para que o seu formato de arquivo se tornasse a base da nova especificação, tornando-o não proprietário e de domínio público. Mesmo com várias outras propostas interessantes sendo submetidas à avaliação na lista de discussão, a da Silicon Graphics foi a mais votada, e o *Open Inventor* tornava-se, de fato, a base para a especificação VRML 1.0, apresentada em outubro de 1994 (Pollo, 1997).

A primeira versão de VRML (VRML 1.0) permite a criação de mundos virtuais com comportamento interativo muito limitado. Os arquivos descrevem ambientes constituídos de objetos que podem conter ligações (*hyperlinks*) com outros mundos, mas que não respondem a ações do usuário em tempo real, como era de se esperar em um sistema realmente interativo. Um dos grandes problemas da versão 1.0 é que as cenas são estáticas, elas não mudam com o passar do tempo. A proposta de VRML 1.0 era propositalmente simples, pois deveria servir apenas como um impulso original para o processo de aprimoramento da linguagem e para chamar a atenção da comunidade internet para a nova tecnologia (Pollo, 1997).

Os objetivos da versão 2.0 já são bem mais ambiciosos. A nova especificação traz inovações que prometem revolucionar a maneira como as pessoas acessam as informações na *web*. Dentre elas, a adição de comportamento ao cenário 3D é, com certeza, a mais importante. Assim como Java está sendo usada para produzir páginas HTML mais dinâmicas

e interativas, VRML foi estendida para permitir que o ambiente mude e evolua, tornando-se sensível às ações ou à simples presença de seus visitantes virtuais. Em VRML 2.0, os objetos podem se mover pelo mundo virtual e responder a eventos temporais ou disparados pelo usuário, e é possível adicionar som espacial e filmes de vídeo à cena, tornando a experiência do observador muito mais realista. Os novos recursos de interação elevaram o padrão a um nível um pouco mais próximo do *cyberspace* ideal (Pollo, 1997).

De acordo com Pollo (1997), o que há de comum entre as duas versões é que, em essência, um arquivo VRML é simplesmente uma coleção de objetos, arranjados (organizados) em uma certa ordem. Como na vida real, um objeto geralmente tem uma forma, uma superfície com certas propriedades (cor, brilho, suavidade, etc.) e uma posição no espaço tridimensional. Outros objetos VRML incluem sons, luzes e pontos de vista, e também têm uma posição no espaço 3D. O princípio básico de funcionamento de VRML é deixar de lado a complexidade matemática exigida pelas operações de renderização dos objetos 3D, provendo única e exclusivamente uma base consistente, apesar de simples, da qual todos possam se beneficiar, desde desenvolvedores de *browsers* até usuários finais.

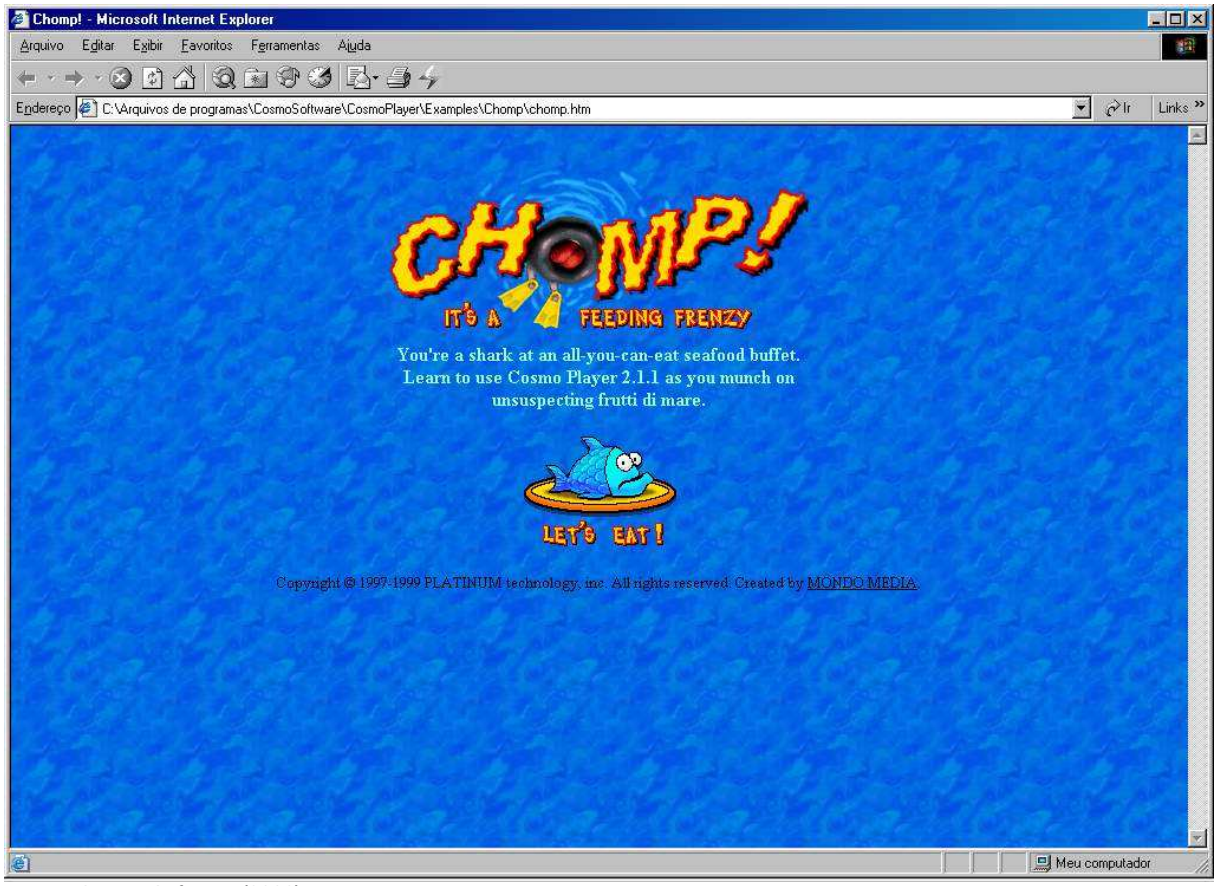
## 3.2 APLICAÇÕES E ATUALIDADE

Instalando o Cosmo Player, *plugin* que possibilita explorar mundos 3D, pode-se verificar uma aplicação em VRML. Esta aplicação, a qual recebeu o nome de Chomp, é instalada junto ao diretório do Cosmo Player, mais especificamente no diretório *examples*. Dentro deste encontra-se o diretório específico da aplicação, onde são encontrados todos os arquivos HTML, arquivos VRML e arquivos de som necessários para a execução da mesma.

A aplicação consiste em um jogo simples, onde o usuário navega em um ambiente virtual, mais especificamente um oceano virtual, controlando um tubarão. O objetivo do jogo, resumidamente, é navegar entre os labirintos construídos neste oceano e comer tudo o que aparecer na frente do tubarão, como por exemplo, patos, peixes e até pessoas brincando no mar. Cada pato, pessoa ou peixe que o tubarão abocanhar faz com que o usuário vá marcando pontos que são acumulados até que se chegue à pontuação máxima, e assim finalizando o jogo.

Através deste jogo, o usuário pode familiarizar-se com as opções de navegação que o Cosmo Player permite, além de conhecer um pouco sobre o que é possível realizar com a linguagem VRML. Para iniciar o jogo é necessário abrir o arquivo HTML chamado *chomp.htm*, o qual será aberto no *browser* e exibido conforme a fig. 3.1.

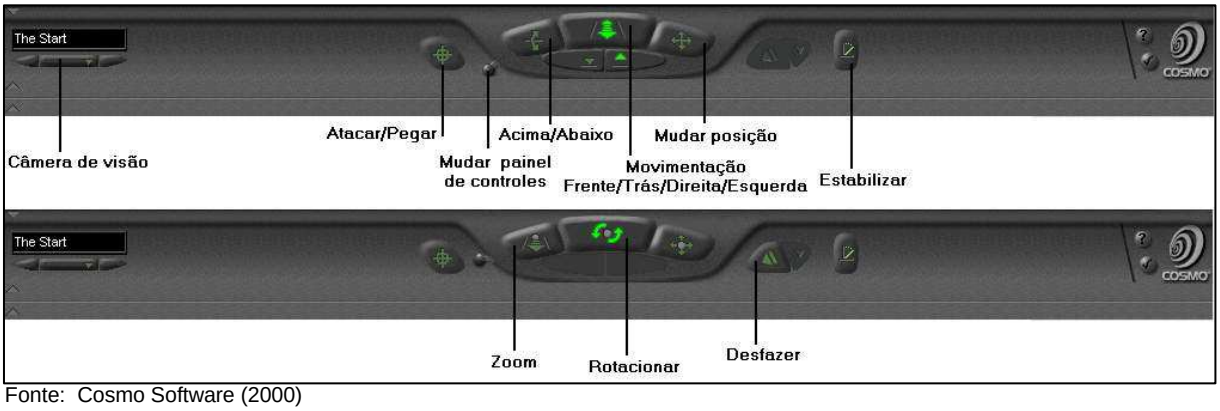
Figura 3.1 – Aplicação em VRML instalada com o Cosmo Player




Fonte: Cosmo Software (2000)

Após a exibição da tela inicial, deve-se clicar sobre a figura do peixe na bandeja, para que então sejam carregados os arquivos em VRML, e então iniciar o jogo. No topo da página é exibido o mundo VRML com as opções de navegação do Cosmo Player, as quais podem ser utilizadas selecionando uma das opções exibidas na fig. 3.2 e então arrastando o mouse sobre o mundo virtual mantendo o botão esquerdo pressionado.

Figura 3.2 – Opções de navegação permitidas no jogo Chomp

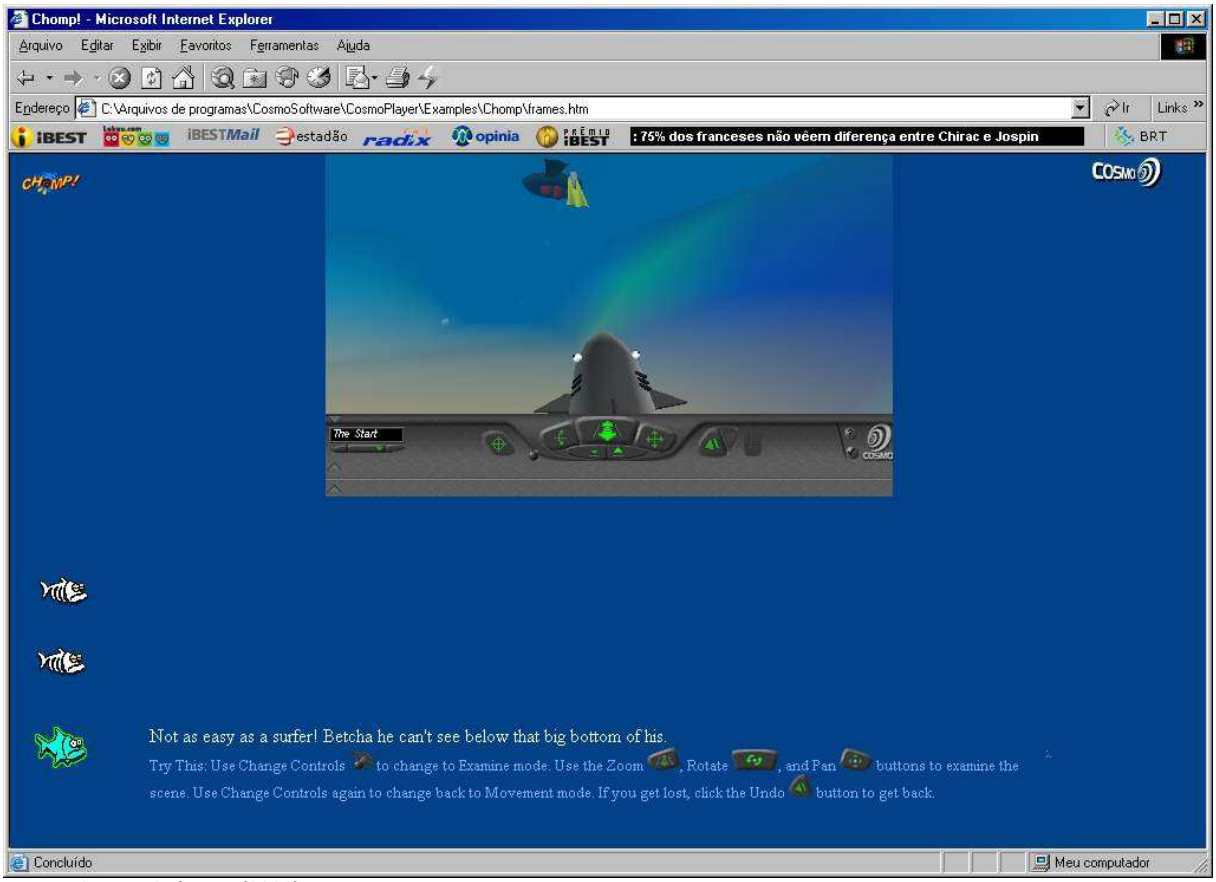



Fonte: Cosmo Software (2000)

Para movimentar-se no mundo virtual também é possível utilizar as setas do teclado, além de existirem outras teclas de atalho e teclas com funções extras, como por exemplo a

tecla *shift*, que aumenta a velocidade de movimentação do tubarão. Durante o jogo também são exibidas mensagens de ajuda para o usuário, um pequeno tutorial, que explica que opções utilizar em determinados momentos do jogo. Além disso, também é exibida a pontuação no lado esquerdo da tela, identificando quantas vítimas o tubarão já fez através da espinha de um peixe. Na fig. 3.3 pode-se ver a tela do jogo durante sua execução, podendo ainda identificar a pontuação, um banhista boiando no mar e os comentários do tutorial no rodapé da página.

Figura 3.3 – Tela do jogo Chomp durante sua execução

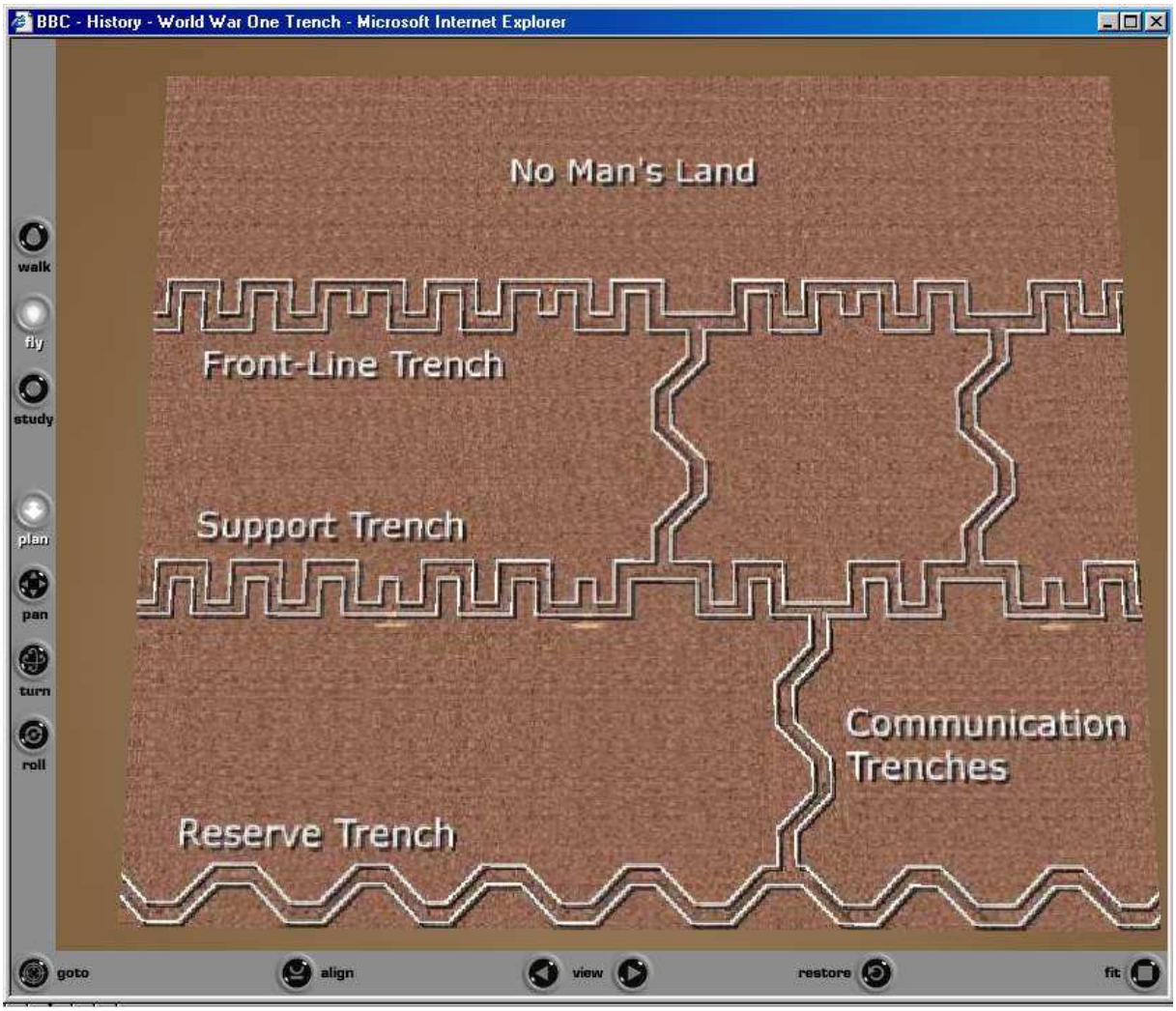



Fonte: Cosmo Software (2000)

Ao final, após ter devorado todos os patos, os peixes, banhistas e surfistas, o tubarão tem uma indigestão e explode, e assim, o jogo chega ao fim com a exibição de uma página congratulando o usuário vencedor.

Outra aplicação utilizando a linguagem VRML pode ser encontrada em BBCi (2002), onde o mundo VRML representa a reconstrução 3D de uma das trincheiras da primeira guerra mundial. Neste mundo virtual pode-se ouvir alguns tiros ao fundo enquanto se anda entre os labirintos formados nas trincheiras, locais protegidos com arame farpado, armas e depósitos de caixas de munição. A fig. 3.4 mostra uma vista aérea das trincheiras e respectivos locais que se pode percorrer e conhecer.

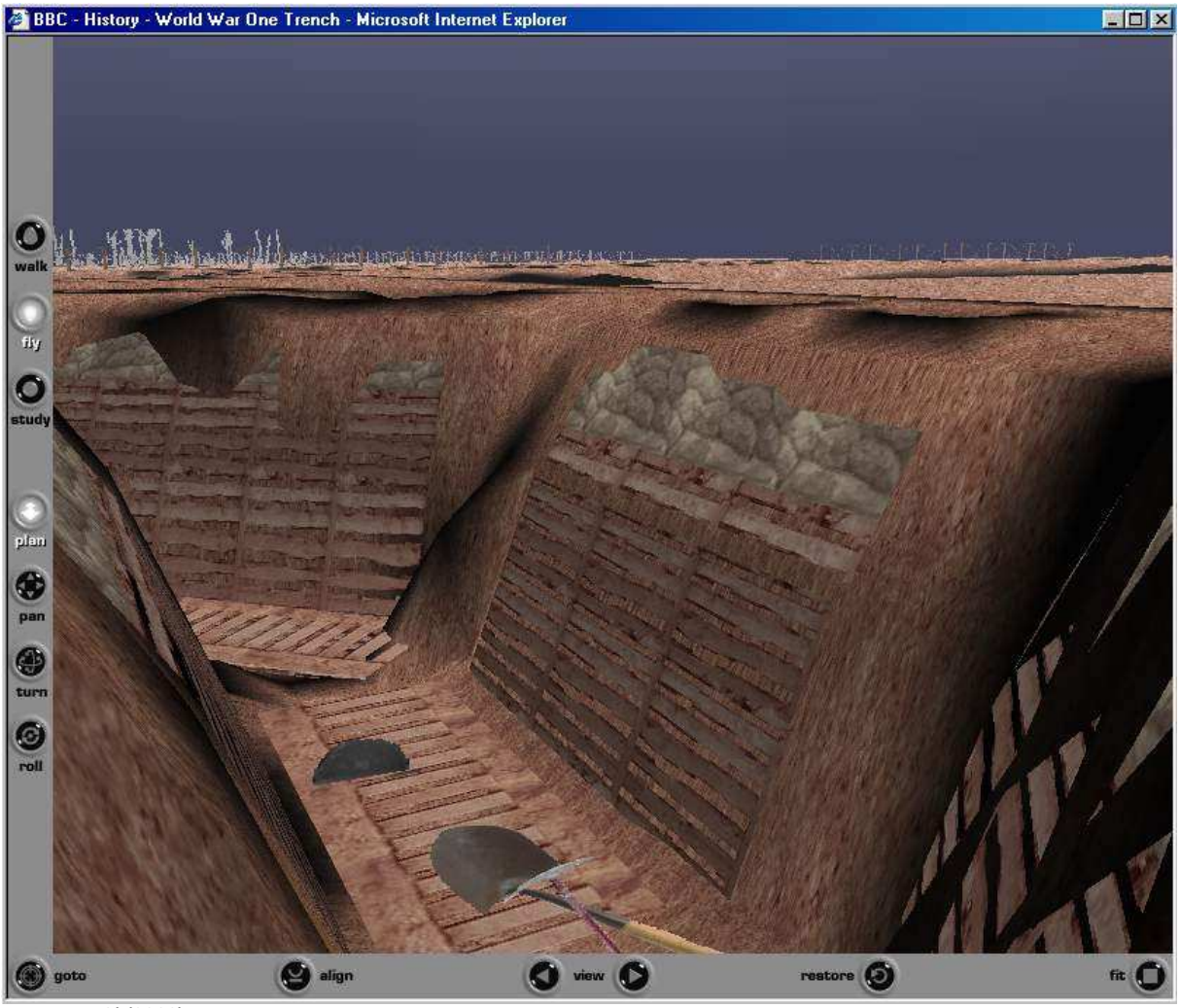
Figura 3.4 – Vista aérea de uma trincheira da primeira guerra mundial




Fonte: BBCi (2002)

Para executar esta aplicação utilizou-se o *plugin* Cortona da Parallel Graphics, o qual apresenta as opções para navegação no mundo virtual em uma barra à esquerda e abaixo do mundo VRML. Como no *plugin* Cosmo Player, pode-se escolher uma dentre várias câmeras de visão disponíveis para o mundo VRML dependendo de como foi criado. Esta aplicação possui várias câmeras de visão, que mostram diferentes pontos e ângulos deste mundo virtual, podendo-se citar a câmera *flyover*, onde a mesma vai se deslocando entre as trincheiras permitindo conhecer diferentes pontos deste mundo virtual, até chegar ao seu destino final. A fig. 3.5 mostra um dos locais da trincheira por onde a câmera *flyover* se desloca.

Figura 3.5 – Trincheira da primeira guerra mundial

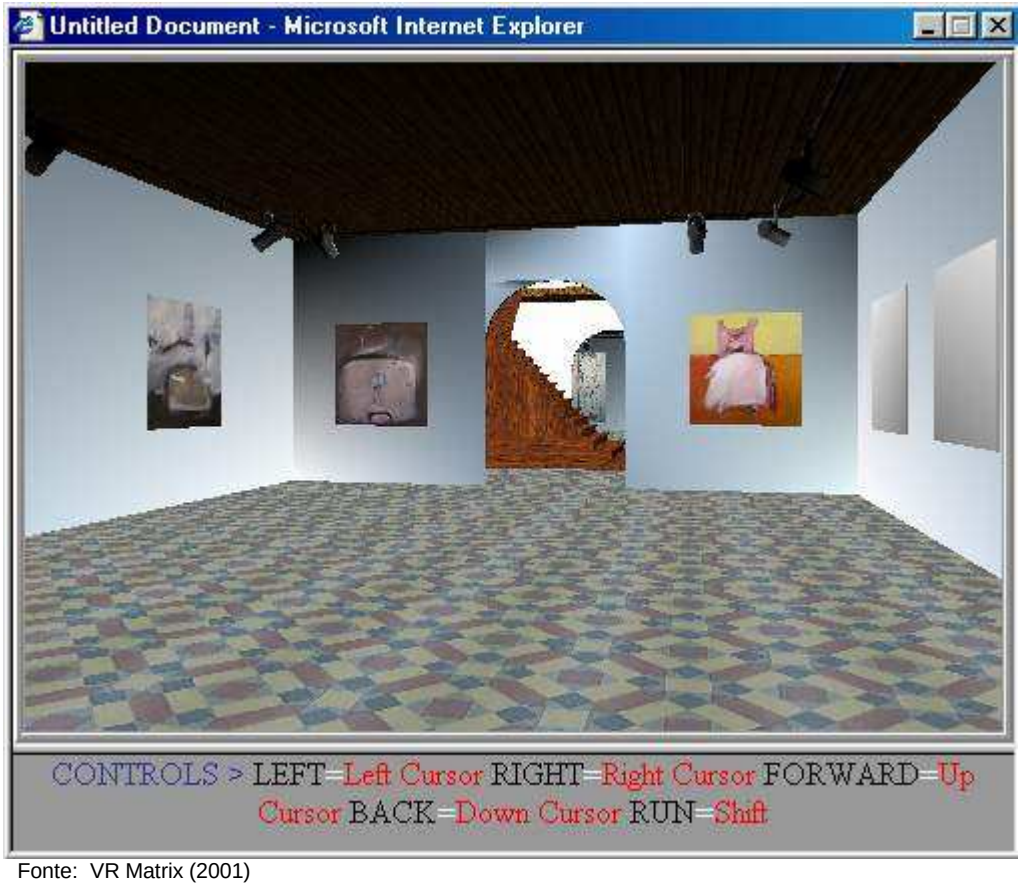



Fonte: BBCi (2002)

Ao contrário da aplicação Chomp, comentada anteriormente, a aplicação que representa a primeira guerra mundial não possui um objetivo específico, porém, representa o local onde aconteceram fatos da história mundial, permitindo conhecer o mesmo através de um passeio neste mundo virtual.

Por último, pode-se citar ainda a aplicação encontrada em VR Matrix (2001), que representa uma galeria de arte virtual. Nesta galeria composta de dois andares, pode-se caminhar, subir as escadas e apreciar as obras de arte expostas na mesma, utilizando as setas do teclado ou arrastando o mouse na direção que se deseja caminhar. A fig. 3.6 mostra uma sala deste mundo virtual, bem como as opções de movimentação.

Figura 3.6 – Mundo VRML representando uma galeria de arte




Fonte: VR Matrix (2001)

Com isso, através das aplicações expostas anteriormente pode-se conhecer um pouco sobre a capacidade da linguagem VRML. Com relação a situação atual da linguagem VRML, verificou-se que, segundo Web3D (2000b), o atual padrão de especificação da linguagem é o VRML97, com o qual será apresentado um tutorial básico a seguir.

## 3.3 TUTORIAL BÁSICO DE VRML

Em construções no mundo real, casas e prédios são construídos utilizando projetos que especificam esta construção. O arquivo VRML serve como o projeto de construção do mundo virtual que se está criando (Ames, 1997). De acordo com Nadeau (1999), a VRML é uma linguagem textual, que possibilita a construção rápida de mundos virtuais incorporando objetos em três dimensões, luzes, animações e efeitos sonoros. Cada mundo virtual é descrito por um ou mais mundos VRML nomeados como arquivos de extensão .wrl, que representa a palavra *world*, mundo na língua inglesa. Para visualizar um mundo VRML, tanto na internet quanto localmente no disco rígido de um microcomputador, é necessário a utilização de um *browser* VRML, configurado como um *plugin* para o *browser*. O *browser* VRML lê os arquivos .wrl, interpreta sua sintaxe, constrói o mundo virtual 3D, e então mostra este mundo

em uma página da *web*. O *browser* também possibilita uma interface ao usuário, com a qual pode-se andar pelo mundo virtual, interagir com os seus objetos, iniciar animações e ouvir efeitos sonoros 3D.

Tudo que se precisa para escrever um código VRML é um editor de textos. O arquivo não precisa ser compilado e não é executado por ninguém. Pode-se, por exemplo, criar um cubo e gravá-lo em um arquivo chamado cubo.wrl. O código VRML para este cubo descreverá as características do ambiente, como coordenadas, luz, cores, sombreamento etc. Também pode-se colocar em um mundo objetos que estão localizados remotamente em outros lugares na internet, além de links que levam a outros *homeworlds* ou *homepages* (Ipolito, 1997). A seguir tem-se a especificação de alguns itens que o código VRML 2.0 ou VRML 97 pode conter.

### 3.3.1 CABEÇALHO DE UM ARQUIVO VRML

Segundo Ipolito (1997), ficou definido, para fins de identificação que todo arquivo VRML deve conter um cabeçalho. Ele é obrigatório, deve ser a primeira linha do arquivo, e ainda deve ser definido conforme descrito no quadro 3.1.

Quadro 3.1 – Cabeçalho de um arquivo VRML

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

O cabeçalho descreve para o *browser* que o arquivo é um arquivo VRML, da versão 2.0 e utiliza o padrão internacional UTF-8 como meio de escrita dos caracteres em diversas línguas (Ames, 1997). Segundo Ipolito (1997), o caractere “#” também significa comentário, portanto, após a definição do cabeçalho todas as linhas que iniciarem com este caractere serão ignoradas pelo *browser*.

### 3.3.2 DEFININDO OBJETOS

Para a definição dos objetos em um arquivo VRML, são utilizados os *nodes* que segundo Ipolito (1997) podem ser definidos como sendo um conjunto de especificações que determinam as características individuais de cada objeto dentro do cenário virtual. Segundo Pinho (2001), os objetos tridimensionais em VRML 2 são chamados de *shapes.* O *shape*, que também é um *node*, é um bloco utilizado para definir os objetos do mundo VRML, e em

geral, possui dois atributos, a aparência e a geometria. De acordo com Ames (1997), todos os objetos construídos em VRML utilizam o *node Shape* que tem sua sintaxe descrita no quadro 3.2.

Quadro 3.2 – Sintaxe do *node Shape*

```
Shape { # define um bloco de um objeto
    appearance   NULL
    geometry     NULL
}
```

A aparência do *shape* é especificada utilizando os *nodes Appearance* e *Material* que dentre outros atributos de aparência, define a cor dos objetos utilizando o padrão RGB. Outro *node* que pode ser utilizado para a aparência do objeto é o *texture* que aplica uma textura sobre o objeto (Pinho, 2001). No quadro 3.3 são apresentados os *nodes* que podem ser utilizados na representação da aparência do objeto.

Quadro 3.3 – Sintaxe dos *nodes* de aparência

```
Shape {
    appearance Appearance { # define a aparência do shape
            material Material {
                  diffuseColor 1.0 1.0 0.0   # cor RGB
            } # fim do material
            texture ImageTexture {
                  url [“imagem.jpg”]
            } # fim do texture
    } # fim do appearance
} # fim do shape
```

O outro atributo, a geometria, é especificada pelo *node geometry* que define uma forma geométrica ou 3D do *shape*. A linguagem VRML possui alguns *nodes* para serem utilizados no *shape* e definir formas geométricas simples como um cubo, um cone, um cilindro e uma esfera (Ames, 1997). Estas formas, por sua vez, possuem outros atributos específicos que descrevem as suas propriedades.

Para definir a forma geométrica de um cubo é utilizado o *node Box*, sendo necessário utilizar um atributo para definir a largura, a altura e a profundidade do cubo conforme descrito no quadro 3.4.

Quadro 3.4 – Sintaxe do *node Box*

```
Shape {
    geometry Box { # define a geometria do shape na forma de caixa
          size 2.5 2.5 5.0  # largura, altura, profundidade
    }
} # fim do shape
```

Outra forma geométrica simples que pode ser definida é a esfera. Para esta forma utiliza-se o *node Sphere*, e é necessário especificar o raio da esfera através de seu atributo, como mostrado no quadro 3.5.

Quadro 3.5 – Sintaxe do *node Sphere*

```
Shape {
    geometry Sphere { # define a geometria do shape na forma de esfera
          radius 4.0  # define o raio da esfera
    }
} # fim do shape
```

Para a definição de uma forma geométrica cilíndrica utiliza-se o *node Cylinder*, e assim como nas outras formas apresentadas, especifica-se os valores de seus respectivos atributos. São eles: raio, altura, se o cilindro tem lateral, se tem a parte de cima e de baixo, conforme descrito no quadro 3.6.

Quadro 3.6 – Sintaxe do *node Cylinder*

```
Shape {
    geometry Cylinder { # def. a geometria do shape na forma de cilindro
          radius 2.0   # raio
          heigth 1.0   # altura
          side TRUE    # tem lateral
          top TRUE     # tem a parte de cima
          bottom TRUE  # tem a parte de baixo
    }
} # fim do shape
```

Por último, no quadro 3.7 tem-se a definição da forma geométrica simples de um cone, o qual é representado pelo *node Cone* e os atributos: raio da base, altura, se o cone tem lateral e se tem base.

Quadro 3.7 – Sintaxe do *node Cone*

```
Shape {
    geometry Cone { # define a geometria do shape na forma de cone
          bottomradius 1.0   # raio da base
          heigth 5.0         # altura
          side TRUE          # tem lateral
          bottom TRUE        # tem base
    }
} # fim do shape
```

Todos os objetos apresentados são criados em sua posição original, porém, pode-se alterar a posição destes objetos. Em VRML, para alterar a posição ou orientação de um objeto aplicam-se transformações geométricas, as quais, são definidas através do *node Transform*, no qual é possível especificar a rotação, a posição e a escala que se deseja aplicar aos objetos e quais serão os "objetos filhos" da transformação (Pinho, 2001). No quadro 3.8 é apresentado o código VRML de uma transformação.

Quadro 3.8 – Código apresentando criação, aparência e transformação de um cubo

```
Transform {
    translation 10 0 0    # coordenadas x,y,z em relação à origem
    rotation 0 0 1 0.7853 # eixo de rotação e o ângulo em radianos
    scale 1 1 2           # coordenadas x,y,z
    children [
        Shape {
           geometry Box {
                size 5 5 5  # largura, altura, profundidade
           }
        }
    ]
} # fim do shape
```

Com este capítulo, pode-se ter uma idéia de como funciona a linguagem VRML e como pode ser simples criar um mundo VRML. No próximo capítulo serão apresentados conceitos do protocolo DIS, da linguagem Java e da tecnologia DIS-Java-VRML.

# 4 DIS-JAVA-VRML

Este capítulo apresenta conceitos e funcionamento do protocolo de simulação DIS, da linguagem Java trabalhando em conjunto com a VRML, e tendo como objetivo principal apresentar o conjunto destes tópicos que é a tecnologia DIS-Java-VRML.

## 4.1 DISTRIBUTED INTERACTIVE SIMULATION (DIS)

Em setembro de 1989, o criador do SIMNET, ambiente virtual apresentado na seção 2.3, tinha definido o estado da arte ao criar o protocolo de simulação SIMNET para satisfazer o acordo com o governo dos Estados Unidos. Porém, este protocolo de simulação não tinha sido definido para um propósito geral, ou melhor, ele não tinha sido definido para atender a outros tipos de simulações, que não fossem da própria aplicação SIMNET. Assim, com o sucesso do protocolo SIMNET, mais a falta de um protocolo para aplicações gerais de simulação, surgiu o protocolo DIS. Para facilitar o desenvolvimento deste protocolo criou-se uma conferência bi-anual chamada *Distributed Interactive Simulation Workshop*, resultando na criação da primeira versão do padrão IEEE para o DIS, o padrão IEEE 1278-1993 (IEEE, 1993). O propósito em estabelecer este padrão formalmente foi de permitir a participação de qualquer tipo de usuário, em qualquer equipamento, em um sistema de ambiente virtual distribuído (Singhal, 1999). De acordo com Locke (1995), o DIS compartilha seus objetivos e propósitos com o SIMNET, porém, o DIS é muito mais ambicioso, permitindo maior complexidade e realismo. Um ambiente ou mundo DIS engloba todas as áreas potenciais de atividades militares, terra, atmosfera, acima e abaixo da superfície do oceano e espaço.

### 4.1.1 PROTOCOL DATA UNIT (PDU)

Segundo Singhal (1999), o software de arquitetura de rede do DIS é claramente derivado do SIMNET, pois possui os mesmos componentes básicos, que são:

- arquitetura objeto-evento: modelagem do mundo virtual como sendo uma coleção de objetos que interagem entre si através de uma coleção de eventos, isto é, através de mensagens enviadas para a rede indicando mudanças no estado dos objetos;
- idéia de simulação autônoma: cada objeto do mundo virtual é autônomo, ou melhor, é responsável por transmitir para a rede as suas mensagens representando exatamente o seu estado atual;

- algoritmos chamados *dead reckoning*: utilizados para reduzir o tráfego de mensagens na rede. Quando um objeto se movimenta no ambiente virtual, ao invés de serem enviadas várias mensagens informando as posições que o objeto está ocupando, são informadas apenas, por exemplo, a direção e velocidade do mesmo, para que através dos algoritmos citados se possa calcular a posição que o objeto está ocupando.

Contudo, o centro do software de arquitetura de rede do DIS é o *protocol data unit* (PDU). O padrão DIS (IEEE 1278) define vinte e sete PDU diferentes, onde somente quatro (*entity state*, *fire*, *detonation*, *collision*) são usados pelos objetos para interagirem com o ambiente virtual. A *entity state* PDU, ou ESPDU (PDU que representa o estado do objeto), é enviada em caso de mudança de posição, orientação e velocidade. Além disso, a ESPDU também deve ser enviada como um *heartbeat*, em intervalos de tempos determinados, por exemplo de cinco em cinco segundos, desde que a última ESPDU tenha sido transmitida e recebida (Singhal, 1999). A tabela 4.1 apresenta a estrutura de uma ESPDU.

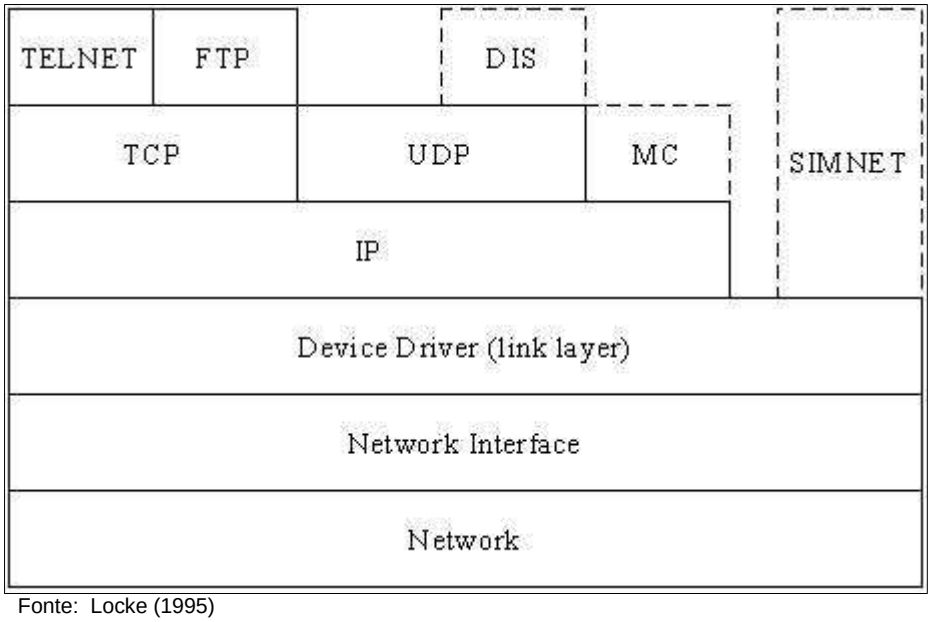
Tabela 4.1 - Exemplo da estrutura de um PDU de estado de objeto

| Nome do Campo | Conteúdo/Assunto | Tamanho Campo (em bytes) |
|---|---|---|
| Cabeçalho do PDU | Versão protocolo, ID do exercício, tipo do PDU, família do protocolo, *timestamp*, tamanho | 12 |
| ID da entidade | *Site*, aplicação, entidade | 6 |
| ID da força | | 1 |
| Número de parâmetros de articulação | | 1 |
| Tipo de entidade | Espécie de entidade, domínio, país, categoria, subcategoria, dados específicos, extras | 8 |
| Tipo entidade alternativa | Espécie de entidade, domínio, país, categoria, subcategoria, dados específicos, extras | 8 |
| Velocidade linear | x, y, z | 12 |
| Localização | x, y, z | 24 |
| Orientação | Psi, theta, phi | 12 |
| Aparência | | 4 |
| Parâmetros de *Dead Reckoning* | Algoritmo, outros parâmetros, aceleração linear e velocidade angular da entidade | 40 |
| Sinalizações da entidade | Grupo de caracteres, sinalizações | 12 |
| Habilidades | | 4 |
| Parâmetros de articulação | Tipo de parâmetro, mudança, ID, valor do parâmetro | N * 16 |

Fonte: Adaptado de Singhal (1999).

Continuando, tem-se a *fire* PDU que é enviada quando uma arma é disparada. Para explosões, batidas ou mortes é enviada a *detonation* PDU, e para objetos que utilizam a detecção de colisão, e que sofrerem algum tipo de colisão é enviada a *collision* PDU (Singhal, 1999). Assim, uma vez que o objeto, sofrendo mudanças em seu estado, envia um PDU colocando-o na rede, é responsabilidade do receptor do mesmo, modificar as tabelas de estado apropriadas e então atualizar a exibição do objeto. No entanto, podem acontecer imprevistos, visto que, em DIS, os PDU são enviados via UDP *broadcast*, o que não é muito confiável, e assim as vezes os PDU podem ser perdidos, sendo provável que as exibições e tabelas de estado diferenciem entre os *hosts* participantes no mesmo ambiente virtual (Singhal, 1999). A fig. 4.1 mostra a localização do protocolo DIS dentre as camadas de rede.

Figura 4.1 – Localização do protocolo DIS




Fonte: Locke (1995)

Por fim, pode-se definir o protocolo DIS como sendo um grupo de padrões, definido pelo Departamento de Defesa dos Estados Unidos e indústrias interessadas, preocupados em determinar uma arquitetura de comunicação comum, definindo “o formato e o conteúdo dos dados comunicados; informações a respeito dos objetos do mundo virtual e sua interação; gerenciamento da simulação; medidas de performance; comunicações de rádio; segurança; fidelidade; controle de exercícios, etc” (Macedônia, 1995).

## 4.2 JAVA E VRML

Segundo Brutzman (1998), a integração da linguagem de modelagem VRML com a linguagem de programação Java fornece um padrão portátil e independente de plataforma para criação de cenas tridimensionais interativas e dinâmicas através da internet. Acrescenta-

se ainda que a linguagem Java adiciona capacidade de programação mais acesso à rede, tornando VRML completamente funcional e portátil. Através desta poderosa combinação mostra-se que a linguagem VRML junto com Java fornece um extenso suporte à construção de ambientes virtuais em grande escala.

A interface entre VRML e Java é efetuada através de *Script nodes*, eventos, nomeando rotinas com *DEF/USE*, e *ROUTEs* ligando vários *nodes* e campos numa cena VRML. Pode-se dizer ainda que a linguagem VRML cria a cena gráfica 3D, os *Script nodes* encapsulam a funcionalidade Java, e os *ROUTEs* fornecem a rede que conecta a computação ao desenho (Brutzman, 1998).

### 4.2.1 SCRIPT NODES

Os *Script nodes* aparecem no arquivo VRML, encapsulando o código Java e nomeando rotinas para fazer a ligação entre as variáveis Java com os valores de campos na cena. As classes Java de interface importam as bibliotecas de classes vrml.* para prover conversões de tipos entre Java e VRML. Além disso, as classes Java utilizadas pelos *Script nodes* devem estender a classe vrml.node.Script para a interface correta com o *browser* VRML (Brutzman, 1998).

De acordo com Web3D (2000b), o *Script node* é utilizado para programar o comportamento em uma cena, além de possuir as seguintes características:

- significa uma mudança ou ação do usuário;
- recebe eventos de outros *nodes*;
- contém módulo de programas que executa alguns cálculos;
- efetua mudanças na cena ao enviar eventos.

No quadro 4.1 tem-se a especificação do *Script node* segundo Web3D (2000b).

Quadro 4.1 – Especificação do *Script node*

```
Script {
    ExposedField MFString url          []
    field        SFBool   directOutput FALSE
    field        SFBool   mustEvaluate FALSE
    # And any number of:
    eventIn      eventType eventName
    field        fieldType fieldName initialValue
    eventOut     eventType eventName
}
```

Fonte: Web3D (2000b)

### 4.2.2 EVENTOS

Os eventos são valores passados e recebidos por diferentes partes de um mundo VRML, são eles que permitem que uma cena VRML seja dinâmica. Para o recebimento de eventos utiliza-se *EventIns* enquanto *EventOuts* é utilizado para o envio dos eventos. Estes eventos enviados devem ter seus tipos (como *integer*, *float*, *color*) ou tipos de *node* (como *Material node*) estritamente compatíveis entre receptor e emissor. Os parâmetros do *Script* são designados como *eventIn*, *eventOut* ou *exposedField*, que correspondem respectivamente a recebimento, envio, ou recebimento/envio. Campos privados ou locais são simplesmente designados como *field* (Brutzman, 1998).

### 4.2.3 NOMEANDO ROTINAS COM DEF/USE

Nomear um *node* e criar várias instâncias de *nodes* é possível através dos mecanismos *DEF* e *USE*. *DEF* é utilizado para associar nomes aos *nodes* enquanto *USE* permite duplicar instâncias de *nodes* para ser referenciado sem uma completa reinstanciação, de forma a aumentar o desempenho da aplicação. Nomes criados via *DEF* também são utilizados para assinalar eventos de *fields* para *fields*, desta forma, todos os *Script nodes* devem ser nomeados utilizando *DEF* (Brutzman, 1998).

### 4.2.4 ROUTE

Instruções *ROUTEs* definem conexões entre *nodes* e *fields*, permitindo eventos passarem do emissor para o receptor. Usualmente, aparecem no fim do arquivo, desde que todos os *nodes* tenham sido nomeados utilizando o mecanismo *DEF*, e são utilizados para todos os eventos passados entre os *Script nodes*. O uso dos *ROUTEs* nem sempre é requerido, desde que os *nodes* da cena VRML possam ser passados por referência como *fields* para o código *Script* encapsulado, opção esta que permite manipulação direta do VRML por Java sem utilizar eventos ou *ROUTEs*.

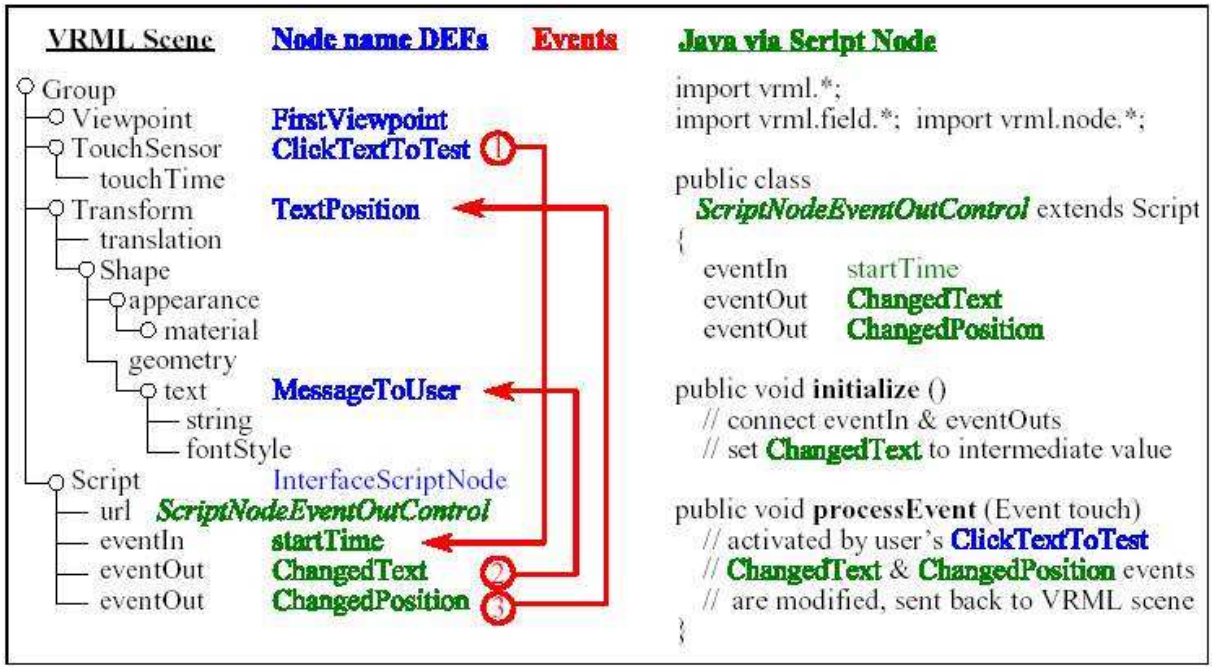
### 4.2.5 EXEMPLOS

Na fig. 4.2 tem-se um exemplo da funcionalidade VRML e Java onde ocorre a seguinte sequência de eventos:

- método *initialize* do Java estabelece ligações, seta o texto na cena 3D para o valor inicial;

- acontece o evento (1) indicado na figura: usuário clica com o mouse sobre o texto da cena 3D ativando o *TouchSensor*, colocando em funcionamento o *Script node eventIn startTime*, que chama o método *processEvent()* na classe Java correspondente. Mudanças na descrição e posição do texto são calculadas pela classe Java que são retornadas para o *Script node* como valores *eventOut*;
- acontece o evento (2) indicado na figura: é enviado o evento *eventOut ChangedText* para o *node text MessageToUser*, setando o valor final do texto exibido na cena 3D;
- acontece o evento (3) indicado na figura: é enviado o evento *eventOut ChangedPosition* para o *node Transform TextPosition*, modificando a posição do texto exibido na cena 3D.

Figura 4.2 – Interface *Script node* entre VRML e Java



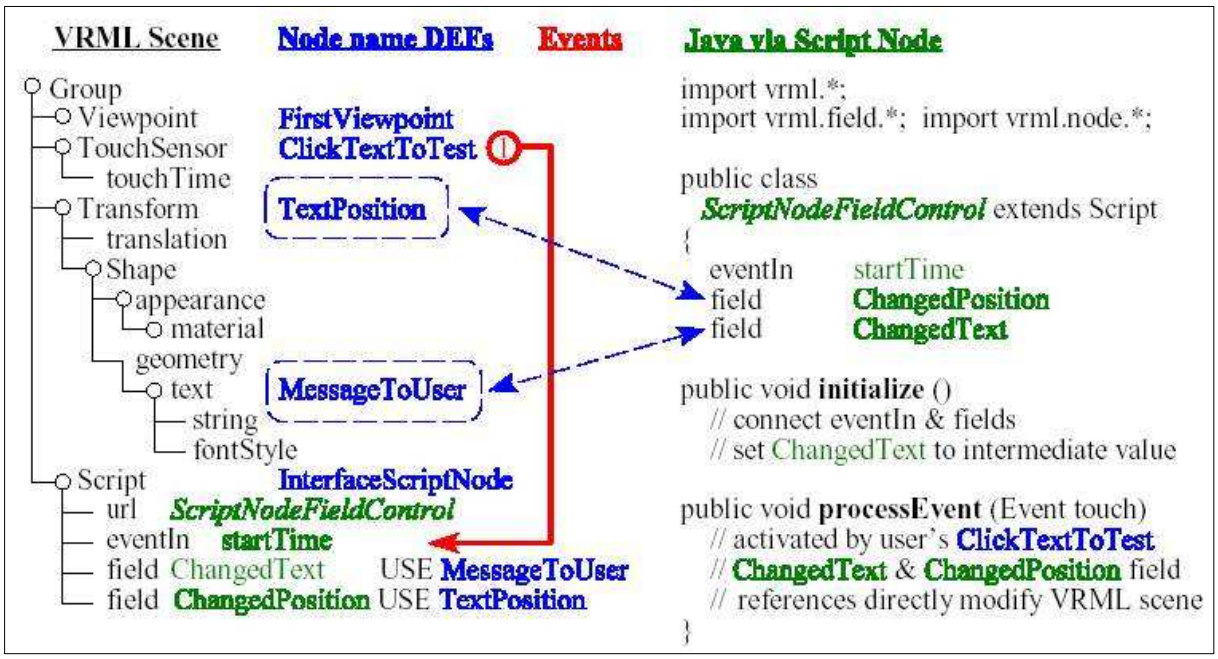

Fonte: Brutzman (1998)

O exemplo completo, com o código fonte da cena VRML, bem como do *script* escrito em Java, referente ao que foi demonstrado acima pode ser encontrado no anexo A deste trabalho.

Outra alternativa, segundo Brutzman (1998), é passar referências para os *nodes* VRML, como *fields*, no *Script node*. Em consequência disso, o Java ganha controle direto sobre *nodes* e *fields* VRML, enviando ou recebendo mensagens de eventos para setar e receber valores. Na inicialização, a classe Java instancia o *node* referenciado como uma variável local, assim, durante subseqüentes pedidos, a classe Java pode ler e modificar

qualquer *field* referenciado na cena gráfica, sem utilizar *ROUTEs*. Na fig. 4.3 tem-se um exemplo mostrando como os *nodes* primeiramente são definidos na cena VRML, e então passados como parâmetros para a classe Java. Esta alternativa será utilizada na implementação do protótipo deste trabalho, portanto será vista em detalhes no capítulo cinco.

Figura 4.3 – Interface *field* entre VRML e Java




Fonte: Brutzman (1998)

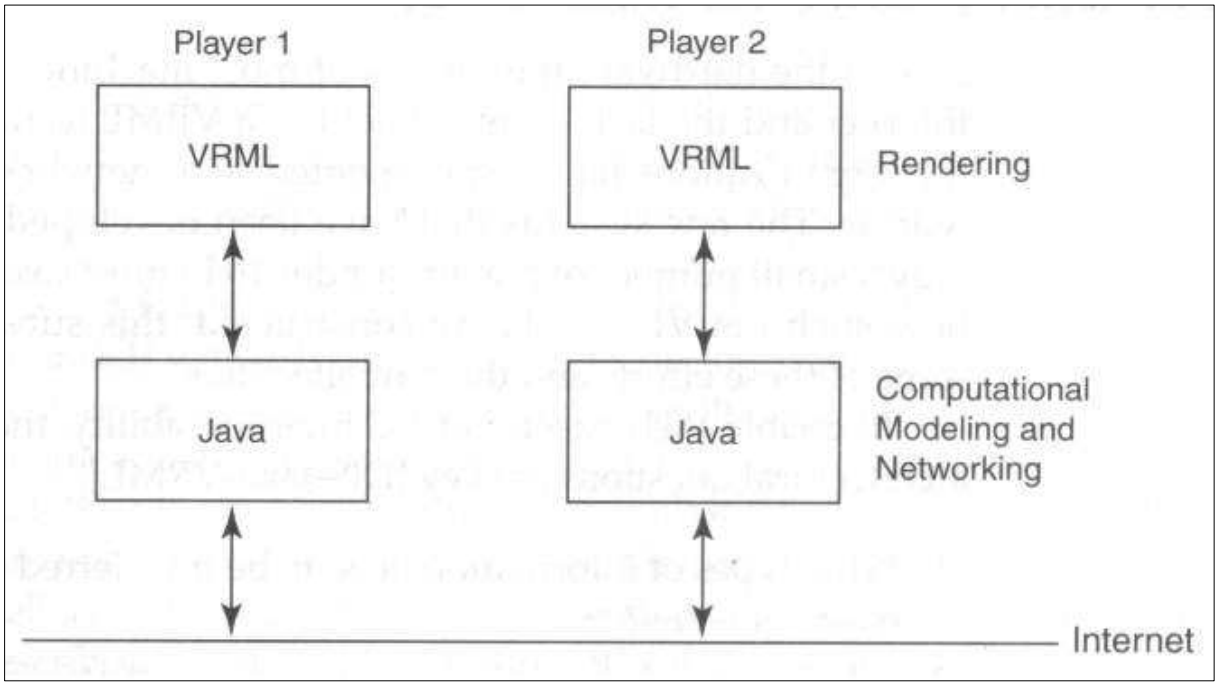
Assim, segundo Brutzman (1998), pode-se dizer que VRML e Java são linguagens poderosas para modelagem 3D, cálculos gerais e acesso a rede. São linguagens compatíveis, bem especificadas, públicas e portáteis para a maioria das plataformas na Internet. Utilizando VRML e Java, experiências práticas e sucesso contínuo levará o campo da realidade virtual passar de ficção e pesquisas isoladas para mesas de trabalho em qualquer lugar, criando a próxima geração da *web*.

## 4.3 A TECNOLOGIA DIS-JAVA-VRML

O projeto DIS-Java-VRML da *Naval Postgraduate School* é um protótipo baseado em ambientes virtuais distribuídos em VRML e que concentra seus esforços em transferir mudanças de estado e interação entre objetos deste tipo de ambiente. Especialmente para criar um novo protocolo, o projeto desenvolveu uma biblioteca de classes Java e uma arquitetura para trocar pacotes DIS pela internet. Como o protocolo já era um padrão IEEE, a tarefa que ficou é desenvolver um software de arquitetura de rede (Singhal, 1999).

Em um alto nível, segundo Singhal (1999), a arquitetura DIS-Java-VRML é semelhante ao que é mostrado na fig. 4.4, onde pode-se ver que a modelagem tridimensional é tratada pela linguagem VRML, e que Java fornece os cálculos e solicitações da rede.

Figura 4.4 – Exemplo em alto nível da estrutura DIS-Java-VRML




Fonte: Singhal (1999)

Atualmente existe um grupo de trabalho do DIS-Java-VRML, o qual está desenvolvendo uma biblioteca de softwares, escritas em Java e trabalhando juntas com DIS e VRML. Este projeto foi criado no início de 1997 e continua ativo, contando com a contribuição de várias pessoas para este código fonte aberto e gratuito sob os termos da GNU (*General Public License*). No passado, os mundos virtuais distribuídos, como por exemplo o NPSNET, eram executados em equipamentos caros e desenvolvidos através de soluções próprias de software, geralmente feitas em C++ e fazendo uso de bibliotecas da Silicon Graphics Incorporation (SGI). Esse fator representava uma certa barreira que dificultava o crescimento na área de desenvolvimento de ambientes virtuais distribuídos. Porém, com o aumento significativo do poder de computação dos PC, redução significativa do seu custo, e a criação do grupo de trabalho do DIS-Java-VRML isto começou a ser superado (Web3D, 2000a).

De acordo com Web3D (2000a), através do DIS, Java e VRML pode-se fornecer todas as habilidades pertinentes e necessárias para a implementação de ambientes virtuais distribuídos. Como foi visto na seção 4.1 o DIS é essencialmente um protocolo direcionado ao comportamento físico, baseado em várias interações, utilizado para comunicar informações

de estado como: posição, orientação, velocidade e aceleração dentre entidades múltiplas participando num ambiente de rede compartilhado. Java é a linguagem de programação utilizada para implementar o protocolo DIS, além de desempenhar cálculos matemáticos, comunicar com a rede e com a cena em VRML. Já a VRML é utilizada para modelar as entidades locais e remotas em mundos virtuais distribuídos.

Segundo Web3D (2000a), os objetivos do grupo de trabalho do DIS-Java-VRML são:

- completar a implementação do protocolo DIS em Java;
- apresentar um grupo de referências e práticas recomendadas para mapeamento entre DIS e mundos VRML;
- classificar utilidades do DIS;
- utilizar bibliotecas matemáticas e físicas.

O próximo capítulo apresenta a descrição do protótipo do mundo virtual distribuído construído como resultado deste trabalho.

# 5 DESENVOLVIMENTO DO PROTÓTIPO

Através dos assuntos abordados nos capítulos anteriores, pôde-se desenvolver o protótipo de um mundo virtual distribuído utilizando a tecnologia DIS-Java-VRML. Uma das principais preocupações na implementação deste protótipo era possibilitar a interação do avatar[5] com o ambiente virtual, além de tornar o protótipo um sistema distribuído.

A seção 5.1 apresenta a especificação desse protótipo, enquanto que sua implementação é apresentada na seção 5.2, e por último, a seção 5.3 apresenta a utilização do mesmo.

## 5.1 ESPECIFICAÇÃO

Como ponto de partida para a especificação do protótipo será apresentada uma visão geral da ligação entre a cena VRML e a classe Java correspondente. Na seqüência, será apresentado o grafo de cena que representa a parte gráfica do protótipo, ou melhor, a representação da estrutura geral do ambiente virtual. Por último, utilizando a linguagem de modelagem UML, serão apresentadas as classes criadas durante a implementação deste protótipo e as classes do protocolo DIS, já criadas pelo grupo de trabalho do DIS-Java-VRML e que, assim como o protótipo, foram implementadas na linguagem Java. É interessante ressaltar que para a especificação do protocolo DIS tomou-se como referência o trabalho de Eduardo (2001).

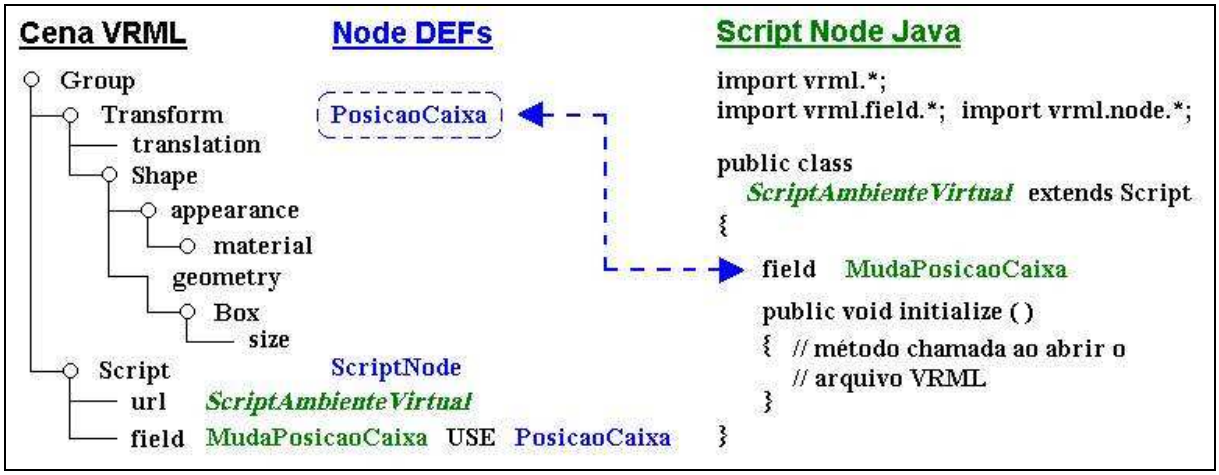
### 5.1.1 CENA VRML X CLASSE JAVA

Conforme apresentado no capítulo quatro, seção 4.2.5, existem duas alternativas para realizar a integração da linguagem VRML com a linguagem Java. São elas: controle baseado em eventos, explicado em detalhes na seção citada, e controle baseado em campos ou *fields*, o qual foi utilizado na implementação deste protótipo. Optou-se por utilizar esta opção por se tratar de uma alternativa onde o Java ganha controle direto sobre os *nodes* e *fields* definidos na cena VRML. Um exemplo identificando o funcionamento desta alternativa pôde ser visto na fig. 4.3, do capítulo anterior, e agora pode ser visto na fig. 5.1, onde, no entanto, teve-se a preocupação de utilizar a mesma estrutura apresentada, porém, adequando-a ao desenvolvimento do protótipo, mostrando assim como o mundo VRML é integrado à classe

[5] Personificação do usuário dentro do ambiente virtual

Java correspondente. Também é importante ressaltar que o mundo VRML criado para este protótipo possui quatro personagens e um objeto que simula a interação com o ambiente virtual, porém, a fig. 5.1 ilustra apenas um dos personagens com o objetivo de facilitar o seu entendimento.

Figura 5.1 – Integração do mundo VRML com a classe Java




Fonte: Adaptado de Brutzman (1998)

O código da cena VRML é simples e composto de sete *nodes* principais, sendo que seis deles são *nodes Transform* e um é *node Script*. Para cada personagem e para o objeto pertencente ao ambiente virtual, todos representados por formas geométricas, foi utilizado o *node Transform* onde foi definido: translação inicial, aparência, tipo de forma geométrica que representa o personagem e tamanho. Após definir estes *nodes* pode-se então iniciar a ligação entre a cena VRML e a classe Java a partir do código VRML. Para isto é necessário ressaltar três pontos importantes. São eles:

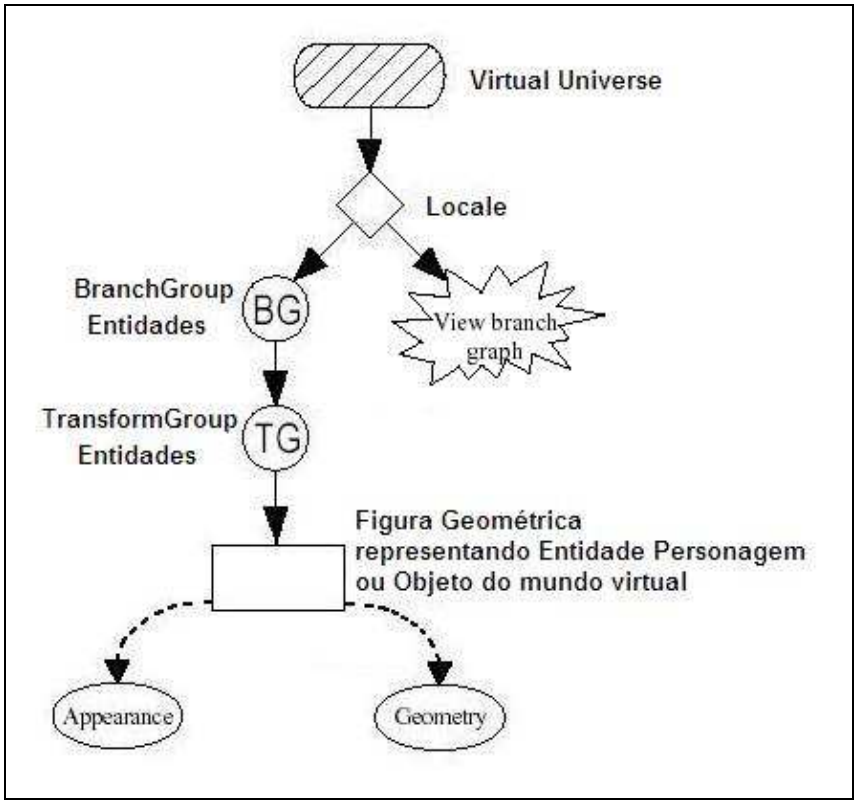
- definição de nomes aos *nodes* – conforme já foi citado no capítulo quatro, seção 4.2.3, é necessário associar nomes aos *nodes* utilizando a opção *DEF* para que os mesmos possam ser utilizados e acessados posteriormente pela classe Java correspondente;
- chamada da classe Java – dentro do *node Script* existe a opção *url* onde deve ser chamada a classe Java correspondente a este *script*;
- definição dos campos – também dentro do *node Script* é necessário definir os campos ou *fields* que farão a ligação entre a cena VRML e a classe Java. Isto sem esquecer de fazer a ligação entre este campo e o *node Transform* correspondente, dentro do próprio código VRML, através da opção *USE*.

Para finalizar tem-se o código da classe Java onde, primeiramente, devem ser importadas as classes do VRML. Após isso, deve-se definir uma classe pública que tenha o mesmo nome dado ao arquivo Java e herdando a classe *Script*, para então, serem definidos todos os campos ou *fields* assim como foram definidos no código VRML.

### 5.1.2 GRAFO DE CENA DO MUNDO VIRTUAL

Uma definição comum para o grafo de cena, segundo Sun (2002), é a de que o mesmo se trata de uma estrutura de dados composta por nós e arcos. De forma geral, pode-se dizer que o grafo de cena serve como uma especificação ou documentação que permite representar graficamente as informações referentes à geometria, sons, luzes, localização, orientação e aparência dos objetos do ambiente virtual. A fig. 5.2 apresenta o grafo de cena que representa o ambiente virtual, isto é, a parte gráfica do trabalho. Sua estrutura mostra um `BranchGroup` raiz onde ficam todos os nós responsáveis por representar os objetos visuais que estarão contidos no ambiente virtual.

Figura 5.2 – Grafo de cena representando o mundo virtual



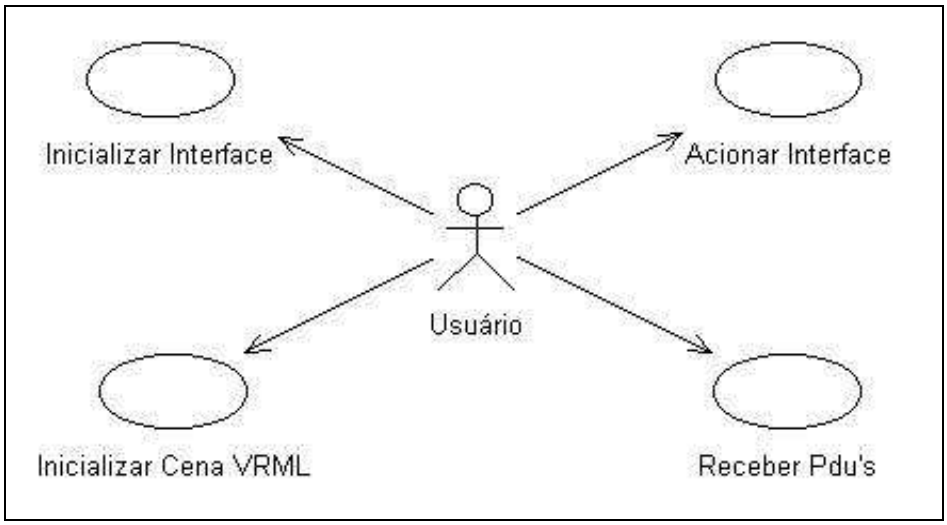
### 5.1.3 VISÃO GERAL DO PROTÓTIPO

Esta seção apresenta uma visão geral da especificação do protótipo, mostrando seus principais processos, através dos casos de uso ou *use-case*, e suas classes, atributos e métodos, através de diagramas de classe utilizando a UML (*Unified Modeling Language*) com o auxílio da ferramenta *Rational Rose*. A fig. 5.3 apresenta o *use-case* com os processos que envolvem o protótipo deste trabalho. São eles:

- inicialização da interface – neste processo será aberta a janela de comandos que permitirá a interação do usuário com o mundo virtual;
- acionamento da interface – este processo representa o acionamento dos botões da janela de comandos resultando no envio de PDU;
- inicialização da cena VRML – processo que representa a inicialização da classe Java correspondente ao arquivo da cena VRML;
- recebimento de PDU – este processo representa o recebimento e processamento dos PDU enviados.

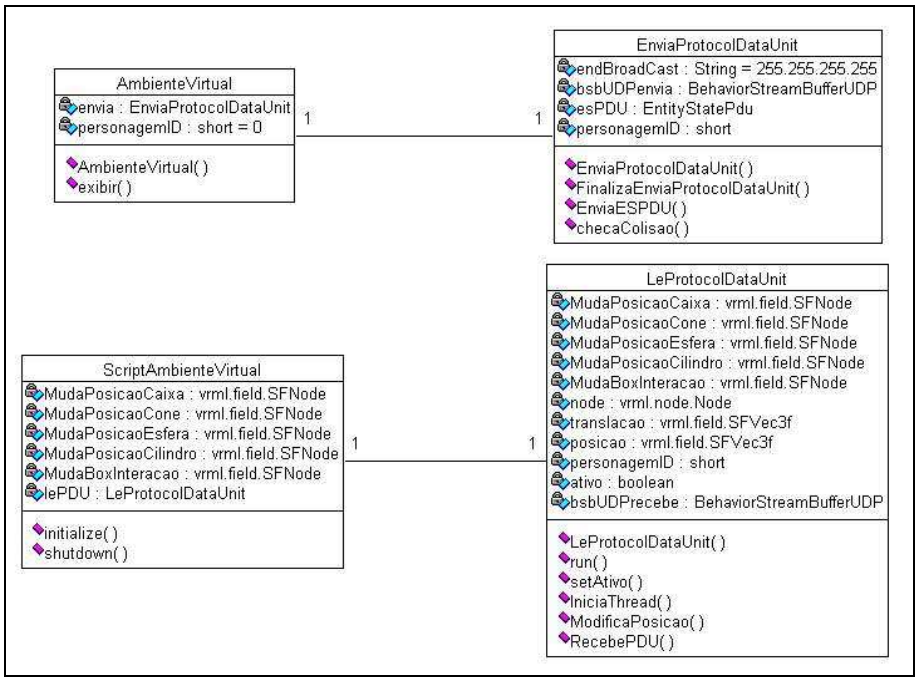
Estes processos serão vistos em detalhes nas próximas seções através de diagramas de seqüência.

Figura 5.3 – *Use-case* representando os processos do protótipo




Tendo uma visão dos processos que envolvem este protótipo pode-se agora detalhar um pouco mais as classes que compõe o mesmo. A fig. 5.4 apresenta o diagrama de classes com seus respectivos atributos e métodos, entretanto, alguns deles foram omitidos para facilitar a sua visualização. O diagrama de classes completo pode ser encontrado no anexo D.

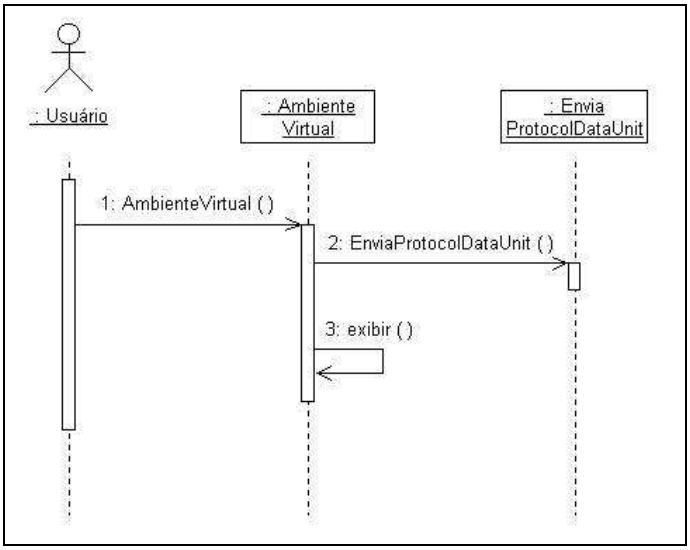
Figura 5.4 – Diagrama de classes utilizadas no protótipo




## 5.1.4 INICIALIZANDO A INTERFACE

O processo de inicialização da interface é aquela que abre a janela de comandos que o usuário poderá utilizar para interagir com o ambiente virtual. Neste processo acontece a instanciação de um objeto da classe `AmbienteVirtual` que inicializa todos os componentes que compõe a janela de comandos e instancia um objeto da classe `EnviaProtocolDataUnit` que é responsável pelo envio de PDU. Por último é chamado o método que de fato mostra a janela criada. A fig. 5.5 ilustra o diagrama de seqüência do processo de inicialização da janela de controles do usuário.

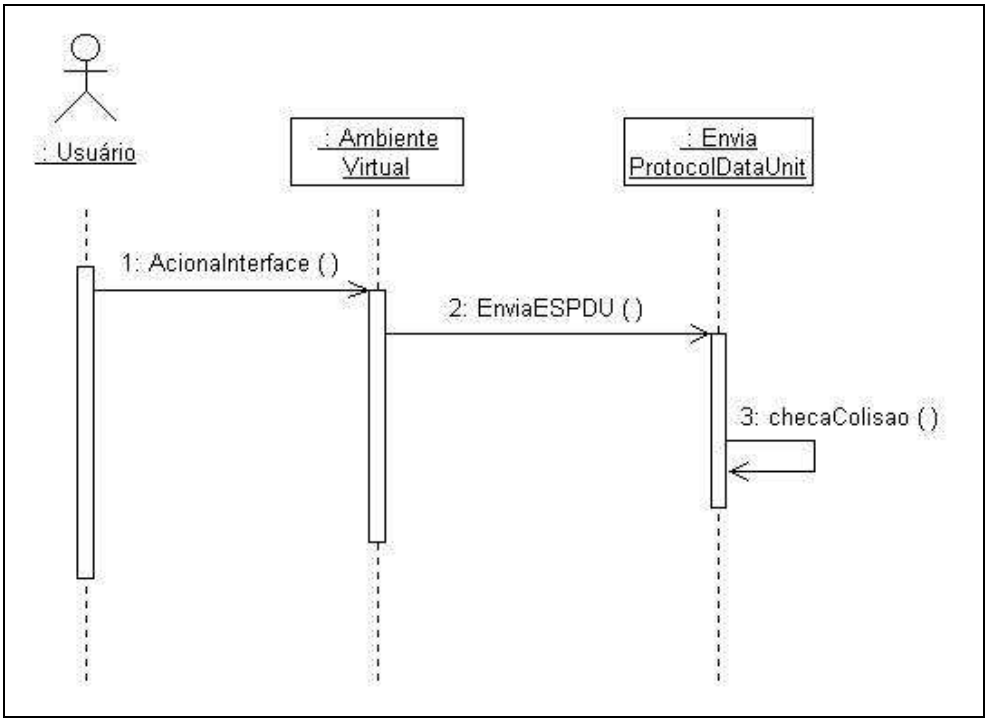
Figura 5.5 – Diagrama de sequência do processo de inicialização da interface



### 5.1.5 ENVIANDO PDU

Este processo é utilizado para o envio de PDU para os usuários participantes do mundo virtual. A cada acionamento de botão da janela de comandos, implementada na classe `AmbienteVirtual`, é enviado um PDU e verificado se ocorreram possíveis colisões através do objeto instanciado da classe `EnviaProtocolDataUnit`. A fig. 5.6 mostra o diagrama de seqüência do processo de envio de PDU.

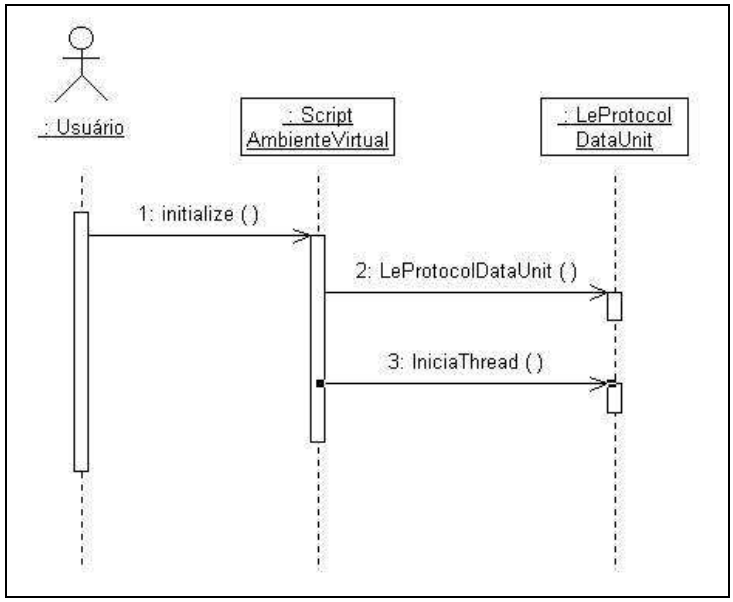
Figura 5.6 – Diagrama de sequência do processo de envio de PDU's




### 5.1.6 INICIALIZANDO A CENA VRML

O processo de inicialização da cena VRML enfoca o mundo virtual do protótipo. No momento em que o arquivo correspondente ao mundo virtual for inicializado a classe `ScriptAmbienteVirtual` também será inicializada, além de ocorrer a instanciação de um objeto da classe `LeProtocolDataUnit`. Este objeto, por sua vez, corresponde a uma *thread* que é inicializada para ficar recebendo e processando os PDU enviados pelos usuários participantes. A fig. 5.7 apresenta o processo de inicialização da cena VRML através de um diagrama de seqüência.

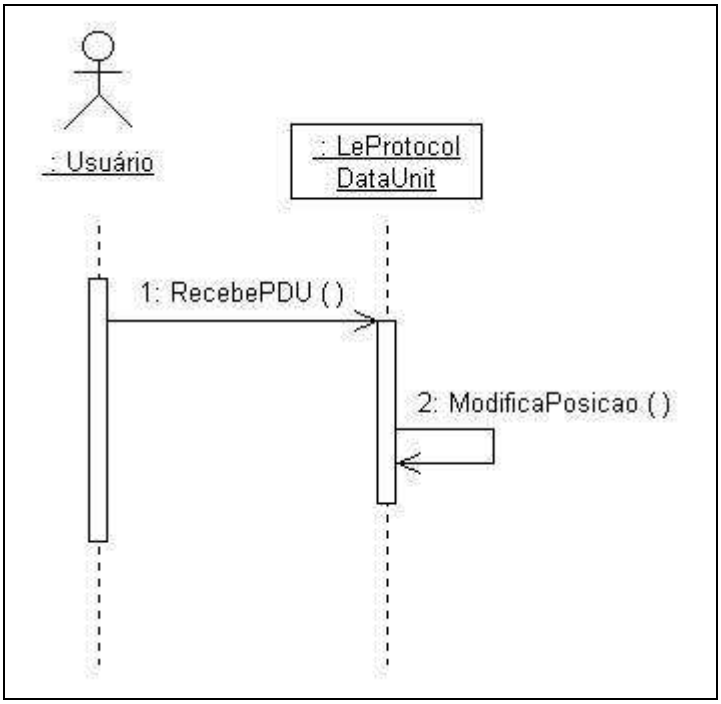
Figura 5.7 – Diagrama de sequência do processo de inicialização da cena VRML




### 5.1.7 RECEBENDO PDU

Este processo, conforme citado na seção anterior, é responsável pelo recebimento e processamento dos PDU enviados pelos usuários participantes do mundo virtual. Quando o objeto desta classe é iniciado pela classe `ScriptAmbienteVirtual`, o mesmo fica recebendo os PDU enviados e então modificando a posição dos personagens. A fig. 5.8 ilustra o diagrama de seqüência que representa o processo de recebimento de PDU.

Figura 5.8 – Diagrama de sequência do processo de recebimento de PDU



### 5.1.8 PRINCIPAIS CLASSES DO DIS-JAVA-VRML QUE IMPLEMENTAM O PROTOCOLO DIS

Conforme descrito com maiores detalhes na seção 4.1, o protocolo DIS é formado por vários tipos de PDU utilizados para informar sobre as mudanças ou eventos que podem ocorrer num ambiente virtual distribuído. Segundo Web3D (2000a), aproximadamente dois terços desses PDU estão implementados na tecnologia DIS-Java-VRML. Considerando que esta quantidade de tipos de PDU é um número consideravelmente alto, cerca de 27, serão apresentadas somente as classes correspondentes a alguns tipos de PDU, mais especificamente, os PDU herdeiros da classe `ProtocolDataUnit` (`EntityStatePdu`, `CollisionPdu`, `DetonationPdu` e `FirePdu`), sendo que, neste trabalho foi utilizado somente o PDU `EntityStatePdu`.

Antes de serem vistas as classes correspondentes aos tipos de PDU citados, é interessante que sejam vistas as classes cujos objetos instanciados permitem controlar o envio e recebimento de PDU.

#### 5.1.8.1 Classe BehaviorStreamBufferUDP

O DIS-Java-VRML possui uma classe cujo objeto instanciado permite enviar e receber PDU num ambiente virtual. Essa classe é chamada `BehaviorStreamBufferUDP`, e seus objetos são utilizados para enviar e receber PDU sobre uma comunicação não orientada a conexão. O quadro 5.1 descreve os principais construtores e métodos dessa classe.

Quadro 5.1 – Principais construtores e métodos da classe `BehaviorStreamBufferUDP`

**Construtores da classe BehaviorStreamBufferUDP**

**BehaviorStreamBufferUDP ()** – instancia um objeto da classe BehaviorStreamBufferUDP que utilizará um modelo de comunicação *unicast*, utilizando-se uma porta de comunicação escolhida pelo sistema operacional.

**BehaviorStreamBufferUDP (DatagramStreamBuffer datagramaParaComunicacao)** – instancia um objeto da classe BehaviorStreamBufferUDP passando como parâmetro um objeto da classe DatagramStreamBuffer, sobre o qual será feita a comunicação.

**BehaviorStreamBufferUDP (int portaConexao)** – instancia um objeto da classe BehaviorStreamBufferUDP passando como parâmetro a porta na qual esse objeto fará a comunicação.

**BehaviorStreamBufferUDP (java.lang.String enderecoMulticast, int portaConexao)** – instancia um objeto da classe BehaviorStreamBufferUDP passando como parâmetro um endereço do tipo *multicast* para o qual esse objeto enviará os PDU e o número da porta na qual estabelecer a conexão.

**Principais métodos da classe BehaviorStreamBufferUDP**

**void cleanup ()** – esse método fecha todas as portas de comunicação utililizadas pelo objeto em questão.

**java.util.Vector receivedPdus ()** – esse método retorna um objeto da classe Vector contendo todos os PDU lidos desde a última vez em que um PDU lido pelo objeto foi recuperado.

**void resumeReading ()** e **void suspendReading ()** – esses método fazem com que o objeto da classe específica (BehaviorStreamBufferTCP ou UDP) volte a ler ou pare, respectivamente, os pacotes enviados para o computador.

**void sendPdu (ProtocolDataUnit pduASerEnviado)** – esse método é utilizado para enviar um PDU.

**void sendPdu (ProtocolDataUnit pduASerEnviado, java.lang.Object enderecoDestino, java.lang.Objecto informacaoAdicional)** – a finalidade desse método também é enviar um PDU, porém o parâmetro enderecoDestino permite especificar o endereço ou nome do *host* de destino desse PDU. O parâmetro informacaoAdicional pode ser nulo, ou se desejado o número da porta de conexão.

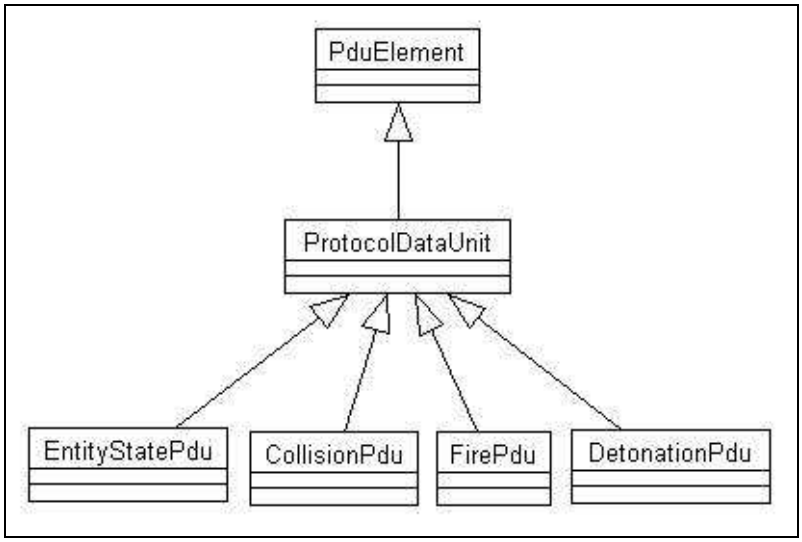
**void setTimeToLive (int TTL)** – método utilizado para definir o tempo de vida dos pacotes enviados por esse objeto. Os valores indicados na documentação do DIS-Java-VRML são 15 para uma comunicação local, 63 para regional e 127 para global.

**void shutdown ()** – método utilizado para terminar o *loop* interno principal do objeto, finalizando a *thread* a qual esse objeto está associado.

### 5.1.8.2PDU herdeiros da classe ProtocolDataUnit

A fig. 5.9 apresenta um diagrama de classes simplificado que ilustra as superclasses das classes `EntityStatePdu`, `CollisionPdu`, `DetonationPdu` e `FirePdu` (os atributos e métodos das classes não foram exibidos por motivos de clareza visual, tendo em vista que essas classes apresentam um número elevado de atributos e métodos).

Figura 5.9 – Diagrama de classe ilustrando a herança das classes `CollisionPdu`, `EntityStatePdu`, `DetonationPdu` e `FirePdu`




Fonte: Web3D (2000a)

Antes de serem detalhados os principais métodos das subclasses da classe `ProtocolDataUnit`, convém mostrar os principais métodos dessa classe, tendo em vista

que todas as suas subclasses também herdam esses métodos. O quadro 5.2 apresenta esses métodos.

Quadro 5.2 – Principais métodos da classe `ProtocolDataUnit`

| |
|---|
| **UnsignedByte getExerciseID ()**, **void setExerciseID (int identificacaoSimulacao)** e **void setExerciseID (UnsignedByte identificacaoSimulacao)** - o primeiro método retorna um objeto da classe UnsignedByte que representa o número de identificação da simulação atribuída ao objeto dessa classe. O segundo e terceiro métodos permitem definir qual o número de identificação da simulação da qual esse objeto faz parte. |
| **UnsignedByte getPduType ()** e **void setPduType (UnsignedByte tipoPdu)** – o primeiro método retorna um objeto da classe UnsignedByte que representa a qual tipo de PDU o objeto dessa classe pertence. O segundo permite definir a qual tipo de PDU o objeto dessa classe pertence. Os valores válidos para o parâmetro tipoPdu podem ser vistos na classe PduTypeField em Web3D (2000a). |
| **UnsignedByte getProtocolFamily ()**, **void setProtocolFamily (int familiaDeProtocolo)** e **void setProtocolFamily(UnsignedByte familiaDeProtocolo)** – o primeiro método retorna um objeto da classe UnsignedByte que representa a qual família de protocolos o objeto dessa classe pertence. O segundo e terceito métodos permitem definir a qual família de protocolos o objeto dessa classe pertence. Os valores válidos para o parâmetro familiaDeProtocolo podem ser vistos na classe ProtocolFamilyField. |
| **UnsignedByte getProtocolVersion ()**, **void setProtocolVersion (int versaoDoProtocolo)** e **void setProtocolVersion (UnsignedByte versaoDoProtocolo)** – idem aos métodos explicados anteriormente, só que esses métodos se referem a versão do protocolo utilizado. Os valores válidos para o parâmetro versaoDoProtocolo podem ser vistos na classe ProtocolVersionField. |
| **void setSimulationStartTime (long horaInicioSimulacao)** – método utilizado para definir no objeto dessa classe o horário de início da simulação. |
| **void setTimeReceived (long horarioRecebimento)** – método utilizado para definir no objeto dessa classe o horário em que ele foi recebido na aplicação. |

Os objetos da classe `EntityStatePdu`, de acordo com o protocolo DIS, são utilizados para comunicar informações acerca do estado de uma entidade num ambiente virtual. Entre essas informações se encontram, por exemplo, a localização da entidade, sua velocidade linear e angular, orientação, como deve ser sua aparência, partes articuladas existentes na entidade, etc. O quadro 5.3 apresenta os principais métodos dessa classe.

Quadro 5.3 – Principais métodos da classe `EntityStatePdu`

| |
|---|
| **void addArticulationParameter (ArticulationParameter parametroDeArticulacao)** e **ArticulationParameter getArticulationParameterAt (int indicePosicao)** – o primeiro método adiciona um objeto da classe ArticulationParameter à lista de partes articuladas existentes no objeto dessa classe (um objeto da classe ArticulationParameter é responsável por conter informações sobre a parte articulada que será adicionada à entidade). O segundo método permite ter acesso as informações de uma parte articulada específica que foi adicionada à entidade. |
| **UnsignedByte getDeadReckoningAlgorithm ()**, **void setDeadReckoningAlgorithm (int tipoAlgoritmoDeadReckoning)** e **void setDeadReckoningAlgorithm (UnsignedByte tipoAlgoritmoDeadReckoning)** – o primeiro método retorna um objeto da classe UnsignedByte que representa o tipo de algoritmo de *Dead Reckoning* que está sendo usado pelo objeto dessa classe. O segundo e terceiro métodos permitem especificar qual é o tipo de algoritmo de *Dead Reckoning* utilizado. Os valores válidos para o parâmetro tipoAlgoritmoDeadReckoning podem ser vistos na classe |

DeadReckoningAlgorithmField.

**byte[] getDeadReckoningParameters ()** e **void setDeadReckoningParameters (byte[] parametrosAdicionaisDeadReckoning** – esses dois métodos permitem ter acesso e especificar, respectivamente, informações adicionais necessárias ao algoritmo de *Dead Reckoning*.

**AngularVelocity getEntityAngularVelocity ()** e **void setEntityAngularVelocity (AngularVelocity velocidadeAngularEntidade)** – esses métodos permitem ter acesso e especificar, respectivamente, a velocidade angular da entidade. Esses métodos utilizam um objeto da classe AngularVelocity responsável por armazenar as informações de velocidade de rotação (em radianos por segundo) da entidade sobre cada um dos seus eixos (x, y e z).

**UnsignedInt getEntityAppearance ()**, **void setEntityAppearance (int tipoAparenciaEntidade)** e **void SetEntityAppearance (UnsignedInt tipoAparenciaEntidade)** – o primeiro método retorna um objeto da classe UnsignedInt que representa a aparência da entidade quando representada no ambiente virtual. O segundo e terceito métodos permitem definir qual o tipo de aparência que a entidade deve ter. Os valores válidos para o parâmetro tipoAparenciaEntidade estão definidos nas várias classes que iniciam com o nome EntityAppearance.

**EntityID getEntityID ()** e **void setEntityID (EntityID identificacaoEntidade)** – esses métodos permitem ter acesso e especificar, respectivamente, a identificação da entidade no ambiente virtual. O objeto da classe EntityID utilizado nesses métodos é responsável por armazenar as informações que vão identificar de forma única a entidade no ambiente virtual. Nesse objeto são armazenados três números que determinam a localidade a qual pertence a entidade (siteID), a aplicação em que está rodando a simulação (applicationID) e a própria entidade (entityID).

**LinearAcceleration getEntityLinearAcceleration ()** e **void setEntityLinearAcceleration (LinearAcceleration velocidadeAceleracaoLinear)** – idem aos métodos que tratam a velocidade angular. Porém, nesse caso o objeto da classe LinearAcceleration é responsável por armazenar as informações de aceleração linear dos eixos x, y e z.

**LinearVelocity getEntityLinearVelocity ()** e **void setEntityLinearVelocity (LinearVelocity velocidadeLinear)** – idem aos métodos anteriores. Porém o objeto da classe LinearVelocity é responsável por armazenar as informações de velocidade linear dos eixos x, y e z.

**WorldCoordinate getEntityLocation ()** e **void setEntityLocation (WorldCoordinate localizacaoEntidade)** – idem aos métodos anteriores. Porém o objeto da classe WorldCoordinate é responsável por armazenar as informações de localização da entidade no ambiente virtual, ou seja, as coordenadas x, y e z da entidade.

**EulerAngle getEntityOrientation ()** e **void setEntityOrientation (EulerAngle angulosOrientacao)** – idem aos métodos anteriores. Porém o objeto da classe EulerAngle é responsável por armazenar as informações de orientação da entidade no ambiente virtual, ou seja o ângulo de rotação em torno do eixo x, y e z.

**EntityType getEntityType ()** e **void setEntityType (EntityType tipoEntidade)** – idem aos métodos anteriores. Porém o objeto da classe EntityType é responsável por armazenar várias informações como o tipo da entidade, o país que desenvolveu a entidade, categoria, etc.

**EntityStatePdu getExemplar ()** e **void setExemplar (EntityStatePdu entityStatePduModelo)** – o método setExemplar permite clonar um objeto da classe EntityStatePdu e manter uma referência interna a esse mesmo objeto para utilização futura. O método getExemplar permite ter acesso a essa cópia clonada.

**UnsignedByte getForceID ()**, **void setForceID (int identificacaoGrupo)** e **void setForceID (UnsignedByte identificacaoGrupo)** – o primeiro método rotorna um objeto da classe UnsignedByte que representa a forma como a entidade é identificada com relação ao grupo, ou seja, se ela é aliada, inimiga, neutra, ou outra. O segundo e terceiro métodos permitem definir qual será a relação dessa entidade com os outros grupos. Os valores válidos para o parâmetro identificacaoGrupo estão definidos na classe ForceFieldID.

Já os objetos da classe `CollisionPdu` são utilizados para comunicar as ocorrências de colisões ocorridas no ambiente virtual. Entre as informações mais relevantes contidas num objeto dessa classe estão a identificação da entidade que está emitindo esse PDU de colisão, a identificação da entidade que está colidindo, uma identificação de evento associada a colisão, o tipo de colisão que está ocorrendo, a velocidade em que a entidade estava na hora da colisão, a massa da entidade e a localização de onde ocorreu a colisão. O quadro 5.4 apresenta os principais métodos dessa classe.

Quadro 5.4 – Principais métodos da classe `CollisionPdu`

| |
|---|
| **EntityID getCollidingEntityID ()** e **void setCollidingEntityID (EntityID identificacaoEntidadeColidindo)** – o primeiro método retorna um objeto da classe EntityID que identifica a entidade que está colidindo. O segundo método permite especificar qual entidade está colidindo. |
| **UnisignedByte getCollisionType ()** e **void setCollisionType (UnsignedByte tipoColisao)** – o primeiro e segundo métodos permitem ter acesso e definir qual o tipo de colisão a qual esse objeto está associado. Os valores válidos para o parâmetro tipoColisao podem ser vistos na classe CollisionTypeField. |
| **EventID getEventID ()** e **void setEventID (EventID identificacaoEvento)** – o primeiro e segundo métodos permitem ter acesso e definir uma identificação de evento associada a essa colisão. O objeto da classe EventID utilizado nesses métodos contém informações referentes a localidade a qual pertence a entidade (siteID), a aplicação em que está rodando a simulação (applicationID) e um número que identifica esse evento. |
| **EntityID getIssuingEntityID ()** e **void setIssuingEntityID (EntityID identificacaoEntidade)** – esses métodos permitem ter acesso e definir a identificação da entidade que está emitindo esse PDU de colisão. |
| **LinearVelocity getLinearVelocity ()** e **void setLinearVelocity (LinearVelocity velocidadeLinear)** – esses métodos permitem ter acesso e definir a velocidade linear dessa entidade no ato da colisão. |
| **EntityCoordinate getLocation ()** e **void setLocation (EntityCoordinate coordenadasEntidade)** – esses métodos permitem ter acesso e definir a localização da entidade que está emitindo esse PDU de colisão. |
| **float getMass ()** e void **setMass (float massaEntidade)** – esses métodos permitem ter acesso e definir a massa da entidade que está emitindo esse PDU de colisão. |

Os objetos da classe `FirePdu` são utilizados para comunicar sobre um tiro que uma entidade esteja disparando. Entre as informações contidas num objeto dessa classe se encontram a identificação da entidade que está atirando, a identificação da entidade alvo desse tiro, a identificação do tiro que está sendo disparado, uma identificação de evento associada ao tiro, a localização do tiro no ambiente virtual, detalhes sobre o tiro, velocidade e o seu alcance. O quadro 5.5 apresenta os principais métodos dessa classe.

Quadro 5.5 – Principais métodos da classe `FirePdu`

| |
|---|
| **BurstDescriptor getBurstDescriptor** e **void setBurstDescriptor (BurstDescriptor detalhesTiro)** – esses métodos permitem ter acesso e definir várias informações associadas ao tiro. No objeto da classe BurstDescriptor estão informações que detalham, por exemplo, categoria do tiro, tipo de explosivo, quantidade disparada por tiro, taxa de disparo, etc. Para maiores detalhes consulte a classe BurstDescriptor. |
| **EventID getEventID ()** e **void setEventID (EventID identificacaoEvento)** – idem aos métodos get/setEventID da CollisionPdu, porém nesse caso o evento está associado ao disparo e não à colisão. |
| **EntityID getFiringEntityID ()** e **void setFiringEntityID (EntityID identificacaoEntidadeDisparando)** – esses métodos permitem ter acesso e definir a idenficação da entidade associada a esse PDU de disparo. |
| **WorldCoordinate getLocationInWorldCoordinate ()** e **void setLocationInWorldCoordinate (WorldCoordinate localizacaoTiroNoAmbiente)** – esses métodos permitem ter acesso e definir a localização do disparo dentro do ambiente virtual. |
| **EntityID getMunitionID ()** e **void setMunitionID (EntityID identifcacaoTiro)** – esses métodos permitem ter acesso e definir a identificação do tiro disparado ao qual esse PDU de disparo está associado. |
| **float getRange ()** e **void setRange (float alcanceTiro)** – esses métodos permitem ter acesso e definir o alcance do tiro disparado. |
| **LinearVelocity getVelocity ()** e **void setVelocity (LinearVelocity velocidadeTiro)** – esses métodos permitem ter acesso e definir a velocidade do projétil que está sendo disparado. |
| **EntityID getTargetEntityID ()** e **void setTargetEntityID (EntityID identificacaoEntidadeAlvo)** – esses métodos permitem ter acesso e definir a identificação da entidade alvo desse disparo. |

Por último, os objetos da classe `DetonationPdu` são utilizados para comunicar a detonação ou impacto de munições. Entre as informações contidas num objeto dessa classe se encontram a identificação da entidade que está atirando, a identificação da entidade alvo desse tiro, a identificação do tiro que está sendo disparado e uma identificação de evento associada ao tiro. Como a maioria dos métodos são idênticos aos da classe `FirePdu`, não será necessário apresentá-los novamente.

## 5.2 IMPLEMENTAÇÃO

Para o desenvolvimento deste protótipo utilizou-se a linguagem Java, mais especificamente, a JSDK 1.3. Na criação e manipulação do mundo virtual foi utilizada a versão 2.0 da linguagem VRML e para tornar este protótipo um sistema distribuído utilizou-se o protocolo DIS, que juntamente com as linguagens citadas anteriormente, forma a tecnologia DIS-Java-VRML, a qual, até a conclusão deste trabalho não possuía uma versão atribuída.

Primeiramente, será exibido o código fonte gerado para a criação dos personagens e objetos do mundo virtual. Na seqüência será apresentado o código da interface de comandos que permite a interação do usuário com o mundo virtual, o código da ligação da linguagem Java com a VRML, e depois o envio e recebimento de PDU através do protocolo DIS. É

importante lembrar que alguns trechos de código que serão apresentados foram simplificados para dar maior clareza. Por esse motivo algumas linhas de código podem ter sido omitidas.

### 5.2.1 CRIAÇÃO DO MUNDO VRML

Esta seção ilustra a criação dos personagens e objetos que fazem parte do mundo virtual deste protótipo. O quadro 5.6 apresenta o código fonte da criação de um dos personagens representado por uma caixa. Para os outros personagens e objetos do mundo virtual o código é basicamente o mesmo, modificando somente os nomes definidos para os *nodes* e os atributos relacionados à forma geométrica, conforme apresentado no capítulo três, seção 3.3.

Quadro 5.6 – Código fonte da criação de um dos personagens do mundo virtual

```
...
# definição do nome PosicaoCaixa para o node Transform, o qual,
# será utilizado para fazer a ligação VRML x Java e permitir
# acesso a este node através da classe Java
DEF PosicaoCaixa Transform {
  # translação definindo posição inicial do personagem
  translation -5 3 0
  children [
    Shape {
      # definindo da cor do personagem
      appearance Appearance {
        material Material {
          diffuseColor 0 0 1
        }
      }
      # definindo a forma geométrica do personagem - caixa
      geometry Box {
        # atributo tamanho
        size 1 1 1
      }
    }
  ]
}
...
```

### 5.2.2 INTERFACE COM O USUÁRIO

A interface com o usuário corresponde à janela de comandos que irá possibilitar a interação com o mundo virtual. Para isto, procurou-se criar uma interface simples, de fácil entendimento, onde o usuário não terá maiores problemas em sua utilização. No quadro 5.7 tem-se o código da classe de criação desta interface.

Quadro 5.7 – Código fonte de criação da interface com o usuário

```
...
// construtor da classe
public AmbienteVirtual() {
  // instanciando objeto para envio de Pdus
  envia = new EnviaProtocolDataUnit();

  // evento chamado ao fechar o frame
  addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent e) {
        envia.FinalizaEnviaProtocolDataUnit();
        dispose();
        System.exit(0);
    }
  });

  // instanciando panels auxiliares
  pn1 = new Panel();   pn2 = new Panel();   pn3 = new Panel();
  pn4 = new Panel();   pn5 = new Panel();

  // Label, campo e botões X
  lbX = new Label("Posição X");
  txX = new TextField(5);        // caixa de texto
  txX.setText("-5");             // setando valor da caixa de texto
  btX1 = new Button("+ ");
  btX2 = new Button(" - ");

  ...

  // Opções de Personagens
  lbPersonagem = new Label("Personagem");
  grupoOpcoes = new CheckboxGroup();
  cbCaixa = new Checkbox("Caixa",grupoOpcoes,true);
  cbCone = new Checkbox("Cone",grupoOpcoes,false);
  cbEsfera = new Checkbox("Esfera",grupoOpcoes,false);
  cbCilindro = new Checkbox("Cilindro",grupoOpcoes,false);

  ...

  // ao clicar no botão seta caixa de texto e envia PDU
  btX1.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent evt) {
      txX.setText(String.valueOf((Integer.parseInt(
                                          txX.getText()))+1));
      // enviando ESPDU
      envia.EnviaESPDU(personagemID,
                       Integer.parseInt(txX.getText()),
                       Integer.parseInt(txY.getText()),
                       Integer.parseInt(txZ.getText()));
    }
  });

}  // fim do construtor

// distribui componentes no frame e mostra na tela
public void exibir() {
  // Panels
  this.setLayout(new GridLayout(1,7));
  this.add(pn1);   this.add(pn2);   this.add(pn3);
  this.add(pn4);   this.add(pn5);

  // Labels
```

```
      pn1.setLayout(new GridLayout(4,1));
      pn1.add(lbPersonagem);
      pn1.add(lbX);    pn1.add(lbY);      pn1.add(lbZ);

      // Checkbox caixa, Caixas de texto
      pn2.setLayout(new GridLayout(4,1));
      pn2.add(cbCaixa);
      pn2.add(txX);    pn2.add(txY);    pn2.add(txZ);

      // Checkbox cone, Botões de soma
      pn3.setLayout(new GridLayout(4,1));
      pn3.add(cbCone);
      pn3.add(btX1);   pn3.add(btY1);   pn3.add(btZ1);

      // Checkbox esfera, Botões de subtração
      pn4.setLayout(new GridLayout(4,1));
      pn4.add(cbEsfera);
      pn4.add(btX2);   pn4.add(btY2);   pn4.add(btZ2);

      // Checkbox cilindro
      pn5.setLayout(new GridLayout(4,1));
      pn5.add(cbCilindro);

      this.setVisible(true);
   }
   ...
```

### 5.2.3 INTEGRANDO A CENA VRML COM A CLASSE JAVA

O quadro 5.8 apresenta o código do mundo VRML onde está definido o *node Script*. Conforme citado na seção 5.2.1, é nesta parte do código que serão utilizados os nomes definidos para os *nodes Transform*, e onde serão definidos os campos que permitirão o acesso da classe Java correspondente.

Quadro 5.8 – Código fonte definindo o *node Script* na cena VRML

```
   ...
   DEF ScriptNode Script {
      mustEvaluate TRUE
      directOutput TRUE

      # definição dos campos que estão ligados aos nodes Transform
      # dos personagens e que serão utilizados pela
      # classe Java correspondente
      field SFNode MudaPosicaoCaixa    USE PosicaoCaixa
      field SFNode MudaPosicaoCone     USE PosicaoCone
      field SFNode MudaPosicaoEsfera   USE PosicaoEsfera
      field SFNode MudaPosicaoCilindro USE PosicaoCilindro
      field SFNode MudaBoxInteracao    USE PosicaoBoxInteracao

      # classe Java que tem acesso aos campos definidos acima
      url  "ScriptAmbienteVirtual.class"
   }
   ...
```

A seguir, no quadro 5.9, tem-se o código fonte da classe Java correspondente ao arquivo VRML contendo o *node Script* definido no quadro 5.8, apresentado anteriormente.

Quadro 5.9 – Código fonte da classe Java ligada a cena VRML

```
...
public class ScriptAmbienteVirtual extends Script {

   // campos field ligados ao arquivo VRML
   private vrml.field.SFNode MudaPosicaoCaixa;
   private vrml.field.SFNode MudaPosicaoCone;
   private vrml.field.SFNode MudaPosicaoEsfera;
   private vrml.field.SFNode MudaPosicaoCilindro;
   private vrml.field.SFNode MudaBoxInteracao;

   // thread  que fica lendo as Pdus enviadas
   private LeProtocolDataUnit lePDU;

   // método chamado ao abrir o arquivo VRML através do browser
   public void initialize() {
      System.out.println("Inicializando Script ...");
      // buscando node correspondente no arquivo VRML
      MudaPosicaoCaixa = (SFNode) getField ("MudaPosicaoCaixa");
      MudaPosicaoCone = (SFNode) getField ("MudaPosicaoCone");
      MudaPosicaoEsfera = (SFNode) getField ("MudaPosicaoEsfera");
      MudaPosicaoCilindro = (SFNode) getField ("MudaPosicaoCilindro");
      MudaBoxInteracao = (SFNode) getField ("MudaBoxInteracao");

     // iniciando thread da leitura de Pdus enviando os campos da cena
     // VRML como parâmetro para serem modificados após a leitura
     // do PDU
     lePDU = new LeProtocolDataUnit(MudaPosicaoCaixa,
                    MudaPosicaoCone,
                    MudaPosicaoEsfera,
                    MudaPosicaoCilindro,
                    MudaBoxInteracao);
     lePDU.IniciaThread();

     return;
   }

   // método chamado ao fechar a cena VRML
   public void shutdown() {
      lePDU.setAtivo(false);
      lePDU.destroy();
   }
}
```

### 5.2.4 ENVIO E RECEBIMENTO DE PDU

Conforme demonstrado na seção 5.2.2, o objeto da classe que realiza o envio de PDU para os demais participantes é instanciada e utilizada pela classe de interface com o usuário. Assim, toda vez que o usuário clicar em algum botão será enviado um PDU. O quadro 5.10 demonstra o código fonte da classe responsável pelo envio de PDU.

Quadro 5.10 – Código fonte da classe responsável pelo envio de PDU's

```
...
class EnviaProtocolDataUnit extends Object {

   // endereço broadcast utilizado para o envio de PDU's
   private String endBroadCast = new String("255.255.255.255");
   // utilizado para o envio de PDU's
   private BehaviorStreamBufferUDP bsbUDPenvia;
   private EntityStatePdu esPDU = new EntityStatePdu();

   // guarda coordenadas de cada participante para checagem de colisão
   private float coordXYZCaixa[]     = new float[3];
   private float coordXYZCone[]     = new float[3];
   private float coordXYZEsfera[]    = new float[3];
   private float coordXYZCilindro[]  = new float[3];
   private float coordXYZInteracao[] = new float[3];
   private int vlrX, vlrY, vlrZ;

   // identificação do personagem
   private short personagemID;



   // construtor da classe
   public EnviaProtocolDataUnit() {
     // instancia objeto da classe utilizando a porta 8134
      bsbUDPenvia = new BehaviorStreamBufferUDP(8134);
     ...
   }

   public void FinalizaEnviaProtocolDataUnit() {
      System.out.println("Finaliza classe EnviaProtocolDataUnit ...");
      bsbUDPenvia.shutdown();
   }

   // seta informacoes de posicao para ESPDU e envia para os demais
   // participantes
   public void EnviaESPDU(short personagem, float x, float y, float z){
      System.out.println("Enviando ESPDU ...");
      personagemID = personagem;
     // setando identificação do PDU
      esPDU.setEntityID((short)1,(short)1,personagemID);
     // setando localização do personagem
      esPDU.setEntityLocationX((double)x);
      esPDU.setEntityLocationY((double)y);
      esPDU.setEntityLocationZ((double)z);
      bsbUDPenvia.setTimeToLive(15);
      Thread enviaPDU = new Thread(bsbUDPenvia);
     // enviando PDU do tipo ESPDU
      bsbUDPenvia.sendPdu(esPDU, endBroadCast, 8133);
      checaColisao(x,y,z);
   }

   // checa se a entidade que está sendo enviada vai colidir com outra
   // entidade e em caso afirmativo envia uma pdu para deslocar a mesma
   public void checaColisao(float x, float y, float z) {
      System.out.println("Checando colisao com outros personagens ...");
      vlrX = 0; vlrY = 0; vlrZ = 0;
     // verifica qual das coordenadas foi modificada e se a mesma
     // aumentou ou diminuiu setando variável que será incrementada à
     // coordenada do personagem que sofreu a colisão
      if (personagemID == 0) {
```

```
            if (x != coordXYZCaixa[0]) {
               if (x < coordXYZCaixa[0]) {
                  vlrX = -1;
               } else { vlrX = 1; }
            } else if (y != coordXYZCaixa[1]) {
                  if (y < coordXYZCaixa[1]) {
                     vlrY = -1;
                  } else { vlrY = 1; }
            } else if (z != coordXYZCaixa[2]) {
                  if (z < coordXYZCaixa[2]) {
                     vlrZ = -1;
                  } else { vlrZ = 1; }
            }
            // verifica se colidiu com outro personagem e então envia um PDU
            if ((x == coordXYZCone[0]) &&
                  (y == coordXYZCone[1]) &&
                  (z == coordXYZCone[2])) {
                  System.out.println("Personagem Caixa colidiu ...");
                  EnviaESPDU((short)1,x+vlrX,y+vlrY,z+vlrZ);
            }
            // este processo ocorre com todos os personagens e objetos
            ...
      }
```

Em seguida, no quadro 5.11, tem-se o código fonte da classe responsável pela leitura de PDU e atualização do ambiente virtual de acordo com as informações enviadas pelos participantes.

Quadro 5.11 – Código fonte da classe responsável pela leitura de PDU's

```
    ...
    class LeProtocolDataUnit extends Thread {

      private boolean ativo = true;
      private BehaviorStreamBufferUDP bsbUDPrecebe;

      // campos field ligados ao arquivo VRML
      private vrml.field.SFNode MudaPosicaoCaixa;
      private vrml.field.SFNode MudaPosicaoCone;
      private vrml.field.SFNode MudaPosicaoEsfera;
      private vrml.field.SFNode MudaPosicaoCilindro;
      private vrml.field.SFNode MudaBoxInteracao;

      // utilizado para modificar a posição do personagem
      private vrml.node.Node node;
      private vrml.field.SFVec3f translacao;
      private vrml.field.SFVec3f posicao;

      private short personagemID;

      // construtor da classe
      LeProtocolDataUnit(vrml.field.SFNode mpCaixa,
                         vrml.field.SFNode mpCone,
                         vrml.field.SFNode mpEsfera,
                         vrml.field.SFNode mpCilindro,
                         vrml.field.SFNode mpInteracao) {
        System.out.println("Inicializando thread ...");
        MudaPosicaoCaixa = mpCaixa;
        MudaPosicaoCone = mpCone;
```

```
        MudaPosicaoEsfera = mpEsfera;
        MudaPosicaoCilindro = mpCilindro;
        MudaBoxInteracao = mpInteracao;

        bsbUDPrecebe = new BehaviorStreamBufferUDP(8133);
        Thread leitorPDU = new Thread(bsbUDPrecebe);
        leitorPDU.start();
      }

      void setAtivo(boolean at) {
        ativo = at;
      }

      public void ModificaPosicao() {
        // busca o campo translacao do node correspondente
        // e seta nova posição
        translacao = (SFVec3f) node.getExposedField("translation");
        translacao.setValue (posicao);
      }

      public void RecebePDU() {
        // vetor recebendo Pdus enviadas
        java.util.Vector vetorPDU = bsbUDPrecebe.receivedPdus();
        // percorre o vetor verificando qual é o personagem e
        // modifica a posicao
        for (Enumeration enum = vetorPDU.elements();
             enum.hasMoreElements();) {
             // pegando PDU do vetor
             ProtocolDataUnit tempPDU =
            (ProtocolDataUnit) enum.nextElement();
             // verificando se tipo de PDU é EntityStatePdu
             UnsignedByte tipoPDU = tempPDU.getPduType();
             switch (tipoPDU.intValue()) {
                case PduTypeField.ENTITYSTATE: {
                   System.out.println("Lendo Pdu enviada ...");
                   // conversão PDU para ESPDU
                   EntityStatePdu tempESPDU = (EntityStatePdu) tempPDU;
                   personagemID =
                   tempESPDU.getEntityID().getEntityID().shortValue();
                   // setando variável posição a partir do PDU recebido
                   posicao = new SFVec3f(
                                (float)tempESPDU.getEntityLocationX(),
                                (float)tempESPDU.getEntityLocationY(),
                                (float)tempESPDU.getEntityLocationZ());
                   // pegando node a partir da identificação do personagem
                   if (personagemID == 0) {
                        node = (Node) (MudaPosicaoCaixa.getValue());
                   } else if (personagemID == 1) {
                        node = (Node) (MudaPosicaoCone.getValue());
                   } else if (personagemID == 2) {
                        node = (Node) (MudaPosicaoEsfera.getValue());
                   } else if (personagemID == 3) {
                        node = (Node) (MudaPosicaoCilindro.getValue());
                   } else {
                        node = (Node) (MudaBoxInteracao.getValue());
                   }
                   this.ModificaPosicao();
                   break;
                }
             }
        } // fim do for
      }
```

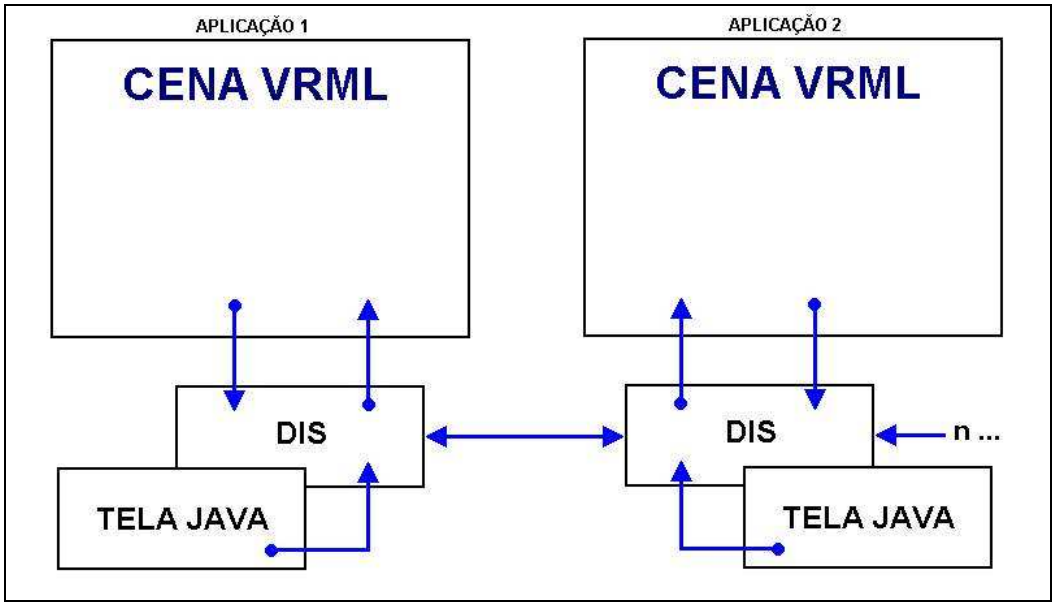
```
    public void IniciaThread() {
       this.start();
    }

    // método chamado ao iniciar a thread
    // fica recebendo PDUs até setar variável ativo = FALSE
    public void run() {
       while (ativo) {
          this.RecebePDU();
          // depois de processor um vetor de Pdus dorme 250ms
          try {
             Thread.sleep(250);
          } catch (Exception e) { }
       }
       bsbUDPrecebe.cleanup();
       bsbUDPrecebe.shutdown();
       System.out.println("Finalizando thread da leitura de Pdus ...");
    }
 }
```

## 5.3 FUNCIONAMENTO DO PROTÓTIPO

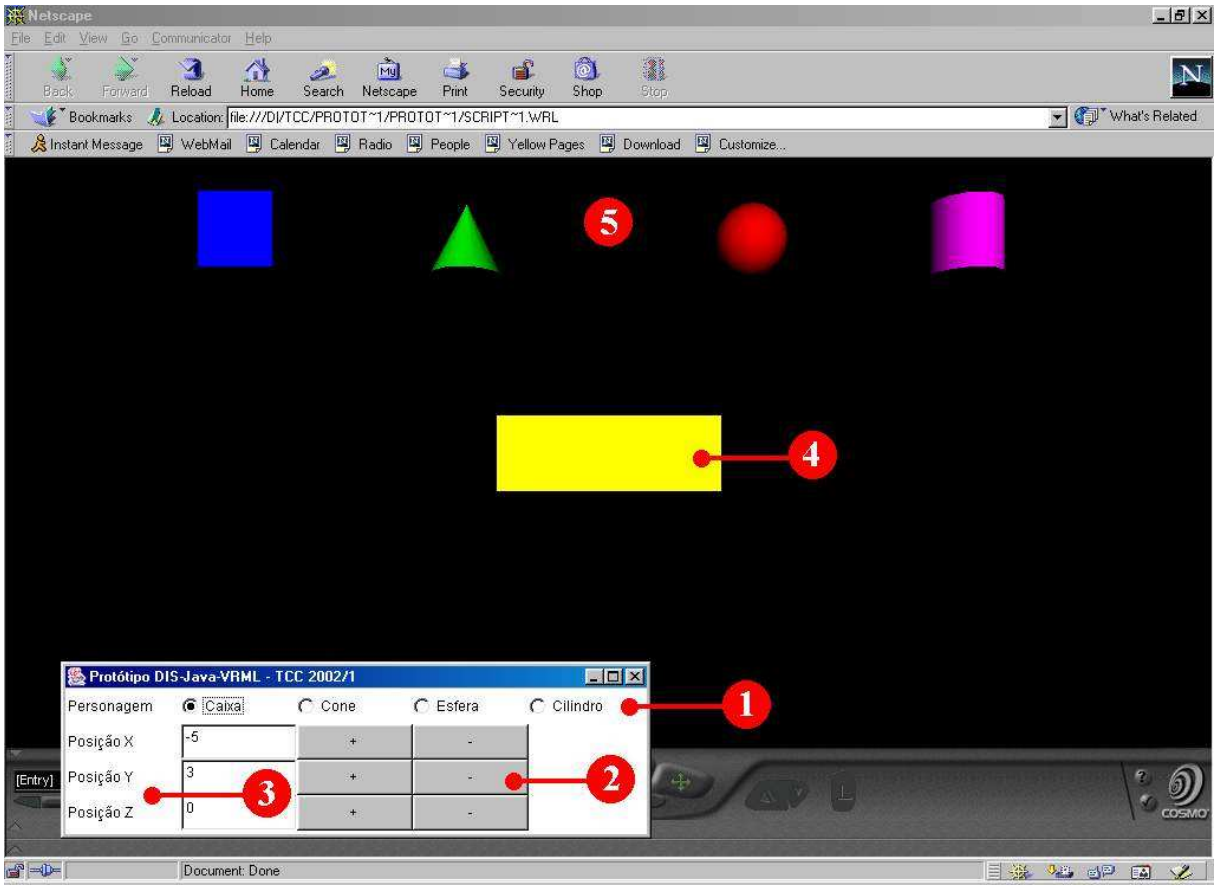
Nesta seção será apresentada a interface do protótipo, bem como seu funcionamento em geral. Possíveis dúvidas relacionadas a execução do protótipo podem ser encontradas no anexo B. O funcionamento da parte distribuída do ambiente virtual é baseada no envio de mensagens através do endereçamento *broadcast* e foi implementada como sendo do tipo homogêneo replicado, apresentado em detalhes na seção 2.4. A fig. 5.10 apresenta o funcionamento da parte distribuída do protótipo que pode ter várias aplicações interligadas através do protocolo DIS.

Figura 5.10 – Funcionamento da parte distribuída do protótipo



Em relação à interface pode-se dizer que ela é bastante simples e possui basicamente cinco pontos a serem vistos: a escolha do tipo de personagem que se quer controlar no ambiente virtual (1), os botões que alteram as coordenadas dos personagens (2), as coordenadas atuais referentes a localização dos personagens (3), objetos do ambiente virtual (4) e os personagens que podem ser controlados dentro do ambiente virtual (5). Cada um desses itens será visto em maiores detalhes nas seções seguintes. A fig. 5.11 apresenta a interface do protótipo.

Figura 5.11 – Interface do protótipo




## 5.3.1 ESCOLHENDO O PERSONAGEM

Após inicializar o protótipo, o usuário terá aberto em seu computador uma janela do *browser* apresentando o mundo virtual e uma janela implementada em Java, a qual, permitirá ao usuário interagir com o ambiente virtual. Uma das opções disponíveis é a escolha do avatar ou personagem que será controlado pelo usuário. Conforme ilustrado na fig. 5.11 e indicado pelo ponto 1 existem quatro opções disponíveis: caixa, cone, esfera e cilindro. Localizados no mundo virtual através do ponto 5 na fig. 5.11 optou-se por representar os personagens através de formas geométricas por se tratar de primitivas gráficas simples, e

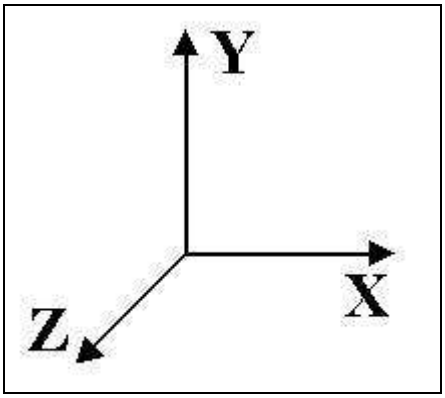
assim, simplificar a implementação dos personagens, já que o objetivo deste trabalho não era relacionado à representação dos mesmos.

Assim que o usuário faz a escolha do personagem que deseja controlar, este volta à sua localização original, conforme indicado pelo ponto 5 na fig. 5.11. Esta localização foi definida internamente durante a implementação do protótipo e será utilizada toda vez que o usuário modificar o seu personagem, opção que o usuário pode utilizar quando desejar. Com isto, poderão acontecer situações onde mais de um usuário estará controlando o mesmo personagem pois não existe um tratamento nesta implementação para que isto não ocorra.

### 5.3.2 LOCALIZAÇÃO NO AMBIENTE VIRTUAL

Os componentes indicados pelo ponto 3 da fig. 5.11 têm por finalidade mostrar as coordenadas de localização nos eixos **X**, **Y** e **Z** do usuário no ambiente virtual. O primeiro exibe as coordenadas no eixo **X**, o segundo as coordenadas no eixo **Y** e terceiro as coordenadas no eixo **Z**. A fig. 5.12 mostra graficamente a representação dos eixos **X**, **Y** e **Z** (mão direita).

Figura 5.12 – Representação dos eixos X, Y e Z




Esses mostradores começam a ser atualizados assim que o usuário entra no ambiente virtual e inicia a navegação no mesmo através do uso dos botões indicados na fig. 5.11 pelo ponto 2. A unidade padrão utilizada para as coordenadas é o centímetro.

### 5.3.3 NAVEGANDO NO AMBIENTE VIRTUAL

Os componentes indicados pelo número 2 na fig. 5.11 são os responsáveis por comandarem as translações dos personagens dentro do mundo virtual. A navegação no ambiente virtual está limitada a movimentos para frente e para trás, correspondentes a

alteração nas coordenadas do eixo **Z**, para esquerda e para direita, correspondentes a alteração nas coordenadas do eixo **X** e para cima e para baixo, correspondentes a alteração nas coordenadas no eixo **Y**.

Ao clicar sobre os botões de navegação é verificada a posição atual do personagem e então, dependendo do comando, é adicionada ou subtraída uma unidade desta posição. A tabela 5.1 apresenta os comandos e ações que são aceitas no protótipo.

Tabela 5.1 – Comandos para navegação no ambiente virtual

| Eixo | Botão | Ação | Observação |
|---|---|---|---|
| Z | - | Desloca avatar para o fundo do ambiente | Subtrai 1cm da coordenada no eixo Z |
| Z | + | Desloca avatar para fora do ambiente | Adiciona 1cm na coordenada do eixo Z |
| X | - | Desloca avatar para a esquerda | Subtrai 1cm da coordenada no eixo X |
| X | + | Desloca avatar para a direita | Adiciona 1cm da coordenada no eixo X |
| Y | - | Desloca avatar para baixo | Subtrai 1cm da coordenada no eixo Y |
| Y | + | Descola avatar para cima | Adiciona 1cm na coordenada do eixo Y |

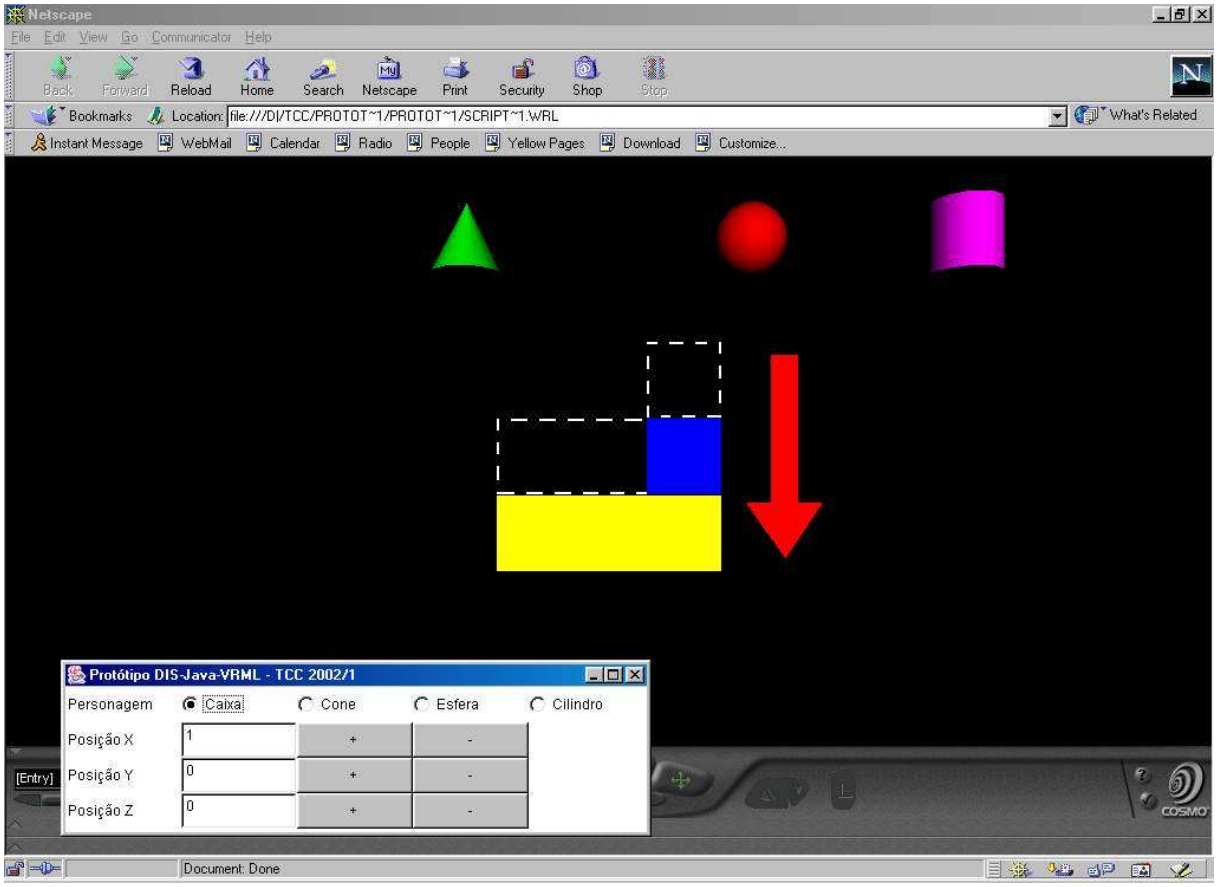
Além de incrementar ou decrementar o valor das coordenadas, também são enviadas PDU para atualizar os mundos virtuais participantes. Assim, após o acionamento dos botões é enviado um *Protocol Data Unit* informando qual é o personagem e o valor dos eixos **X**, **Y** e **Z**, os quais, serão atualizados nos mundos virtuais participantes através de uma rotina que faz a leitura de PDU enviadas.

### 5.3.4 INTERAGINDO COM O AMBIENTE VIRTUAL

Como uma das preocupações na implementação deste protótipo era a de permitir interação com o mundo virtual através de um avatar, foi necessário representar um objeto dentro deste ambiente e então permitir de alguma forma a interação dos personagens com este objeto. Para a representação do mesmo foi criada uma caixa de tamanho diferente conforme indicado pelo ponto 4 na fig. 5.11. Já para representar a interação deste objeto com os personagens foi implementada uma rotina tratadora de colisões, onde toda vez que um participante colidir com o objeto, o mesmo será “empurrado”, ou melhor, ocorrerá uma translação do objeto e as coordenadas serão alteradas de acordo com o sentido em que o personagem veio a colidir com o objeto. A interação de um dos personagens com o ambiente

virtual é demonstrado na fig. 5.13, onde a parte tracejada representa a posição anterior à colisão e a seta mostra o sentido da colisão.

Figura 5.13 – Interação do avatar com o mundo virtual




Também é importante ressaltar que o tratamento de colisão não ocorre somente entre os personagens e o objeto representado pela caixa amarela, mas também, entre os próprios personagens, portanto, assim que um personagem colidir com outros personagens também ocorrerá translação, envio e recebimento de PDU.

# 6 CONCLUSÕES

Com o desenvolvimento deste trabalho foram atingidos os objetivos propostos tendo como resultado final a implementação de um mundo virtual distribuído, contendo personagens e objetos criados através de primitivas gráficas simples, e onde possibilitou-se a interação entre estes respectivos personagens e objetos do ambiente virtual. Para isto, foi realizado um estudo e então demonstrado como a tecnologia DIS-Java-VRML foi utilizada para o desenvolvimento deste ambiente virtual distribuído e suas respectivas funções.

Para que este trabalho pudesse ser realizado, além de um grande empenho sobre o estudo da tecnologia em si, foram necessários cuidados com detalhes relacionados à instalação dos softwares utilizados para o desenvolvimento deste trabalho. A própria documentação da tecnologia DIS-Java-VRML indica os passos a serem seguidos para a sua instalação e a dos softwares necessários na utilização de suas aplicações. Entretanto, o fator que mais prejudicou o andamento deste trabalho foram as versões dos softwares inicialmente utilizadas, pois após testes realizados durante o desenvolvimento do protótipo verificou-se que também é importante respeitar as versões dos softwares indicadas na documentação de instalação do DIS-Java-VRML, caso contrário as aplicações não irão funcionar corretamente.

Em relação à tecnologia estudada verificou-se vários pontos importantes referentes ao DIS, à linguagem Java e VRML que devem ser comentados. O protocolo DIS teve como fator importante para a realização deste trabalho sua facilidade de uso pois suas principais classes já estão implementadas e assim, foi necessário apenas entender o seu funcionamento e utilizá-las no protótipo. Por outro lado, este fator se torna um ponto negativo quando é levado em consideração o fato das classes possuírem PDU desenvolvidas especificamente para aplicações do Departamento de Defesa dos Estados Unidos, pois poderão ocorrer situações onde os PDU existentes não atendam uma aplicação específica e terão de ser desenvolvidos. Outro ponto negativo é o fato dos PDU serem enviados via endereço *broadcast*, opção que utiliza todos os recursos da rede além de poder ocasionar perda de PDU. O endereçamento *broadcast* pode ser um ponto positivo se utilizado em uma rede onde só será executada a aplicação do mundo virtual distribuído, pois assim os recursos da rede estarão disponíveis somente para ela e o envio de PDU será feito uma única vez para todos os participantes.

É importante ressaltar que no desenvolvimento do protótipo utilizou-se apenas a *EntityStatePdu*, porém, a *CollisionPdu*, relacionada ao tratamento de colisões, também poderia ter sido utilizada, entretanto, optou-se por realizar um tratamento de colisões mais

simplificado devido a problemas de comunicação entre a classe responsável pelo recebimento de PDU e classe relacionada a tela de comandos desenvolvida em Java conforme detalhado em anexo neste trabalho (anexo C). Além deste fator relacionado à linguagem Java foram identificados pontos importantes como o fato de as classes do protocolo DIS terem sido implementadas nesta linguagem, facilitando a implementação do protótipo, e o fato de existirem classes desenvolvidas em Java para trabalhar com a linguagem VRML.

Com relação à linguagem VRML verificou-se fatores importantes para o desenvolvimento do protótipo como o fato de não necessitar a implementação de uma interface Java para exibir a cena VRML, pois a mesma pode ser exibida em um *browser* através de *plugin* instalado. Outro ponto importante foi o fácil entendimento da linguagem devido à utilização de primitivas gráficas simples na implementação do protótipo, além de ser uma linguagem que permite acessar sua cena através da linguagem Java. Por outro lado, um ponto negativo que pôde ser identificado na utilização do VRML foi o fato de que, caso o mundo VRML fosse acessado diretamente através do *plugin* instalado, e não pela classe Java de interface, o sistema operacional travava em determinadas situações.

Por último, em relação ao DIS-Java-VRML como um todo, além dos detalhes relacionados à instalação de softwares, encontrou-se dificuldades em relação à documentação de apoio que não teve grande parcela de ajuda no desenvolvimento do protótipo. Para compensar este problema foi utilizada a grande variedade de exemplos disponíveis na instalação do DIS-Java-VRML e a sua especificação (Web3D, 2000a), os quais, contribuíram muito para o trabalho. Com os objetivos alcançados, este trabalho poderá contribuir para futuros trabalhos relacionados à ambientes virtuais distribuídos utilizando a tecnologia DIS-Java-VRML, além de demonstrar que ela permite desde a criação de ambientes virtuais distribuídos mais simplificados até os mais complexos.

## 6.1 EXTENSÕES

As possíveis extensões que podem ser feitas a partir desse trabalho estão enumeradas a seguir:

a) melhorar a aparência do ambiente virtual, incluindo no mesmo elementos adicionais que o tornem mais real, através de representações gráficas mais detalhadas e utilizando recursos de som;

b) permitir um número maior de participantes no ambiente virtual, mostrando os tipos de ambientes virtuais que melhor se adequam a este tipo de sistema, utilizando melhor os recursos de rede e identificando seus pontos fortes e fracos;

c) melhorar o processo de comunicação entre os ambientes, bem como o controle do mesmo, fazendo uso de outros tipos de PDU que não foram utilizados no desenvolvimento desse protótipo;

d) utilizar o envio de PDU via *multicast*, permitindo acessar os mundos virtuais através da internet. Segundo pesquisas feitas em Web3D (2000a), as classes do protocolo DIS já permitem fazer o envio e recebimento de PDU através de endereçamento *multicast*;

e) implementar o processo de comunicação da classe que recebe os PDU através do protocolo DIS para a tela de comandos Java que é utilizada pelo usuário para interagir com o ambiente virtual. Maiores detalhes podem ser encontrados no anexo C deste trabalho.

# REFERÊNCIAS BIBLIOGRÁFICAS

AMES, Andrea L.; NADEAU, David R.; MORELAND, John L. **The VRML 2.0 sourcebook.** 2. ed. New York: John Wiley & Sons, 1997.

ASPETUS, Ida. **Tudo sobre aplicações de software empresarial**, [S.l.], 2001. Disponível em: <http://www.aspetus.com/Files/Glossario/Index_F_P_Glossario_E_0004.htm>. Acesso em: 13 mar. 2002.

BBCI. **World war one trench**, [S.l.], abr. 2002. Disponível em: <http://www.bbc.co.uk/history/3d/trench.shtml>. Acesso em: 02 abr. 2002.

BENFORD, Steve. *et al*. **User embodiment in collaborative virtual environments**, Denver, dez. 1995. Disponível em: <http://www.acm.org/sigchi/chi95/proceedings/papers/sdb_bdy.htm>. Acesso em: 15 mar. 2002.

BIT. **Realidade virtual**, [S.l.], ago. 1999. Disponível em: <http://www.stinet.com.br/bit07.html>. Acesso em: 01 mar. 2002.

BRUTZMAN, Don. **The Virtual Reality Modeling Language and Java**, California, jun. 1998. Disponível em: <http://www.web3d.org/WorkingGroups/vrtp/docs/vrmljava.pdf>. Acesso em: 10 abr. 2002.

COSMO SOFTWARE. **Cosmo player**, New York, 2000. Disponível em: <http://ca.com/cosmo/>. Acesso em: 28 mar. 2002.

EDUARDO, Vandeir. **Protótipo de um ambiente virtual distribuído multiusuário**. 2001. 104 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

FLANAGAN, David. **Java:** o guia essencial. Tradução Katia Roque. Rio de Janeiro: Campus, 2000.

GASPARY, Luciano P. **Multicast tutorial**, Porto Alegre, dez. 1996. Disponível em: <

http://penta.ufrgs.br/redes296/multicast/tutorial.html>. Acesso em: 13 mar. 2002.

GRADECKI, Joe. **Kit de montagem da realidade virtual.** Tradução Josue Vieira. São Paulo: Berkeley, 1994.

IEEE, Institute of Electrical and Electronics Engineers. IEEE Std 1278: Standard for information technology, protocols for distributed interactive simulation. [S.l], 1993.

IPOLITO, Juliano. **Tutorial VRML 1.0**, São Carlos, 1997. Disponível em: <http://www.realidadevirtual.com.br/publicacoes/tutorial_vrml/index.html>. Acesso em: 13 mar. 2002.

KAY, David; MUDER, Douglas. **VRML e 3-D na web para leigos.** Tradução de Pedro Conti. São Paulo: Berkeley, 1997.

KIRNER, Cláudio. **Realidade virtual:** dispositivos e aplicações, Marília, [1999?]. Disponível em: <http://www.realidadevirtual.com.br/publicacoes/apostila_rv_disp_aplicacoes/apostila_rv.html>. Acesso em: 02 mar. 2002.

KIRNER, Cláudio. **Sistemas de realidade virtual**, São Carlos, [2000?]. Disponível em: <http://www.dc.ufscar.br/~grv/tutrv/tutrv.htm>. Acesso em: 20 fev. 2002.

KUBO, Mario M. *et al*. Interação nos ambientes virtuais colaborativos de ensino. In: Workshop on Virtual Reality, 1., 2000, Gramado. **Anais**... Gramado: SBC, 2000. p. 55-64.

LOCKE, John. **An introduction to the internet networking environment and SIMNET/DIS**, [S.l.], ago. 1995. Disponível em: <http://www.web3d.org/WorkingGroups/dis-java-vrml/DISIntro.ps>. Acesso em: 06 abr. 2002.

MACEDONIA, Michael R.; ZYDA, Michael J. A taxonomy for networked virtual environments. **IEEE multimedia,** Los Alamitos, v. 4, n. 1, p. 48-56, jan./mar. 1997.

MACEDONIA, Michael. **A network software architecture for large scale virtual environments.** 1995. 200 p. Dissertação de doutorado (Doutor de filosofia em ciências da computação), Naval Postgraduate School, Monterey.

MACHADO, Liliane dos Santos. **Conceitos básicos da realidade virtual**, São José dos

Campos, nov. 2001. Disponível em: <http://www.lsi.usp.br/~liliane/conceitosrv.html>. Acesso em: 09 mar. 2002.

NADEAU, David R. Building virtual worlds with VRML. **IEEE computer graphics and applications,** Los Alamitos, v. 19, n. 2, p. 18-29, mar./abr. 1999.

PASQUALOTTI, Adriano; FREITAS, Carla M. D. S. Ambientes VRML para o ensino-aprendizagem de matemática: modelo conceitual e protótipo. In: Workshop on Virtual Reality, 1., 2000, Gramado. **Anais**... Gramado: SBC, 2000. p. 65-76.

PERUZZA, Ana Paula P. M. *et al.* Realidade virtual aplicada à recuperação do centro comercial de Marília. In: Workshop on Virtual Reality, 1., 2000, Gramado. **Anais**... Gramado: SBC, 2000. p. 3-11.

PINHO, Márcio S. *et al*. **Um modelo de interface para navegação em mundos virtuais,** Porto Alegre, 1999. Disponível em: <http://grv.inf.pucrs.br/Pagina/Publicacoes/Bike/Portugues/Bike.htm>. Acesso em: 25 fev. 2002.

PINHO, Márcio S. **Realidade virtual como ferramenta de informática na educação**, [S.l.], [1996?]. Disponível em: <http://grv.inf.pucrs.br/Pagina/Educa/educa.htm>. Acesso em: 09 mar. 2002.

PINHO, Márcio S. **VRML 2 – VRML97**, Porto Alegre, mai. 2001. Disponível em: <http://www.inf.pucrs.br/~pinho/CG/Aulas/Vrml/Vrml2/vrml2Pinho.htm>. Acesso em: 24 mar. 2002.

PINHO, Márcio. **Interação em ambientes tridimensionais**, Porto Alegre, 2000. Disponível em: <http://www.inf.pucrs.br/~pinho/3Dinteraction/>. Acesso em: 25 fev. 2002.

POLLO, Luis Fernando. **Software para a geração automática de modelos 3D em VRML**, Santa Maria, 1997. Disponível em: <http://www.inf.ufsm.br/~pollo/TG/>. Acesso em: 15 mar. 2002.

RAPOSO, Alberto B. **Interação na web – tendências**, Campinas, jul. 2000. Disponível em: <http://www.dca.fee.unicamp.br/courses/IA368F/1s1998/Monografias/alberto/index.html>. Acesso em: 13 mar. 2002.

SEIDMAN, Gregory. **VRML**: past, present, and future, [S.l.], nov. 2001. Disponível em: <http://zing.ncsl.nist.gov/~gseidman/class/vrml.html>. Acesso em: 21 mar. 2002.

SEMENTILLE, Antônio C. *et al*. Ambientes Virtuais usando CORBA: um Estudo de Caso. In: Workshop on Virtual Reality, 1., 2000, Gramado. **Anais**... Gramado: SBC, 2000. p. 145-156.

SINGHAL, Sandeep; ZYDA, Michael. **Networked virtual environments:** *design and implementation*. New York: Addison-Wesley, 1999.

SUN MICROSYSTEMS INC. **Getting start with the java3d API**, [S.l.], abr. 2002. Disponível em: <http://java.sun.com/products/java-media/3D/collateral/j3d_tut.zip>. Acesso em: 03 jun. 2002.

VR MATRIX. **3D art gallery**, [S.l.], [2001?]. Disponível em: <http://www.vr-matrix.com/3d-gallery/index.html>. Acesso em: 02 abr. 2002.

WEB3D, Web3D Consortium. **Distributed interactive simulation DIS-Java-VRML Working Group**, [S.l.], out. 2000a. Disponível em: <http://www.web3d.org/WorkingGroups/vrtp/dis-java-vrml/>. Acesso em: 28 fev. 2002.

WEB3D, Web3D Consortium. **Web3D Consortium FAQ**, [S.l.], out. 2000b. Disponível em: <http://www.web3d.org/fs_faq.htm>. Acesso em: 26 mar. 2002.

# ANEXO A: CÓDIGO FONTE DE UM EXEMPLO DA FUNCIONALIDADE JAVA E VRML

Conforme o exemplo apresentado no capítulo quatro, seção 4.2.5, este anexo contém o seu código fonte, apresentando o código da cena em VRML e o *script* utilizado pela mesma escrito na linguagem Java. O quadro 6.1 apresenta o código do arquivo *ScriptNodeEventOutControl.wrl* que representa a cena VRML. Nele são definidos os eventos, os nomes dos *nodes* e ligações entre a classe Java e a cena VRML.

Quadro 6.1 – Código fonte da cena VRML

```
#VRML V2.0 utf8

Group {
  children [

    DEF ClickTextToTest TouchSensor { }

    DEF InterfaceScriptNode Script {

      mustEvaluate  TRUE   # FALSE atrasa o envio de eventIns p/ script
      directOutput  FALSE  # TRUE para trabalho direto com field nodes
                           # FALSE somente envio de eventOuts p/ nodes

      eventIn   SFTime startTime          # evento chamado após o clique
                                          # sobre o texto
      eventOut  MFString ChangedText      # evento de mudança do texto
      eventOut  SFVec3f  ChangedPosition  # evento de mudança de posição

      # define a classe que será chamada ao carregar a cena VRML
      url    "ScriptNodeEventOutControl.class"

    }

    DEF TextPosition Transform {
      children [
        Shape {
          appearance  Appearance {
            material Material {
              diffuseColor .2 .9 .2
              shininess .9
            }
          }
          geometry DEF MessageToUser Text {
            string [  "Texto padrão apresentado na cena VRML,"
                 "que será substituído pelo método initialize do Java"
                 "via Script node usando controle EventOut."
                 ""
                 "Caso este texto não seja substituído,"
                 "Java via Script node não está funcionando..."
            ]
            fontStyle FontStyle {
```

```
                size 0.8
                justify [ "MIDDLE" ]
              }
            }
          }
        ]
      }
    ]
    # definindo ligações entre os nodes
    ROUTE ClickTextToTest.touchTime TO InterfaceScriptNode.startTime
    ROUTE InterfaceScriptNode.ChangedText TO MessageToUser.set_string
    ROUTE InterfaceScriptNode.ChangedPosition TO
          TextPosition.set_translation
}
```

Fonte: Adaptado de Brutzman (1998)

Já o quadro 6.2 apresenta o código do arquivo *ScriptNodeEventOutControl.java* que representa o *script* chamado durante a execução do exemplo citado.

Quadro 6.2 – Código fonte do *script* escrito na linguagem Java

```
// importando classes VRML
import vrml.*;
import vrml.field.*;
import vrml.node.*;

public class ScriptNodeEventOutControl extends Script
{

   // declarando eventos como no código VRML
   private vrml.field.SFTime startTime;         // eventIn
   private vrml.field.MFString  ChangedText;    // eventOut
   private vrml.field.SFVec3f   ChangedPosition; // eventOut

  // método chamado ao abrir a cena VRML
   public void initialize ()
   {

      // instanciando e buscando evento na cena VRML
      ChangedText = (MFString) getEventOut ("ChangedText");

      String [] message = new String [3];
      message [0] =  "Java ScriptNodeEventOutControl.class";
      message [1] = "inicializou o ChangedText node";
      message [2] = "e o texto está pronto para ser clicado...";

      // setando novo texto na cena VRML
      ChangedText.setValue ( message );

      // instanciando e buscando evento na cena VRML
      ChangedPosition = (SFVec3f) getEventOut ("ChangedPosition");
      SFVec3f position = new SFVec3f ( 0, 3, 0 );
      // setando nova posição do texto
      ChangedPosition.setValue ( position );

      return;
   }
```

```
    // método chamado pela cena VRML quando o usuário clicar sobre o texto
    public void processEvent (Event touch)
    {
      String [] message = new String [4];
      message [0] = "Foi chamado o método processEvent";
      message [1] = "via Script node eventIn.";
      message [2] = "Texto e posição foram alterados com sucesso";
      message [3] = "via controle eventOut.";
      // modifica o texto após o clique sobre o texto
      ChangedText.setValue ( message );

      SFVec3f position = new SFVec3f ( 0, -1, 0 );
      // modifica a posição do texto após o clique sobre o texto
      ChangedPosition.setValue ( position );

      return;
    }
  }
```

Fonte: Adaptado de Brutzman (1998)

# ANEXO B: PASSOS A SEREM SEGUIDOS PARA A EXECUÇÃO DO PROTÓTIPO

Para que seja possível executar o protótipo é necessário que os seguintes passos sejam seguidos:

a) ter instalado o arquivo do tipo **`jar`** correspondente à tecnologia DIS-Java-VRML no subdiretório **`$JAVA_HOME\jre\lib\ext`**. Acesse o endereço http://www.web3d.org/WorkingGroups/vrtp/dis-java-vrml/download.html e baixe o arquivo **`dis-java-vrml.tar.gz`** ou **`dis-java-vrml.zip`**, descompacte-o num novo diretório qualquer e mova o arquivo **`dis-java-vrml.jar`**, que está nesse diretório onde foi descompactado, para o subdiretório **`$JAVA_HOME\jre\lib\ext`**;

b) ter instalado o programa Netscape Communicator na versão 4.77 que pode ser encontrado acessando a *home page* da Netscape, no endereço http://www.netscape.com/download/. Segundo Web3D (2000a), testes relacionados ao Microsoft Internet Explorer ainda estão sendo feitos pelo grupo de trabalho do DIS-Java-VRML;

c) ter instalado o *plugin* Cosmo Player, que permite a visualização de ambientes virtuais construídos através da linguagem VRML. Sua instalação pode ser obtida na *home page* da Cosmo Software, no endereço http://ca.com/cosmo/. Após a instalação é importante que se inclua o caminho **`$NETSCAPE_HOME\Netscape\Communicator\Program\Plugins\NPCOSMOP211.Jar`** na variável **`CLASSPATH`** do sistema operacional em uso;

d) ter instalado a plataforma de desenvolvimento Java, no caso o JSDK versão 1.3, ou no mínimo, o ambiente de execução Java correspondente a essa versão (JRE 1.3). Pode-se obter a instalação na *home page* da Sun, no endereço http://java.sun.com/products/. Em "*Product Shortcuts*" selecione a versão desejada. É importante que o caminho **`$JAVA_HOME\bin`** esteja incluído na variável **`PATH`** do sistema operacional em uso;

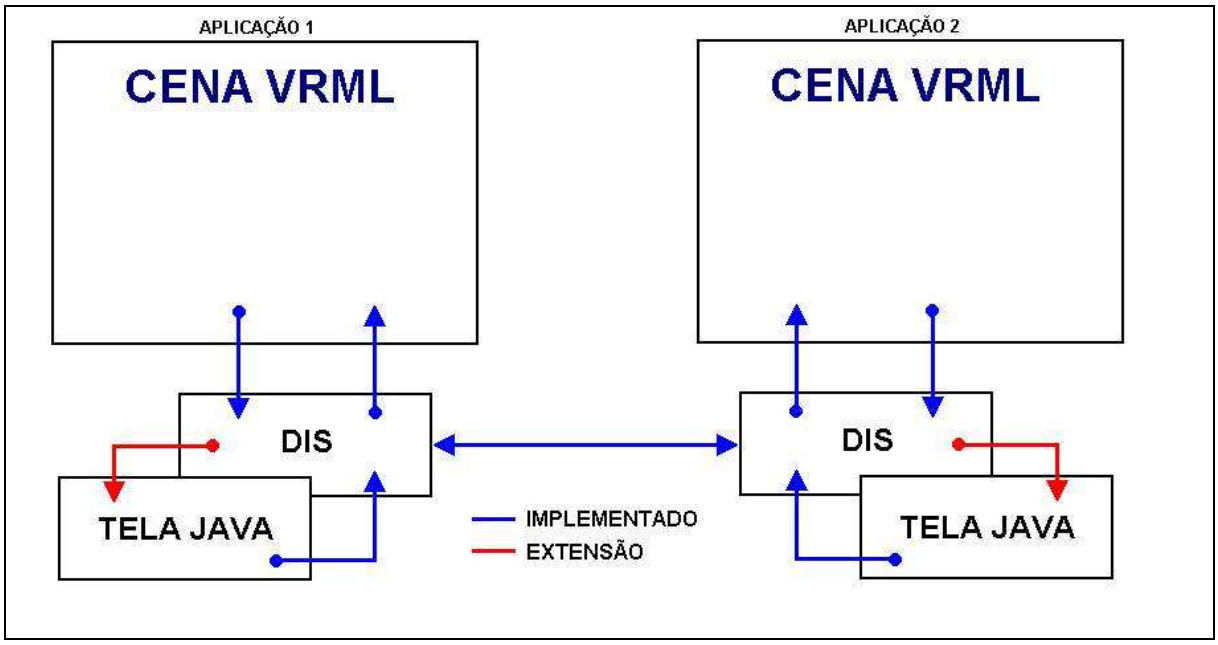
e) Depois disso, basta acessar o diretório onde estão as classes correspondentes ao protótipo implementado nesse trabalho, e através do Netscape Communicator abrir

o arquivo **`ScriptAmbienteVirtual.wrl`**, o qual, corresponde ao mundo virtual e seus personagens. Por último, deve-se digitar a linha de comando **`java AmbienteVirtual`** para que a janela com as opções de comandos seja aberta.

# ANEXO C: FUNÇÕES IMPLEMENTADAS E EXTENSÕES

Este anexo apresenta as principais funções desenvolvidas no protótipo deste trabalho, e também, as funções que poderão ser implementadas em trabalhos de extensão por não terem sido criadas ainda. A fig. 6.1 ilustra as funções implementadas e não implementadas durante o desenvolvimento deste trabalho, sendo as mesmas, representadas por setas azuis e vermelhas respectivamente.

Figura 6.1 – Representação das funções implementadas e não implementadas no protótipo




As funções implementadas durante o desenvolvimento do protótipo possibilitaram atingir os objetivos propostos neste trabalho, permitindo a interação do usuário com o ambiente virtual através de uma tela de comandos desenvolvida em Java, a qual, envia informações para uma classe que realiza a comunicação entre os mundos virtuais participantes, incluindo a si próprio, através do protocolo DIS.

Por outro lado, não foi implementada a comunicação de retorno entre a classe que realiza as funções do protocolo DIS e a tela de comandos Java impossibilitando a atualização, quando necessário, das informações exibidas na tela dos usuários participantes do mundo virtual. Foram encontrados problemas em relação ao desenvolvimento desta função pois, ao ser implementada, ocorreram problemas relacionados ao envio de PDU para os mundos

virtuais participantes, fato que comprometia os objetivos propostos neste trabalho. Portanto, conforme citado na seção 6.1, fica como sugestão para trabalhos de extensão futuros descobrir alguma forma de implementar esta função sem comprometer o funcionamento dos processos que foram implementados e estão funcionando. Sugere-se também que, juntamente com a implementação desta função, seja acrescentada ao diagrama de classes, apresentado na seção 5.1.3, uma nova classe representando os personagens e objetos do ambiente virtual.

# ANEXO D: DIAGRAMA DE CLASSES COMPLETO

Este anexo, conforme apresentado na fig. 6.2, apresenta o diagrama de classes com todos os atributos e métodos implementados no protótipo deste trabalho.

Figura 6.2 – Diagrama de Classes com todos os atributos e métodos