Implementação de um Ambiente Virtual Distribuído Baseado em Java e VRML

UFRJ

Projeto submetido para a obtenção do título de

Engenheiro Eletrônico

ao Departamento de Eletrônica

da Escola de Engenharia da UFRJ

por

Artur Ziviani

Implementação de um
Ambiente Virtual Distribuído
Baseado em Java e VRML

Autor:

Artur Ziviani

Orientador:

Prof. Otto Carlos Muniz Bandeira Duarte, Dr.Ing.

Examinadores:

Prof. José Ferreira de Rezende, Dr.

Marcelo Dias de Amorim, M. Sc.

Departamento de Eletrônica - Escola de Engenharia - UFRJ

RIO DE JANEIRO, RJ - BRASIL

MARÇO DE 1998

Aos meus pais.

Agradecimentos

Aos meus pais, Cilio e Angela, e demais familiares pelo apoio e incentivo ao longo de toda a minha vida.

Ao professor Otto Carlos Muniz Bandeira Duarte, meu orientador.

Aos amigos Falcão, Ana Paula, Júnior e Shunsuke pelos bons momentos compartilhados ao longo de anos de convívio. Também agradeço à Karla pelo apoio e carinho, e aos amigos Alexandre Bento e Katia pela força e incentivo.

Aos colegas formandos do segundo período de 1997, especialmente ao Paulo André e Alexandre, companheiros de estudo e inúmeros trabalhos de grupo durante o curso.

Aos professores do Departamento de Eletrônica da Escola de Engenharia da UFRJ, especialmente aos professores Antônio Cláudio, Aloysio, Casé, Luiz Wagner, Mauros, Moreirão, Otto e Sérgio Palma. Agradeço também à Dona Conceição pela presteza no atendimento na secretaria do Departamento.

Aos colegas, membros do Grupo de Teleinformática e Automação (GTA), que incentivaram o trabalho, estiveram sempre prontos a ajudar e a esclarecer quaisquer dúvidas que surgissem, das mais simples às mais complexas. São eles Antônio, Baiano (obrigado pelas revisões), Belem, Bill, Daniel, Emerson, Flávio, Fred, Hermes, Jonny, Luis Henrique, Renata (obrigado pelas sugestões), Ronaldo, Rubi, Sílvia e Vidal.

Resumo do Projeto Final apresentado ao DEL/UFRJ como parte dos requisitos necessários para a obtenção do grau de Engenheiro Eletrônico.

Implementação de um Ambiente Virtual Distribuído Baseado em Java e VRML

Artur Ziviani

Março de 1998

Orientador: Otto Carlos Muniz Bandeira Duarte
Departamento: Engenharia Eletrônica

Ambientes virtuais despontam potencialmente como a futura interface homem-máquina a ser utilizada, suplantando os sistemas multimídias amplamente adotados nos dias atuais. Neste trabalho discute-se a utilização combinada das linguagens VRML (*Virtual Reality Modeling Language*) e Java para a criação de um ambiente virtual que permita a um número limitado de usuários compartilhar um cenário virtual. Todos têm sua visualização e percepção da presença dos outros participantes mantidas atualizadas enquanto movem-se pelo mundo virtual criado. Os participantes podem trocar mensagens entre si, favorecendo o trabalho cooperativo entre eles. A aplicação desenvolvida no laboratório do Grupo de Teleinformática e Automação (GTA) da UFRJ busca demonstrar a potencialidade da aplicação desta abordagem. Todos os passos realizados para o desenvolvimento dos componentes do sistema são descritos. As vantagens principais do sistema implementado são a simplicidade e a amigabilidade da interface oferecida ao usuário.

Palavras Chave

Redes de computadores

VRML

Java

Realidade virtual

Interatividade

Sumário

Lista de Figuras

LISTA DE FIGURAS

Lista de Tabelas

Capítulo I

Introdução

Os avanços alcançados na indústria de computadores nos últimos anos têm consolidado os sistemas multimídias como forma de interface homem-máquina amplamente disponível nos dias atuais. A busca por uma interação mais simples, direta e natural entre os usuários e os seus computadores vem indicando a tecnologia de realidade virtual (RV) como a forma mais adequada de chegar a estes objetivos.

Com o crescimento vertiginoso experimentado pela Internet nos últimos anos, o interesse por realidade virtual distribuída também aumentou devido ao grande potencial de aplicações que podem utilizar suas técnicas, abrangendo áreas que vão desde o entretenimento até aplicações de medicina ou simulação militar. Realidade virtual é também referenciada indistintamente pela literatura como ambiente virtual, mundo virtual ou *cyberspace*.

A linguagem VRML (*Virtual Reality Modeling Language*) vem firmando-se como padrão para a descrição de realidade virtual na Internet. VRML contorna suas limitações inerentes ao fato de ser uma linguagem simplesmente descritiva, com a possibilidade de criação de uma interface com outras linguagens de programação. A linguagem Java da Sun Microsystems é amplamente utilizada neste caso, devido ao seu caráter portável, compatível com a proposta de portabilidade de VRML.

Este trabalho, desenvolvido no Grupo de Teleinformática e Automação (GTA) da

UFRJ, visa um ambiente virtual distribuído de forma a tornar claro o potencial deste tipo de aplicação em áreas como trabalho cooperativo, treinamento, simulação, entretenimento, entre outras. O seu desenvolvimento baseou-se na utilização combinada de VRML e Java, onde VRML ficou responsável pela descrição dos objetos tridimensionais que formam o ambiente virtual e Java pela interface gráfica com o usuário, comunicação dos clientes com o servidor e implementação do próprio programa servidor.

I.1 Conceitos Básicos de Realidade Virtual

A tecnologia de realidade virtual apresenta-se como a forma mais avançada de interação homem-máquina. O ambiente de realidade virtual consiste em um espaço tridimensional sintético, gerado por computador, no qual o usuário é capaz de visualizar e manipular os componentes do ambiente, criando assim um grau de interatividade com o cenário apresentado.

Três idéias básicas caracterizam um ambiente de realidade virtual: imersão, interação e envolvimento [1]. A idéia de imersão está relacionada com a sensação que o usuário tem de estar realmente inserido no ambiente virtual, possuindo a ilusão que os objetos dispostos no cenário estão realmente à sua volta. Interação está ligada à capacidade do ambiente virtual detectar ações do usuário e reagir conforme estas ações. Por sua vez, a idéia de envolvimento exprime o grau de motivação do usuário em engajar-se na atividade proposta pelo ambiente virtual. Este envolvimento pode ser passivo, onde o participante apenas explora o mundo virtual, ou ativo, onde há a possibilidade de interação com outros participantes em um mundo virtual compartilhado.

Uma maior sensação de imersão pode ser obtida através do uso de dispositivos não-convencionais para a apresentação do cenário ao usuário. O *head mounted display* ou HMD é um exemplo típico de dispositivo não-convencional que objetiva aumentar a sensação de imersão do usuário no ambiente virtual. Outros dispositivos não-convencionais captam algumas ações do usuário sobre o ambiente virtual no qual está inserido. Um exemplo deste tipo de dispositivo é o conhecido como *Powerglove*, uma luva especial que repassa os movimentos de uma mão e seus dedos para que estes exerçam alguma atividade do espaço virtual. Na Figura I.1, estão apresentados um modelo de HMD e outro de uma *Powerglove*.

Figura I.1: Head mounted display (HMD) e Powerglove.

Baseado no conceito de imersão, pode-se definir diversos níveis de imersão em um mundo virtual. Em realidade virtual completamente imersiva, o usuário traja alguns dispositivos não-convencionais e tem uma grande sensação de imersão, pois os seus movimentos físicos agem diretamente no ambiente virtual, havendo muitas vezes a perda da noção de espaço do mundo real exterior. O custo elevado dos dispositivos não-convencionais torna bastante restrito o uso de realidade virtual completamente imersiva. Pode-se utilizar apenas um monitor comum de computador para apresentação do cenário virtual e dispositivos convencionais de entrada, como teclado e mouse, para a captação das ações do usuário. A qualidade da apresentação tridimensional e a utilização de som dentro de um ambiente virtual contribuem decisivamente para a sensação de imersão do usuário que utiliza apenas dispositivos convencionais.

Sistemas que utilizam dispositivos ou abordagens não-convencionais (como RV em diferentes níveis) para a relação do usuário com o ambiente, real ou imaginário, podem ser classificados em quatro tipos distintos:

- Sistemas de Telepresença - possibilita que o usuário esteja virtualmente presente em um ambiente real geograficamente distante. Por exemplo, controlando as ações de um robô a distância em um ambiente remoto modelado em realidade virtual;
- Sistemas de Realidade Virtual - neste sistema o usuário participa em um ambiente

virtual gerado por computador, onde este ambiente pode imitar um ambiente real, ser um ambiente totalmente imaginário ou uma combinação de ambos;

- Sistemas de Realidade Ampliada [2, 3] - a cena visualizada pelo usuário neste sistema mescla visões do ambiente real com do ambiente virtual. Com o uso de capacetes especiais, normalmente são sobrepostas à imagem do ambiente real informações fornecidas pelo ambiente virtual. Alternativamente, pode-se coletar a imagem do ambiente real com uma câmera de vídeo e misturar as imagens geradas com a imagem virtual antes da exibição ao usuário. Por exemplo, pode-se ter a imagem real de um equipamento e a visão virtual de seu interior. O ponto crítico neste tipo de sistema é a acurácia na monitoração da posição do usuário para a superposição exata dos objetos do mundo virtual com as imagens do mundo real;

- Sistemas de Realidade Melhorada - semelhante ao sistema de realidade ampliada, aqui um sistema de processamento de imagem acrescenta informações superpostas às imagens do mundo real. Anotações podem ser acrescentadas às imagens, como distância ao objeto real, tipo do objeto ou instruções de manuseio.

I.2 Aplicações de Realidade Virtual

Realidade virtual vem encontrando aplicações em diversas áreas. Em medicina [4] surgem várias aplicações possíveis. O estudo de anatomia pode ser incrementado com os diversos órgãos podendo ser visualizados tridimensionalmente e à distância pela rede, facilitando a consulta por estudantes e médicos. O planejamento e treinamento médico para situações de emergência, como grandes desastres ou combates, podem ser obtidos por simulação. É possível a prototipação de um hospital, estudando a funcionalidade da distribuição espacial dos diversos setores e equipamentos também através de simulação. Na área cirúrgica, realidade melhorada pode ser aplicada com o uso de óculos ou capacetes com lentes de cristal líquido pelo cirurgião, onde os resultados dos exames prévios do paciente poderiam ser sobrepostos às imagens reais do paciente na mesa, conforme conveniência do médico. Tratamentos por exposição virtual para diversas fobias comuns, como medo de altura, de voar, de falar em público, de dirigir ou de grandes espaços abertos, estão sendo pesquisados [5].

Treinamento de profissionais de diversas áreas através de simulação por RV [6] vem crescendo nos últimos anos. Exemplos podem ser encontrados em treinamento de direção [7], de vôo, de astronautas etc. Simulações militares [8] também encontram uso para a tecnologia da realidade virtual, podendo exercitar estratégias de combate na movimentação de tropas e veículos ou capacitação individual dos soldados.

A área educacional pode obter avanços com a criação de animações em realidade virtual que facilitem a visualização e conseqüente entendimento por estudantes dos fenômenos estudados. Um uso semelhante pode ser empregado na área de visualização científica [9].

Ambientes virtuais podem ser criados tanto para a simulação da funcionalidade de futuras construções, aplicações basicamente de arquitetura e engenharia civil, como a reconstrução para estudo de antigos monumentos ou prédios, aplicação típica de arqueologia. Em ambos os casos, a possibilidade de trabalho cooperativo entre os participantes é a principal vantagem da aplicação de ambientes virtuais.

Um mercado potencial é a área de entretenimento, onde jogos ou simulações possuem o único objetivo de divertir o usuário.

I.3 Organização do Texto

Este projeto está organizado da seguinte forma. O Capítulo II trata das características de ambientes virtuais distribuídos, descrevendo os principais empecilhos ao desenvolvimento de sistemas de realidade virtual distribuída e as principais propostas de solução pesquisadas.

No Capítulo III são apresentas a linguagem VRML, a linguagem Java e as formas possíveis de combiná-las de forma a se alcançar maior interatividade em ambientes virtuais descritos em VRML e ter acesso à rede através da linguagem Java.

O Capítulo IV é destinado à descrição da implementação realizada, um ambiente virtual que suporta um número limitado de usuários com possibilidade de troca de mensagens entre si. São apresentados o mundo virtual, o *software* cliente, o programa servidor, o mecanismo para gerenciamento de usuários e atualizações das ações dos diversos participantes.

Finalmente, o Capítulo V traz as conclusões a respeito do trabalho desenvolvido e as propostas de trabalhos futuros.

Capítulo II

Ambientes Virtuais Distribuídos

Vistos como um meio de comunicação, ambientes virtuais [10] apresentam um grande potencial de aplicação em diversas áreas. O principal objetivo de um ambiente virtual é motivar o usuário a acreditar que ele está imerso no cenário sintético apresentado. Para alcançar esta ilusão são utilizadas técnicas de realidade virtual.

II.1 Realidade Virtual Distribuída

Um ambiente virtual mostra a representação tridimensional sintética de um espaço real, imaginário ou de uma combinação de ambos. Caso o ambiente suporte o acesso simultâneo por vários usuários e permita a interação destes entre si e também com o ambiente apresentado, ele é chamado de ambiente virtual distribuído (*distributed virtual environment* - DVE) [11, 12].

Os usuários são representados em um ambiente virtual por uma entidade conhecida como **avatar**. O avatar é a forma tridimensional pela qual o usuário será visualizado pelos outros usuários inseridos no mesmo ambiente virtual compartilhado. Ou seja, as ações tomadas pelo usuário serão executadas no ambiente virtual pelo seu avatar.

II.1.1 Desenvolvimento de Ambientes Virtuais Distribuídos

Há pelo menos duas décadas existem pesquisas em DVEs. Porém, poucos tinham acesso a computadores capazes de executar com eficiência o processamento demandado por um sistema de DVE e menos pessoas ainda tinham acesso a conexões de rede de computadores de longa distância com a grande largura de banda passante e a baixa latência exigida pelos DVEs. No entanto, nos últimos anos, a capacidade computacional dos processadores vem aumentando gradativamente enquanto seu custo reduz-se. Os modens caseiros estão alcançando taxas de transmissão que atendem as necessidades de um sistema DVE simples. Mundos virtuais multi-usuários estão se tornando amplamente acessíveis [13].

As pesquisas em DVEs estão sendo conduzidas basicamente por duas comunidades distintas com prioridades diferentes. Elas são o mundo da Internet, apoiada pelos desenvolvedores comerciais e o mundo da Simulação Distribuída Interativa (*Distributed Interactive Simulation* - DIS), suportado pelos desenvolvedores de simulação militar. O objetivo final de ambos é semelhante: um ambiente virtual de interação com alto grau de complexidade e flexibilidade que suporte simultaneamente um grande número de usuários. Os caminhos escolhidos por ambos, no entanto, divergiram. O grupo da Internet prioriza a acessibilidade, enquanto o grupo do DIS baseia-se em equipamentos dedicados de alto custo.

Abordagem da Internet

O mundo da Internet defende que um ambiente virtual distribuído deve poder ser executado pelos modelos de computador e acessado pelo tipo de conexão que a maioria dos usuários comuns possui. Surgem problemas como a limitação de banda passante e a latência nas conexões caseiras.

O início dos trabalhos em DVE na Internet ocorreu em meados da década de 70, com o surgimento de um novo tipo de jogo para computador, o *Adventure*, criado por Will Crowther e Donald Woods do centro de pesquisa da Xerox em Palo Alto, Califórnia. Este jogo era executado em computador de grande porte PDP-10 da Digital Equipment Corp. Este jogo de computador foi inspirado no *Dungeon and Dragons* (D&D), um *role playing game* (RPG) criado por Dave Arneson, da TSR Inc., em 1973.

Em D&D, como em outros RPGs, os participantes utilizam apenas lápis, papel e um conjunto de regras pré-definidas. Toda a descrição do mundo é feita por um dos jogadores, chamado de *dungeon master* (DM). Os demais jogadores interpretam personagens envolvidos em aventuras no mundo virtual descrito pela narrativa do DM. O conjunto de regras serve para resolver conflitos entre os personagens ou dos personagens com os obstáculos colocados pelo narrador da história. O ambiente virtual está na imaginação de cada participante.

Adventure, no entanto, era um jogo para um único jogador. O jogo fornecia a descrição em forma de texto do que estava sendo visto pelo participante e cabia a este formar a imagem do ambiente baseado na descrição apresentada. Com o passar de alguns anos, o *Adventure* tornou-se um gênero de jogos de computador, populares até hoje. A evolução chave deste gênero de jogo, ao menos da perspectiva dos DVEs, foi a possibilidade de múltiplos jogadores interagirem. O primeiro destes jogos multi-usuários foi o *Multiuser Dungeon*, conhecido como MUD, criado por Roy Trubshaw e Richard Bartle da Universidade de Essex, Inglaterra, em 1980. O MUD também é todo baseado em texto. Devido a popularidade alcançada pelo MUD e os seus sucessores, o termo MUD muitas vezes é utilizado como referência a jogos de computador multi-usuário. Do desenvolvimento posterior do MUD surgiram as salas de conversa (*chat rooms*) da Internet. Outra linha de desenvolvimento, utilizando tecnologia de programação orientada a objetos, permitiu ao criador do mundo virtual construir um módulo básico, facilmente modificável e expansível. Esta vertente do MUD ficou conhecida como MOO (MUD orientado a objetos).

Certamente, os usuários e desenvolvedores de MUD sempre sonharam com a possibilidade de visualizar e ouvir o mundo virtual originalmente descrito apenas por texto. Com a popularização dos computadores pessoais e a maior facilidade de acesso à rede de computadores, o mundo da Internet pode experimentar a construção de ambientes virtuais com imersão tridimensional e interação de áudio. Alguns esforços foram concentrados em sistemas proprietários, mas também houve um forte movimento defendendo a criação de sistemas abertos. O principal resultados deste movimento foi o surgimento do padrão VRML (*Virtual Reality Modeling Language*) discutido no Capítulo III.

Abordagem Militar

O mundo do DIS busca simulações altamente realistas para serem utilizadas em treinamento militar. De um sistema DIS é esperada a capacidade de suportar milhares de usuários simultâneos imergindo-os em um ambiente tridimensional satisfatoriamente convincente. Um sistema como este, tipicamente, exige *hardware* caro e dedicado [8].

Desde de a década de 70, o departamento de defesa (DoD) do governo americano possui simuladores imersivos para o treinamento individual dos comandados na operação de diversos veículos militares. A partir da década de 80, o DoD financiou pesquisas para conectar aparelhos de simulação através de redes de alta velocidade dedicadas. Esta rede ficou conhecida como Simnet. Decidiu-se usar os simuladores existentes, que possuiam formatos incompatíveis (por exemplo, para a modelagem tridimensional) e fazê-los funcionar em conjunto, ao invés de desenvolver padrões para conteúdo portável. Partindo-se desta decisão, chegou-se ao padrão IEEE DIS, cujo enfoque está em como os simuladores envolvidos podem comunicar informações dinâmicas como a posição de objetos que retratem tropas ou veículos virtuais. O padrão DIS serviu como base para toda uma indústria de simulação e sistemas de treinamento militar.

O DIS sofre ainda hoje pela falta de padronização em conteúdo portável, dificultando a sua extensibilidade. Ou seja, caso se deseje acrescentar um novo modelo de tanque de guerra, por exemplo, os fabricantes de cada modelo de simulador envolvido precisam desenvolver o modelo tridimensional do novo objeto no seu *software* proprietário.

Esforços estão sendo realizados para o desenvolvimento de uma arquitetura de alto nível que permita a reutilização do *software* dos simuladores. Esta abordagem visa reduzir custos de aquisição e manutenção de sistemas.

II.1.2 Considerações sobre Ambientes Virtuais Distribuídos

Ambientes virtuais distribuídos possuem objetivos de desempenho extremamente ambiciosos, dentre eles o projeto de interfaces naturais em tempo real para ambientes fisicamente corretos [14]. A fidelidade na representação do ambiente virtual não necessita de um realismo fotográfico, porém deve acuradamente exprimir os objetos do mundo real retratados.

O desenvolvimento de ambientes virtuais distribuídos encontra grandes desafios tecnológicos, tais como o gerenciamento da grande quantidade de dados envolvida, a interação entre os vários objetos envolvidos e a escolha da topologia de rede adequada.

Os objetos virtuais inseridos no mundo que possuam qualquer tipo de estado mutável no tempo são chamados de **entidades**, ou se controlados por um usuário, **avatares**.

A tarefa de um sistema DVE é intermediar a troca de informações sobre o estado dinâmico das entidades entre as estações da rede. A mudança de estado de um avatar, comandada por um participante, deve ser comunicada a todas as outra estações da rede. Através desta atualização de dados é possível manter a consistência na visualização do mundo virtual por todos os participantes. Logo, os pacotes de informação trocados entre as estações envolvidas em um sistema de DVE tipicamente retratam o estado atual de entidades.

Os desafios básicos na implementação de um sistema DVE estão na decisão de como lidar com a latência alta e variável, e com a largura de banda limitada nas comunicações de rede. O sistema deve ainda assim manter a consistência do mundo virtual para todos os seus participantes [15].

Um abordagem inicial ao compartilhamento de informações sobre o estado de entidades levaria a cada entidade enviar continuamente informações sobre o seu estado a todas as outras estações na rede. O problema aparece quando o número de usuários cresce e o mundo virtual torna-se mais complexo, aumentando o volume de dados trocados entre as estações envolvidas até o ponto onde a banda passante disponível torna-se o fator limitante. Muitas vezes os participantes recebem pacotes com informações para si irrelevantes, gerando processamento desnecessário.

O problema de banda passante limitada para o potencial grande volume de dados gera outro problema: a latência. Latência pode ser definida como o período de tempo entre a mundança de estado de uma entidade em uma estação da rede e a efetiva alteração de seu estado em uma outra estação. Quando um usuário deseja que seu avatar desloque-se por um ambiente virtual, existe um atraso para o *software* local detectar este comando. Outro atraso é gerado quando esta instrução é convertida em informação apropriada à transmissão pela rede. Outro montante de tempo é gasto para a efetiva transmissão da informação, principalmente se a rede utilizada for do tipo longa distância (WAN - *Wide Area Network*). Finalmente, um tempo adicional é necessário para que na recepção do pacote o destinatário possa converter as informações contidas efetivamente na visualização tridimensional do cenário virtual. Caso o tempo total seja excessivamente grande, haverá um longo período de espera entre, por exemplo, um usuário instruir seu avatar para mover-se e este realmente deslocar-se à vista dos demais participantes do mundo virtual.

Uma alta latência é prejudicial à qualidade de um sistema DVE, assim como uma latência altamente variável. Atualizações de estado em usuários diferentes com intervalos variáveis de tempo podem gerar problemas de consistência ao mundo virtual. Por exemplo, se um usuário A aciona um recurso comum disponível no ambiente virtual e um usuário B tenta o mesmo simultaneamente, quando um receber a informação sobre a ação do outro é possível que tenha decorrido um atraso excessivo. O mundo virtual poderá ter perdido sua consistência.

Idealmente, a banda passante seria ilimitada e a latência desprezível. Entretanto, a Internet atual não é adequada à interação em tempo real. Os pacotes sofrem atrasos na entrega e esta latência é extremamente variável e imprevisível. Por isto algumas estratégias foram pesquisadas para lidar com o problema de banda passante limitada e atraso alto ou variável.

Os problemas de banda passante limitada e latência são obstáculos para o desenvolvimento de ambientes virtuais distribuídos, principalmente aqueles com exigência de escalabilidade. Logo, um ambiente virtual deve considerar as limitações impostas pela rede de comunicação de forma a ser escalável, possibilitando um grande número de participantes [16, 17].

Compressão de Dados

O uso de compressão de dados diminui a exigência de banda passante no envio de dados. Em compressão de vídeo, por exemplo, usa-se o fato que, em média, os quadros mudam muito pouco em relação aos seus antecessores imediatos, sendo possível enviar somente as diferenças entre os quadros. A mesma idéia pode ser estendida a sistemas DVE. O envio constante de atualização de posição em um ambiente virtual distribuído, por exemplo, pode gerar um desperdício de banda passante. A idéia é aproveitar-se do quão similar o estado de uma entidade está de seu estado anterior.

Um objeto que movimenta-se por um ambiente virtual distribuído pode enviar a sua posição atual e um vetor de velocidade, indicando assim o sentido de seu deslocamento e sua velocidade neste deslocamento. Uma nova atualização somente seria necessária caso o objeto altere o sentido ou a magnitude de seu deslocamento, ao invés de enviar diversas posições intermediárias continuamente. O *software* em cada estação pode deduzir a posição instantânea do objeto pela sua posição inicial e seu vetor de velocidade. Este conceito ficou conhecido como *dead reckoning* e foi introduzido no início dos anos 80 no sistema Simnet das forças armadas americanas [18]. O nome tem origem na navegação marítma e sugere a obtenção da posição atual a partir da posição e velocidade anteriormente conhecidas. O protocolo DIS (*Distributed Interactive Simulation*) derivado do sistema Simnet transmite o estado de seus objetos fornecendo sua posição, sua velocidade e sua velocidade angular. De fato, cada simulador calcula a sua posição de forma detalhada, através de um algoritmo que leva em conta a sua posição e velocidade anteriores, a entrada do usuário e outros dados relevantes. Outro cálculo mais simples é realizado levando-se em conta somente os dados enviados aos outros simuladores. Quando os dois resultados divergem por um determinado valor limite, o resultado do cálculo mais detalhado é enviado para os demais simuladores. Mesmo que não haja qualquer divergência, cada entidade envia, em intervalos regulares de tempo, atualizações de seu próprio estado apenas para que o sistema identifique que elas ainda estão em operação. Estas atualizações são conhecidas como *heartbeats* ou *keepalives*.

O conceito de *dead reckoning* economiza banda passante e torna o sistema menos vulnerável à latência. Como cada mensagem de atualização também leva uma estampa de tempo (*timestamp*) registrada, mesmo que o pacote atrase é possível calcular a posição atual do objeto em questão.

A técnica de *dead reckoning* foi originalmente idealizada para aplicação em sistemas de simulação militar, onde o número de objetos diferentes envolvidos é relativamente pequeno e seus modelos tridimensionais estão replicados em todos os simuladores envolvidos. Basta, portanto, que sejam enviadas informações de posição e deslocamento entre os simuladores. Porém a idéia do *dead reckoning* pode ser levada para sistemas que possuam objetos mais complexos. Assim, tratando com objetos que tenham articulações que modelem seres humanos, por exemplo, os pacotes poderiam enviar informações adicionais às de posição e deslocamento, como pule, ande ou acene.

Filtragem de Dados

Em um ambiente virtual compartilhado, normalmente, nem todos os usuários interagem entre si simultaneamente. Técnicas de filtragem de dados podem aproveitar-se do fato de que um determinado usuário não necessite naquele instante das informações de um outro participante. Procura-se minimizar o envio de informações irrelevantes.

De forma geral, pode-se agrupar as diversas entidades em classes com relação à área de interesse de cada entidade. Estas classes podem ser de três tipos: espacial, temporal ou funcional [19]. As atualizações de uma entidade somente são enviadas para os membros do mesmo grupo em cada classe. Um grupo da classe espacial congrega entidades que compartilhem o mesmo espaço no ambiente virtual. Os membros de um grupo da classe temporal necessitam de taxas de atualização similares. Por exemplo, em um sistema militar de simulação, aviões em combate aéreo necessitam de um controle apenas superficial da posição de veículos terrestres. Portanto, podem receber atualizações de um determinado veículo a uma taxa menor do que outros veículos terrestres estão recebendo. Desta forma, os aviões envolvidos no combate aéreo estão em uma classe temporal e os veículos terrestres em outra. Caso este mesmo avião passe a realizar um ataque aos veículos terrestres ele seria inserido na mesma classe temporal destes. Finalmente, um grupo da classe funcional contém entidades que possuam funcionalidades semelhantes. Em uma simulação militar, por exemplo, uma classe funcional envolveria um grupo tático de controle aéreo.

A filtragem de dados mais comum envolve a classe espacial. As entidades inseridas em um mesmo espaço virtual freqüentemente não estão na área de interesse de outras entidades. O espaço virtual pode ser dividido em espaços abertos, como cidades, florestas, áreas ao ar livre em geral, ou confinados, como salas, prédios ou áreas fechadas. Pode-se utilizar a filtragem por oclusão para espaços confinados. Caso exista um obstáculo entre duas entidades, estas não necessitam receber informações uma sobre a outra. Por exemplo, se o usuário A estiver movimentando-se no interior de uma sala, o usuário B, que esteja na sala ao lado, não necessita receber os dados relativos à movimentação de A, pois a visão de B limita-se ao interior de sua sala. O mesmo ocorre com A em relação a B. Espaços abertos podem sofrer filtragem por oclusão ou por distância. A filtragem por oclusão, neste caso, envolveria o interior e exterior de um espaço confinado. Para a filtragem por distância é necessário estabelecer uma alcance para a visão das entidades, ou seja, a partir de uma determinada distância em um espaço aberto as entidades estão fora do alcance da vista da entidade e não faz-se necessária a atualização de seus dados.

O escopo de interação do participante em um mundo virtual é delimitado pela filtragem de dados aplicada, diminuindo-se o volume de mensagens trocada, pois apenas os participantes de um mesmo grupo trocam informações.

Buscando minimizar o tráfego na rede, muitos sistemas DVE estão adotando a comunicação *multicast*. Nesta configuração os pacotes não são endereçados a uma única estação, mas a um grupo de estações. Cada grupo *multicast* define uma área de interesse para cada entidade e equivale a um grupo formado com base nas classes espacial, temporal ou funcional.

Manutenção de Consistência

Um grande desafio para um sistema DVE é a manutenção da consistência entre as diversas representações do mundo virtual, uma para cada estação participante. O problema surge quando múltiplos usuários tentam interagir simultaneamente com uma mesma entidade.

Várias abordagens são propostas para solucionar o problema. Uma simples defende que o sistema pode confiar em regras de convívio social para evitar conflitos. A partir do momento que cada usuário está ciente das ações dos outros usuários, o primeiro pode deixar que os outros ajam para então acessar um recurso comum. Por exemplo, dois avatares podem tentar abrir e atravessar uma porta simultaneamente, o que acarretará uma perda na consistência do cenário virtual. Um dos dois pode deixar o outro atravessar a porta para seguí-lo logo após. Haverá a aplicação de uma regra de convívio social de nosso cotidiano pelos usuários, controladores de seus avatares, no ambiente virtual. Esta abordagem funciona satisfatoriamente para ambientes virtuais colaborativos, porém ela é desaconselhável em sistemas que haja disputa entre os participantes. Em sistema de simulação de combate ou em jogos, seguir regras de convívio social ou educação significa normalmente a morte ou derrota do usuário.

Nas aplicações em que há disputa entre os participantes, fica a cargo do sistema definir a ordem de solicitação de acesso ao recurso. A primeira mensagem a chegar recebe autorização do *software* de controle para acessá-lo. Esta abordagem pode ser injusta, quando leva-se em conta a latência da rede, principalmente em WANs. Uma estação mais próxima ao servidor de recursos seria beneficiada em detrimento de uma localizada mais distante.

Outra abordagem define que uma entidade ao ser solicitada por um avatar deve trancar-se (*lock*) aos outros usuários. Somente o usuário, agora proprietário da entidade, pode liberá-la para o uso por outros. De forma semelhante, uma entidade pode ligar-se (*attachment*) a um usuário, porém a entidade pode desligar-se deste usuário a qualquer momento. Em ambas as abordagens a posição e orientação das entidades trancadas ou ligadas passam a ser relativas ao usuário proprietário ou associado, não adicionando tráfego a rede com a sua movimentação.

Arquitetura

Importante no projeto de sistemas de DVE é a escolha do protocolo de transporte adotado. O TCP (*Transmission Control Protocol*) oferece uma abordagem orientada a conexão, confiável, mas com o custo de uma alta latência. O UDP (*User Datagram Protocol*) oferece uma abordagem não orientada a conexão, não confiável, mas pode-se utilizar retransmissões periódicas para prover uma maior garantia de que nenhuma das estações perdeu alguma informação crucial. Entre as vantagens e desvantagens de cada abordagem, a maioria dos sitemas de DVE existentes utiliza uma combinação de ambos.

A topologia de rede utilizada também é importante. A solução mais simples é um único servidor central, onde todas as estações conectam-se e para onde enviam todas as informações de atualização, como na Figura II.1(a). O servidor faz as filtragens de dados possíveis e repassa as informações às estações pertinentes. Esta abordagem funciona bem para ambientes de poucos usuários, mas nitidamente o servidor é um limitante para a escalabilidade do sistema.

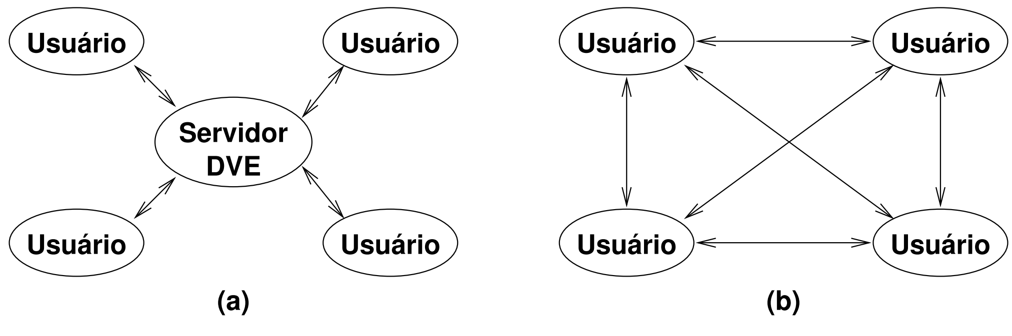

Figura II.1: Topologia: (a) Servidor central (b) Descentralizado.

Uma alternativa é o estabelecimento de conexões diretas entre as estações, eliminando-se o servidor central, como mostrado na Figura II.1(b). No entanto, cada estação deveria enviar $n-1$ atualizações, onde n é o número de estações envolvidas. Esta configuração acaba tornando-se também não escalável a milhares de usuários. Para mundos virtuais pequenos pode-se replicar os dados do ambiente em cada estação, deixando que o tráfego na rede limite-se a atualizações. Mundos virtuais maiores e mais complexos podem ser particionados de forma que cada estação possua as informações relacionadas à parte do mundo virtual em que se encontra. Os aspectos críticos para desenvolver ambientes escaláveis são discutidos por Macedonia e Zyda [20].

II.2 Sistemas de Ambiente Virtual Distribuído

Aqui serão apresentadas resumidamente características básicas de alguns dos principais sistemas DVE em desenvolvimento, tanto oriundos do mundo da Internet como da simulação militar. Estes sistemas possuem abordagens diferenciadas com relação aos problemas apresentados para o desenvolvimento de DVEs.

II.2.1 DIVE

No sistema DIVE (*Distributed Interactive Virtual Environment*) [21] da SICS (Instituto Sueco de Ciência da Computação) a descrição do mundo virtual é distribuída entre as estações envolvidas. Esta descrição é atualizada conforme os atores (ou avatares) entram ou saem do mundo ou de um grupo de *multicast*. Podem existir vários mundos simultâneos de forma independente, onde cada mundo possui o seu próprio conjunto de objetos e entidades participantes. Estes mundos virtuais são conectados através de portais inseridos em cada mundo. Ao cruzar um destes portais, o ator troca de mundo virtual e conseqüentemente de grupo *multicast*.

A descrição do mundo virtual é compartilhada por todos os atores participantes daquele mundo. O sistema DIVE privilegia a interação entre os usuários em detrimento da qualidade gráfica do mundo virtual, sendo muitas vezes criticado pela aparência de sua modelagem tridimensional.

A comunicação de rede é baseada em protocolos *multicast* confiáveis. A técnica de *dead reckoning* é utilizada fornecendo-se a posição e as velocidades linear e angular dos objetos. Os atores interagem limitadamente com o mundo virtual em que estão inseridos, limites estes baseados em parâmetros chamados de aura, focus e nimbus. A aura de um ator define o seu limite de comunicação com outros atores, pois dois atores somente podem interagir quando suas auras interceptam-se. O focus determina a quais entidades o ator está dedicando sua atenção, enquanto o nimbus define o alcance de sua capacidade de ser detectado por outros. Ao manipular estes três parâmetros um usuário pode controlar a capacidade de outros atores detectarem-no e iniciarem uma comunicação. Por outro lado, também permite que um ator limite a sua comunicação a somente aqueles outros atores de seu maior interesse.

II.2.2 MASSIVE

O MASSIVE (*Model, Architecture and System for Spatial Interaction in Virtual Environments*) [22] é um projeto da Universidade de Nottingham, Inglaterra. Este sistema permite que usuários comuniquem-se por qualquer combinação de texto, gráfico ou áudio sobre redes LAN ou WAN. O objetivo principal é o desenvolvimento de ambientes virtuais colaborativos escaláveis, que possuam possibilidades de acesso heterogêneas. Em um caso extremo, usuários em sistemas dedicados de realidade virtual poderiam manter algum tipo de interação mínima com outros usuários que estejam acessando de terminais texto.

O sistema possui um controle de consciência da presença de outros objetos (*awareness levels*) baseado nos conceitos de aura, focus e nimbus, semelhante ao DIVE. A mediação espacial (*Spatial Mediation*) faz com que a percepção de outros através de diferentes mídias seja influenciada por fatores como a posição e orientação relativas. Por exemplo, um usuário mantendo o mesmo tom de voz é ouvido com um volume mais alto ao aproximar-se de outro.

II.2.3 NPSNET

O pacote NPSNET [23] foi desenvolvido na *Naval Postgraduate School* para suportar simulações virtuais de treinamento militar em larga escala. Atualmente suporta centenas de usuários, mas objetiva suportar milhares de usuários em uma área de vasta extensão.

Este sistema baseia-se no padrão DIS para realizar a comunicação entre as entidades envolvidas. O sistema distribui os pacotes DIS utilizando *IP-multicast*, com o objetivo de minimizar o número de pacotes enviados.

Originalmente, a maioria dos objetos comunicantes consistia de veículos militares. Posteriormente foram desenvolvidos modelos que pudessem representar a figura humana com articulações, que tem sua movimentação efetuada por programas *script* reagindo a comandos simples.

O sistema replica, por ser parte do padrão DIS, a base de dados relativa a todos os objetos inseridos no ambiente em cada estação, criando um fator limitante para uma aplicação com milhares de usuários devido ao alto custo individual das estações necessárias. Atualmente o NPSNET está desenvolvendo o suporte à divisão dos objetos em classes espaciais, temporais e funcionais, com o objetivo de minimizar o volume de informação armazenada em cada estação.

II.2.4 SPLINE

O sistema SPLINE (*Scalable Plataform for Large Interactive Networked Environments*) suportado pelo MERL (*Mitsubishi Electric Research Labs*) possui o objetivo de criar ambientes virtuais multiusuários para trabalho cooperativo. A comunicação entre os participantes pode ser realizada por simulações tridimensionais ou por linguagem falada natural. Ao conteúdo de objetos de um determinado mundo virtual podem ser acrescidos novos objetos temporários ou permanentes desenvolvidos por usuários, ou seja, o desenvolvimento do ambiente virtual não fica apenas a cargo dos implementadores do sistema. É definida uma interface aberta para a interligação destes mundos criados por diferentes usuários.

O mundo virtual é particionado pelo sistema SPLINE em zonas conhecidas por locais [24]. Cada local possui um sistema próprio de coordenadas e seus endereços correspondem ao endereços *multicast* usados para o envio de informações sobre o local e seu conteúdo. Também podem ser definidos locais dentro de outros locais. Por exemplo, em uma cidade virtual, um prédio pode ser um local, dividido em vários locais, como salas, elevadores, etc.

Capítulo III

As Linguagens VRML e Java

Para a implementação do ambiente virtual distribuído proposto neste trabalho utilizou-se a linguagem VRML 2.0 para a descrição do cenário virtual tridimensional de forma combinada com a linguagem Java, a qual é responsável pela interface gráfica com o usuário e pela comunicação de cada cliente com o servidor, também escrito em Java.

III.1 VRML

VRML é o acrônimo de *Virtual Reality Modeling Language*. Trata-se de um formato de arquivo para a descrição de objetos tridimensionais interativos que podem então compor mundos virtuais. A linguagem VRML pode ser vista como um formato de arquivo descritivo para três dimensões análogo ao que HTML (*HyperText Markup Language*) representa para duas dimensões.

III.1.1 Surgimento e Evolução de VRML

Ao mesmo tempo que os sistemas de RV começaram a amadurecer, a utilização da Internet começou a ter um crescimento exponencial. O fato mais marcante para este crescimento foi o advento da *World Wide Web* (WWW), que possibilitou o acesso à informação na Internet através de uma interface gráfica amigável. HTML é a linguagem que descreve as páginas bidimensionais a serem apresentadas pelos interpretadores WWW. A fusão de realidade virtual com a Internet tem um grande potencial de aplicação. Era necessário, porém, um conceito equivalente ao HTML, uma linguagem que pudesse armazenar mundos tridimensionais da mesma maneira que HTML armazena documentos. Como há a possibilidade de executar *helper applications* sob os interpretadores WWW, as pessoas poderiam acessar e visualizar estes mundos virtuais com seus interpretadores já existentes.

VRML foi idealizada no início de 1994 na Primeira Conferência Anual de WWW em Genebra, Suíça. Tim Berners-Lee, o criador do sistema de hipermídia WWW, e Dave Raggett organizaram sessões para discutir interfaces de RV para WWW. Diversos projetos, já em andamento, foram apresentados como ferramentas para a visualização gráfica tridimensional que interoperavam com a Web. Entre estes projetos, estava o *Labyrinth*, criado por Mark Pesce e Anthony Parisi. O *Labyrinth* não era um interpretador VRML, até mesmo porque este conceito ainda não havia se desenvolvido, porém já possuia a habilidade de compôr objetos 3D no WWW utilizando os mesmos protocolos que as páginas deste ambiente utilizavam-se. Todos os presentes concordaram que estas ferramentas necessitavam de uma linguagem comum para especificar a descrição de uma cena tridimensional e de *hyperlinks* para outros endereços WWW, ou seja, uma linguagem para realidade virtual análoga à HTML. Foi cunhado o termo *Virtual Reality Markup Language* (VRML) e foram iniciados os trabalhos para a especificação da linguagem após a conferência. O termo *Markup* foi posteriormente substituído por *Modeling*, pois este estava mais adequado ao conceito de modelagem proposto pela VRML.

Logo após a conferência, uma lista de discussão, chamada *www-vrml*, foi criada para debater as propostas para a primeira versão de VRML. A lista era coordenada por Mark Pesce e em uma semana já havia mais de mil membros. Passadas as primeiras semanas, Mark Pesce anunciou sua intenção de ter um primeiro rascunho para a próxima conferência a ser realizada em outubro de 1994 em Chicago, apenas cinco meses adiante. Foi decidido que a primeira versão atenderia o básico e versões posteriores aumentariam a interatividade. Foram estabelecidos como requisitos: a independência de plataforma, a extensividade e a capacidade de funcionar bem sobre conexões de largura de faixa estreita, possibilitando o uso doméstico através de modens com uma linha telefônica.

Para evitar que fosse preciso reinventar procedimentos para funções bem conhecidas, a primeira versão [25] baseou-se em uma linguagem já existente: a *Open Inventor* da Silicon Graphics, Inc. O primeiro rascunho foi apresentado após a segunda conferência por Mark Pesce, Anthony Parisi e Gavin Bell, da Silicon Graphics. Nesta primeira versão os objetos são estáticos e há possibilidades limitadas de interação entre o cenário virtual e o usuário.

A versão 2.0 [26] incorpora significativos avanços nas capacidades de interação. Ela foi projetada pela mesma equipe da Silicon Graphics que trabalhou na primeira versão com contribuições da Sony Research e da Mitra, além das sugestões pertinentes colhidas através da lista de discussão sobre vrml (www-vrml@wired.com). A proposta feita pela Silicon Graphics para a versão 2.0 de VRML foi batizada de *Moving Worlds*. O nome foi escolhido pela comunidade participante da lista de discussão. Esta foi a proposta escolhida, concorrendo com outras cinco, após um debate de dois meses promovido pelo Grupo de Arquitetura VRML (*VRML Architecture Group* - VAG) em março de 1996.

III.1.2 Conceitos Básicos de VRML 2.0

Nesta seção serão apresentados conceitos básicos para a descrição de cenários virtuais utilizando a linguagem VRML. Ames *et al* [27] é uma referência completa para a versão 2.0 de VRML.

Arquivo VRML 2.0

A extensão adotada nos arquivos VRML é *.wrl*, em uma referência à *world*. O arquivo pode ser criado em qualquer editor de textos, sendo que a primeira linha deve conter um cabeçalho que o identifica como sendo um arquivo VRML 2.0:

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

O caractere # indica que o restante da linha é formada por comentários e deve ser ignorada pelo interpretador VRML. UTF8 indica um conjunto internacional de caracteres definido pela ISO, que engloba os caracteres ASCII.

Nós, Campos e Eventos

Os objetos descritos podem conter vários recursos para a composição do cenário virtual: geometria espacial, arquivos MIDI ou WAV, imagens JPEG ou GIF entre outros. Em VRML é definido um tipo especial de objeto capaz de tratar todos estes recursos. Este objeto, conhecido por **nó** (*node*), é definido pelo seu tipo e pelos parâmetros que o distinguem de outros nós do mesmo tipo. Estes parâmetros são chamados **campos** (*fields*) e **eventos** (*events*). A especificação de cada nó define o seu tipo, nome e o valor inicial para cada um dos campos. Os campos contém o estado atual do nó e podem ser de dois tipos: *field* ou *exposedField*. Os campos de ambos os tipos definem o valor inicial do estado do nó. No entanto, os campos do tipo *field* não podem ser alterados, sendo considerados privados, enquanto os campos do tipo *exposedField* são considerados públicos e podem ser alterados por outros nós. Eventos são melhor detalhados adiante.

A sintaxe para a representação de um nó em um arquivo VRML é a seguinte:

tipoNó { campos }

Somente o tipo do nó e as chaves são realmente necessários. Caso algum campo não esteja especificado, o valor *default* definido pela especificação do nó será adotado.

Também é possível fazer referência a um nó múltiplas vezes em um arquivo VRML. Pode-se nomear um nó através da comando DEF e posteriormente reutilizar a definição deste nó referenciando-o pelo comando USE.

Grafo de Cena

Em VRML, existem basicamente dois tipos de nós: os de agrupamento e os nós filhos. Cada nó de agrupamento contém um conjunto de nós filhos que são hierarquicamente descendentes destes nós de agrupamento.

Todos os objetos descritos em um arquivo VRML, nós de agrupamento mais nós filhos, formam uma hierarquia de objetos conhecida como **grafo de cena**. Qualquer ação realizada sobre um determinado nó afeta todos os nós localizados abaixo deste na hierarquia do grafo de cena.

Considerações sobre a Linguagem

As unidades de medida de tempo e ângulo são, respectivamente, segundos e radianos. Cores são definidas pelo padrão RGB. Para a medição de distância é recomendada a adoção do metro como unidade de referência para possibilitar a reutilização de objetos sem comprometer a escala entre os diversos objetos componentes do mundo virtual.

VRML usa um sistema de coordenadas tridimensional cartesiano. O mundo virtual, descrito em um arquivo VRML, contém um sistema de coordenadas único. Cada nó de agrupamento cria um sub-sistema de coordenadas para seu conjunto de nós filhos. Toda transformação sofrida por um nó é acumulativa ao longo da hierarquia do grafo de cena, ou seja, todos os nós sofrem a transformação sofrida por um nó que esteja situado acima deles na hierarquia.

Na Figura III.1, há um possível grafo de cena para a modelagem de um boneco simples. Para ilustrar o conceito de grafo de cena hierárquico descrito acima, pode-se usar um exemplo simples. Caso o boneco tivesse a cabeça rotacionada para algum lado, todos os pontos da hierarquia abaixo da cabeça teriam seus sub-sistemas de coordenadas rotacionados de acordo. Assim, olhos, boca e nariz são movidos acompanhando a cabeça mantendo a consistência do boneco.

Tipos de Nós

Existem nós que tem a característica de serem únicos em um dado momento do mundo virtual. Eles são chamados de nós filhos associáveis (*Bindable Children Nodes*). Exemplos são os nós *Background*, que define o céu e o chão em espaços abertos, *NavigationInfo*, que informa ao interpretador o modo de navegação do usuário e a geometria de sua

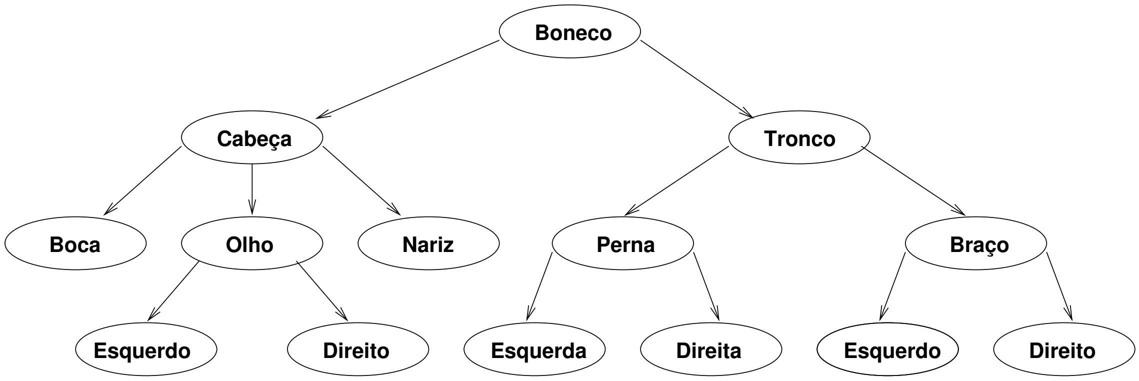

Figura III.1: Exemplo de Grafo de Cena.

representação no mundo virtual, ou *Viewpoint*, que define as coordenadas de onde o usuário está visualizando o ambiente virtual. Não pode haver dois nós de cada tipo ativos simultaneamente. Existe, portanto, uma espécie de pilha para cada um dos tipos destes nós. O último nó de cada tipo a ser ativado vai para o topo da pilha e permanece sendo usado pelo interpretador para sintetizar o mundo virtual até ser destituído pelo mesmo processo.

Outros tipos genéricos de nós existem:

- Geométricos - estes devem estar contidos em um nó *Shape* para serem visíveis ao usuário. Neles são definidos, entre outros, os objetos volumétricos primitivos, como esferas, cilindros, cubos ou cones;
- Sensores - detectam as ações do usuário ou a passagem de tempo gerando eventos para outros nós. São fundamentais para o estabelecimento de interatividade com o usuário;
- Interpoladores - estes definem uma função linear discreta $f(t)$, onde t pode assumir valores de $-\infty$ a $+\infty$. Existe um campo chamado *key* que define os n valores de t e outro denominado *keyValue*, com os n valores correspondentes a $f(t)$. Logo, deve haver o mesmo número de elementos em ambos os campos. Considerando-se os extremos no infinito, existem $n+1$ sub-intervalos. Os valores de $f(t)$ nestes sub-intervalos são obtidos fazendo-se uma interpolação entre os valores definidos. O

tipo dos elementos no campo *keyValue* são determinados pelo tipo de interpolador. Estes podem interpolar cores, posições, orientação e escalas, entre outros. Com isto pode-se animar objetos, mudando suas cores, dimensão ou posição;

- Fontes de Luz - de maneira geral, os objetos são iluminados pela composição de todas as luzes no mundo virtual que os afetam. Os diferentes tipos de iluminação são ilustrados na Figura III.2. O tipo direcional forma um plano de luz perpendicular a um vetor determinado em sua definição. A fonte pontual gera luz em todas as direções à sua volta. A fonte tipo *spot* forma um cone de luz, com dimensões estabelecidas em sua definição. Todos possuem campos para controlar a intensidade, cor e emissão de luz para o ambiente.

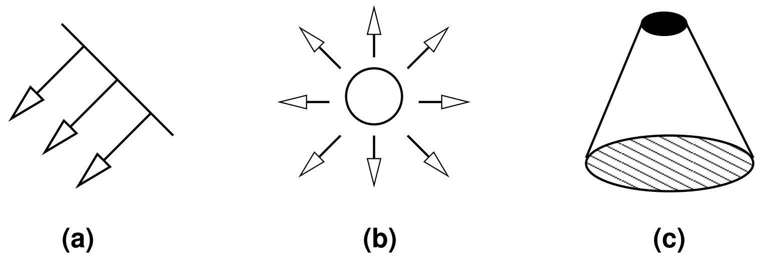

Figura III.2: Fontes de Luz: (a) Direcional (b) Pontual (c) *Spot*.

Eventos

Os eventos podem ser de dois tipos: de entrada (*eventIn*) ou de saída (*eventOut*). Eventos sendo recebidos são mensagens de outros nós com o intuito de alterar algum estado do nó receptor. Os campos evento de entrada e de saída são as interfaces que os nós possuem para comunicar-se com os outros nós descritores do mundo virtual.

A conexão entre o nó gerador do evento e o nó receptor do evento é conhecida por **rota** (*route*). Uma rota pode ser estabelecida através do comando ROUTE utilizando-se a seguinte sintaxe:

```
ROUTE nomeNó.nomeEventOut_changed TO nomeNó.set_nomeEventIn
```

Os tipos dos eventos de entrada e de saída devem ser os mesmos. Rotas devem ser estabelecidas de eventos de saída para um ou mais eventos de entrada ou campo exposto (*exposedField*).

Nós conhecidos como nós sensores geram eventos, devido a ações do usuário (*ProximitySensor*, *VisibilitySensor*, *TouchSensor*, *CylinderSensor*, *PlaneSensor* e *SphereSensor*) ou a passagem de tempo (*TimeSensor*). Eles podem iniciar sequências de eventos, onde mensagens são passadas a outros nós que por sua vez geram eventos para outros nós. Este processo é conhecido como cascata de eventos. Muitas vezes existem nós intermediários neste processo, que são conhecidos como nós interpoladores (*ColorInterpolator*, *CoordinateInterpolator*, *NormalInterpolator*, *OrientationInterpolator*, *PositionInterpolator*, *ScalarInterpolator*). Ao serem acionados estes enviam valores como eventos de saída. Estes valores são interpolações dos valores estabelecidos em sua definição.

Na Figura III.3 está ilustrada uma cascata de eventos que faz um objeto chamado Exemplo girar ao ser tocado pelo usuário. Quando o interpretador percebe o toque do usuário no objeto, ele aciona o sensor de toque associado ao objeto. Este passa uma mensagem, através de uma rota, iniciando a ação de um sensor de tempo. Por sua vez, o sensor de tempo irá acionar um nó interpolador de orientação que fará o objeto Exemplo girar pelas posições definidas no nó interpolador durante o tempo definido no nó sensor de tempo.

A estrutura do código em VRML que implementa a cascata de eventos do exemplo anterior (omitindo as definições de cada nó) é mostrada a seguir:

```
#VRML V2.0 utf8
DEF EXEMPLO Transform{
    children[ # Geometria do objeto exemplo ]
DEF TOQUE TouchSensor{}
}
DEF TEMPO TimeSensor {
    ... # Definição do sensor de tempo
}
```

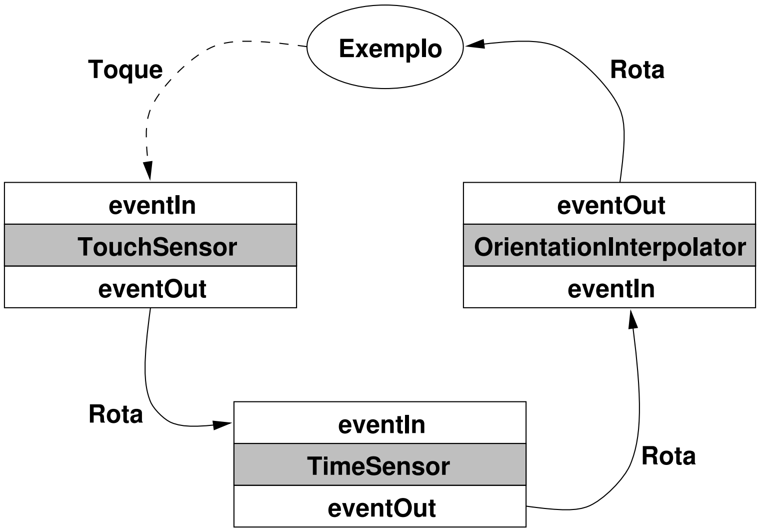

Figura III.3: Exemplo de Roteamento de Eventos.

```
DEF GIRO OrientationInterpolator {
    ... # Definição do interpolador de orientação
}
ROUTE TOQUE.touchTime TO TEMPO.startTime
ROUTE TEMPO.fraction_changed TO GIRO.set_fraction
ROUTE GIRO.value_changed TO EXEMPLO.rotation
```

Protótipos

A prototipação permite o encapsulamento e a parametrização da geometria, atributos, comportamentos ou alguma combinação destes. Novos nós podem ser criados pelo usuário. A definição de um protótipo assume a seguinte forma:

```
PROTO  nomePrototipo [
       eventIn       nomeTipoEvento nome
       eventOut      nomeTipoEvento nome
       exposedField  nomeTipoCampo  nome  valorDefault
       field         nomeTipoCampo  nome  valorDefault
```

```
... ] {
       Definição da geometria e comportamento do PROTO
}
```

Os nomes de campos e eventos devem ser únicos dentro de um protótipo. Um protótipo não cria o objeto definido na cena gráfica, portanto, é necessário definir um nó do tipo do protótipo para realmente criar o objeto no mundo virtual. O primeiro nó achado na definição do protótipo é usado para definir o tipo de nó do protótipo.

Os campos do protótipo são associados com os campos da definição do nó usando a palavra reservada IS. Um exemplo de prototipação pode ser visto a seguir:

```
# Definindo CARRO
PROTO CARRO [ field SFVec3f localizacao 0 0 0 ] {
Transform {
    translation IS localizacao
    children [
        ... # Geometria do carro
] } }
# Criando carros no mundo virtual
Transform {
    children [
    CARRO { localizacao 10 0 10 }
    CARRO { }
] }
```

Neste exemplo é definido um protótipo de um carro onde é passado como parâmetro a sua localização. Após a definição do protótipo, são criados dois objetos CARRO nas posições (10, 0, 10) e (0, 0, 0).

Também é possível a criação de protótipos em arquivos externos. Estes são referenciados através da palavra reservada EXTERNPROTO. Sua sintaxe é mostrada abaixo:

```
EXTERNPROTO nomePrototipo [
    # mesmos campos e eventos de PROTO
] "url" ou [ url, url, ...]
```

No arquivo VRML fica definida a interface do novo nó com o mundo virtual e registrado o *url* ou uma lista de *url's* que indiquem o arquivo externo onde está definido o protótipo.

LOD

Na síntese de um cenário virtual existe um compromisso entre a quantidade de detalhes para um maior realismo e a rápida execução da cena gráfica para uma maior interatividade. Um objeto que está próximo ao usuário deve ser desenhado com mais detalhes, porém um objeto distante não necessita de tantos detalhes. Por exemplo, um interpretador não tem a necessidade de mostrar a cor dos olhos de um avatar que está a 50 metros de distância do usuário.

Em VRML, existe um nó que permite ao criador do mundo estabelecer níveis de detalhe para um determinado objeto. Este nó é o LOD de *level of detail.* Cada objeto pode ser criado com múltiplas versões, geralmente com alto, médio e baixo detalhismo para distâncias baixa, média e alta, respectivamente.

A definição de um nó LOD provê ao interpretador a informação necessária para que este faça a troca automática entre as diferentes versões do objeto com base na distância entre o objeto e o usuário que o está visualizando.

```
Transform {
    children LOD {
        range [ 40, 80 ]
        level [
        Transform { ... } # Geometria para distância entre 0 e 40
        Transform { ... } # Geometria para distância entre 40 e 80
        Transform { ... } # Geometria para distância de 80 em diante
```

```
] } }
```

O Usuário

O usuário é visto pelo interpretador como mais um objeto inserido no mundo virtual. O modo como o usuário será tratado com relação aos outros objetos é descrito no nó *NavigationInfo*. Como definido no Capítulo II, o usuário é representado por uma entidade conhecida como **avatar**. No nó *NaviagationInfo* é definido um cilindro que cerca o avatar para efeitos de detecção de colisão e ativação de sensores de proximidade, visibilidade, etc. Outras características que podem ser definidas são a velocidade de deslocamento e a forma de navegação através do mundo virtual (andar, voar ou examinar).

A visão que o usuário tem do cenário virtual está sempre iluminada como se o participante possuísse um luz sobre a cabeça permanentemente voltada na direção de seu olhar, como em um capacete de mineiro. Para a criação de efeitos de luzes, em mundos com realismo mais avançado, pode-se desligar esta luz e iluminar o cenário com os nós próprios para iluminação. O controle de ativação da luz no "capacete" do usuário é feita pelo campo *headlight* no nó *NavigationInfo*.

Agregando arquivos VRML

Pode-se acrescentar a um arquivo VRML o conteúdo definido em outro arquivo VRML. Este recurso é realizado pelo uso do comando Inline. O conteúdo do arquivo chamado pelo comando Inline sofre toda a ação executada sobre o ponto da hierarquia do grafo de cena onde esse foi chamado.

Através deste recurso pode-se distribuir a geometria do mundo virtual e a partir de um arquivo principal chamar os detalhes do mundo virtual definidos em arquivos secundários.

III.2 A Linguagem Java

A Sun Microsystems, Inc. lançou a linguagem Java [28, 29, 30] em 1995. Os estudos começaram em 1991 e a sua principal motivação era o desenvolvimento de uma linguagem de programação independente de plataforma para a criação de programas para controlar equipamentos eletrônicos de consumo, como fornos de microondas ou controles remotos. No decorrer do projeto, a Internet emergiu para o grande público e os estudos foram reorientados para a atender o novo mercado de WWW.

A linguagem Java pode ser utilizada para a criação de aplicações que executem independentes de qualquer interpretador WWW (*stand-alone applications*) ou *applets*, que são programas em Java que podem ser incluídos em uma página HTML. Para a criação da interface gráfica há o *Abstract Window Toolkit* (AWT).

III.2.1 Aspectos Gerais de Java

Tradicionalmente, as linguagens de programação são classificadas em linguagens interpretadas ou compiladas. As linguagens interpretadas, como Basic ou Tcl, possuem a vantagem de serem independentes de plataforma. Um interpretador executa cada comando à medida que vai lendo o arquivo fonte do programa. Este procedimento torna a execução do programa extremamente lenta. Nas linguagens compiladas, como C ou Pascal, o arquivo fonte é previamente lido por um programa chamado compilador, que gera um programa executável na linguagem de máquina do processador utilizado. Há um grande aumento de desempenho em relação às linguagens interpretadas, mas perde-se portabilidade.

Em Java, propõe-se uma abordagem híbrida. Um compilador Java não cria um código específico a cada processador, mas um código intermediário chamado de código de máquina virtual Java (*Java Virtual Machine Code*) ou ***bytecode***. A Figura III.4 ilustra o processo de compilação de um programa Java. Neste passo é feita toda a verificação sintática e há otimização do código. O *bytecode* é um conjunto de intruções altamente otimizado executado pela máquina virtual que o sistema Java emula em tempo de execução. Para a execução do programa Java utiliza-se um interpretador que lê o código de máquina virtual e o transforma em código específico para cada processador, como mostrado na Figura III.5. Este interpretador pode ser um *browser* WWW apto a operar com Java, no caso de *applets*, ou um interpretador fornecido pelo sistema Java para aplicações *stand-alone*.

Figura III.4: Compilação de um Programa Java.

Figura III.5: Execução de um Programa Java.

III.2.2 Sintaxe e Estrutura de Java

A sintaxe da linguagem Java baseia-se na linguagem C com as características de orientação a objetos de C++. O objetivo foi atrair o grande número de programadores C/C++ já existentes. As regras sintáticas são basicamente as mesmas, mas as linguagens não são compatíveis. Existem oito tipos primitivos de dados, mostrados na Tabela III.1, cada um com um tamanho específico, independente da implementação local.

Operações aritméticas podem ser realizadas pelas mesmas regras de C/C++ (inclusive ++ ou −−) e comparações pelos operadores relacionais comuns, como $<$, $>=$, $==$ ou $!=$.

Tabela III.1: Tipos de Dados em Java.

Tipo	Tamanho(bits)	Abrangência
Byte	8	inteiro com sinal entre -128 e 127
Short	16	inteiro de 2 bytes com sinal
Int	32	inteiro de 4 bytes com sinal
Long	64	inteiro de 8 bytes com sinal
Float	32	1.7e-308 a 1.7e+308
Double	64	3.4e-038 a 3.4e+038
Boolean	1	verdadeiro ou falso
Char	8	código em Unicode

Pacotes (*Packages*)

Pacotes são conjuntos de arquivos *.class* agrupados de acordo com sua funcionaldade. Estes pacotes podem, inclusive, agrupar outros pacotes, criando uma hierarquia entre as **classes** (arquivos *.class*). Para que um programa Java possa acessar classes ou pacotes inteiros deve-se incluí-los com a palavra de comando *import*. A sintaxe do comando é mostrada a seguir:

```
import pacote1[.pacote2].(nomeClasse ou *);
```

Para incluir uma classe específica, nomeia-se a classe no comando *import*, ou um pacote inteiro usando o caracter *.

III.2.3 *Java Developer's Kit* (JDK)

As ferramentas necessárias para desenvolver programas em Java compõem o *Java Developer's Kit* (JDK), que é fornecido gratuitamente pela Sun Microsystems, Inc. via rede. Existem versões para Windows, Mac ou UNIX. Os principais programas que fazem parte do JDK e suas funções estão na Tabela III.2.

Tabela III.2: Componentes do JDK.

Programa	Propósito
javac	compilador Java
java	interpretador Java
javadoc	gerador de documentação
javap	decodificador (*disassembler*) de *bytecode*
jdb	*debugger* para Java
appletviewer	testar e executar *applets*

III.2.4 Aplicações Java e *Applets*

Após a compilação de um arquivo fonte Java (programa javac do JDK), são gerados os *bytecodes* correspondentes em arquivos com extensão *.class*. Caso a aplicação seja do tipo *stand-alone* pode-se executá-la diretamente acionando o interpretador Java (programa java do JDK). Se estiver sendo criado um *applet*, pode-se usar o *appletviewer* para visualizá-lo. Ao *appletviewer* passa-se um arquivo HTML que aciona o *bytecode* através do comando APPLET CODE. Por exemplo, pode-se criar o seguinte *applet*:

```
// Programa TesteApplet.java
import java.awt.*;
import java.applet.*;

public class TesteApplet extends Applet {
    public void paint(Graphics g) {
        g.drawString("Testando o uso de Applets.", 5, 25);
} }
```

Para compilá-lo utiliza-se o programa javac do JDK.

```
javac TesteApplet.java
```

Deve-se criar um arquivo HTML que ative o *applet*.

```
<!-- TesteApplet.html -->
<HTML>
<APPLET CODE="TesteApplet.class" WIDTH=300 HEIGHT=200></APPLET>
</HTML>
```

Utiliza-se então o *appletviewer* e tem-se o resultado na Figura III.6.

```
appletviewer TesteApplet.html
```

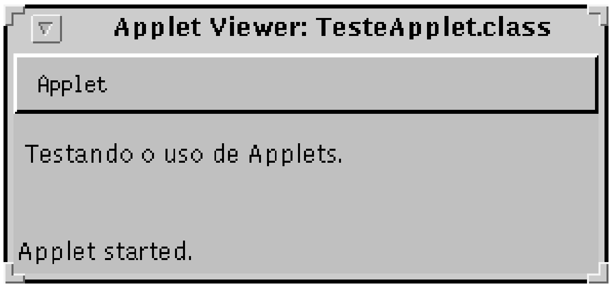

Figura III.6: Exemplo de *Applet*.

III.2.5 *Abstract Window Toolkit* (AWT)

O *Abstract Window Toolkit* (AWT) provê aos *applets* suporte para diversos recursos relacionados com o uso de uma interface gráfica, possibilitando a adição de controles e procedimentos de resposta à eventos. A seguir são apresentados os principais controles com figuras ilustrativas de sua apresentação na tela.

Painel (*Panel*)

A área gráfica disponível ao *applet* é um objeto do tipo *Panel*. Todos os demais componentes são acrescidos a um objeto deste tipo. Pode-se definir um painel dentro deste painel principal, reservando-se uma sub-área dentro da área principal disponível ao *applet*. Facilita-se assim a diagramação da interface gráfica apresentada ao usuário.

Rótulos (*Labels*)

Um rótulo é um objeto do tipo *Label* que apresenta na área gráfica o conteúdo de uma seqüência de texto.

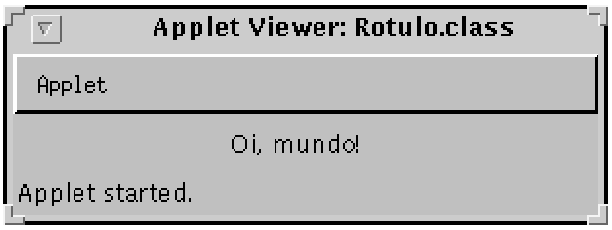

Figura III.7: Exemplo de Rótulo (*Label*).

Botões (*Buttons*)

Botões executam uma determinada seqüência de comandos quando acionados.

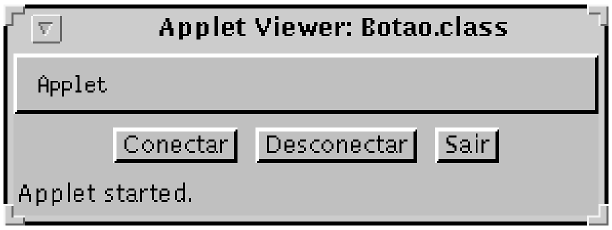

Figura III.8: Exemplo de Botão (*Button*).

Caixas de Checagem (*Check Boxes*)

Uma caixa de checagem é um controle que determina se uma opção está ativada ou não. Para cada caixa há um rótulo associado que descreve qual a opção que a caixa representa. Ao clicar-se sobre a caixa, troca-se seu estado. Estas caixas podem ser usadas individualmente ou como parte de um grupo de caixas de checagem. Quando parte de um grupo, somente uma caixa do grupo pode estar selecionada em determinado momento.

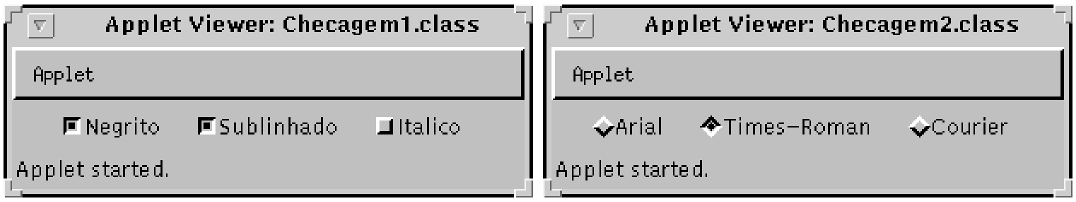

Figura III.9: Exemplo de Caixas de Checagem e Grupo de Caixas de Checagem.

Controles de Escolha (*Choice Controls*)

Através deste controle é possível criar uma lista do tipo *pop-up* de itens, de onde o usuário pode escolher uma entre várias opções.

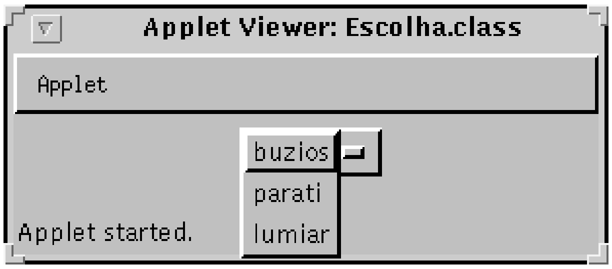

Figura III.10: Exemplo de Controle de Escolha (*Choice Control*).

Listas (*Lists*)

Listas diferenciam-se dos controles de escolha por permitirem que várias opções estejam à mostra e sejam selecionadas simultaneamente.

Barras de Rolagem (*Scroll Bars*)

São controles de visualização de outros componentes quando estes ocupam uma área maior do que a definida. No exemplo da Figura III.12 é apresentada a mesma lista do exemplo da Figura III.11, porém com uma área disponível menor.

As ações realizadas sobre as barras de rolagem podem ser associadas a comandos que modifiquem valores conforme a barra seja movimentada. Estas barras podem ser verticais

Figura III.11: Exemplo de Lista (*List*).

Figura III.12: Exemplo de Barra de Rolagem (*Scroll Bar*).

ou horizontais como na Figura III.13.

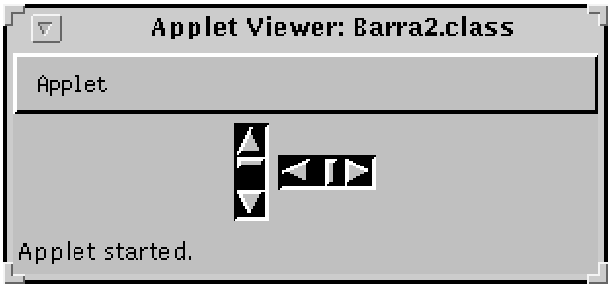

Figura III.13: Outro Exemplo de Barra de Rolagem (*Scroll Bar*).

Campo de Texto (*TextField*) e Área de Texto (*TextArea*)

Ambos os controles definem uma área de texto editável pelo usuário, diferenciando-se por possuirem uma linha (*TextField*) ou mais de uma linha (*TextArea*). Um exemplo de seu uso está na Figura III.14.

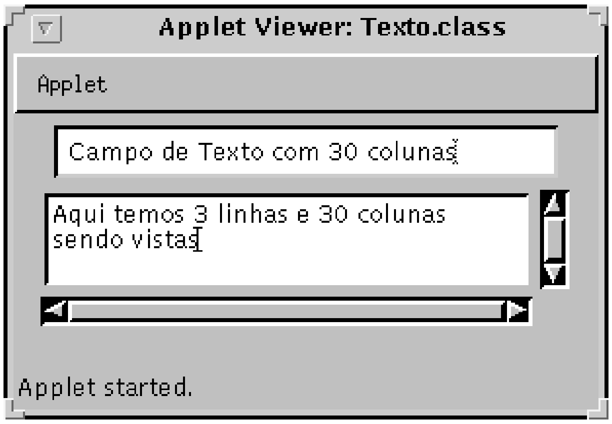

Figura III.14: Exemplo de Campo de Texto e Área de Texto.

Gerenciador de Diagramação (*Layout Manager*)

Todos os componentes descritos foram posicionados pelo gerenciador de diagramação *default*. Usando o conjunto AWT é possível redefinir o gerenciador de forma a distribuir os componentes na janela da forma desejada de uma maneira mais amigável ao programador.

III.3 Interface entre VRML e Java

Por ser uma linguagem descritiva, VRML não atende a necessidade de interações mais complexas [31, 32], como as que envolvam cálculos em tempo de execução, decisões lógicas com mais de uma condição a ser verificada ou suporte a comunicação de dados entre usuários visando a construção de ambientes virtuais compartilhados. Pode-se integrar um ambiente virtual descrito em VRML com programas escritos em outra linguagem, sendo Java a mais utilizada, através do estabelecimento de uma interface entre as linguagens pelo uso do nó *Script* [33].

III.3.1 Nó *Script*

O nó *Script* é similar aos outros nós em VRML com a distinção de permitir que o usuário defina campos extras. Os eventos que chegam a estes campos são automaticamente repassados ao programa associado ao nó. Pode-se, inclusive, criar novos nós com comportamento determinado pelo programa em Java. A seguir está a especificação do nó *Script*.

```
Script {
    exposedField  MFString    url           [ ]
    field         SFBool      directOutput  FALSE
    field         SFBool      mustEvaluete  FALSE
    # a partir deste ponto, pode-se definir qualquer quantidade de:
    eventIn       tipoEvento  nomeEvento
    field         tipoCampo   nomeCampo     valorInicial
    eventOut      tipoEvento  nomeEvento
}
```

Quando é criado um nó deste tipo, na realidade, está sendo criada uma interface entre o mundo VRML e o programa em Java associado. O programa associado é indicado pelo campo *url* que define uma lista de URLs (*Universal Resource Locator*) onde o programa pode ser encontrado.

Além do campo *url*, a sintaxe básica deste nó fornece dois outros campos que controlam o comportamento do nó *Script* em relação ao mundo virtual: *mustEvaluate* e *directOutput*.

O campo *mustEvaluate*

Conceitualmente, o programa *script* será executado toda vez que um evento de entrada for recebido pela interface do nó *Script* e então, algum processamento será realizado e um evento de saída será fornecido ao mundo virtual descrito pelo arquivo VRML. No entanto, se o cenário virtual contiver uma grande quantidade de nós do tipo *Script*, sensores e interpoladores, o interpretador VRML pode ficar bastante congestionado executando estes nós toda vez que algo mudar. Para evitar atrasar a síntese do mundo para usuário, o interpretador VRML pode adiar a execução de um nó *Script* para um momento mais conveniente. Vários eventos de entrada podem ficar acumulados até que o programa associado ao nó seja executado. Ao entrar em execução, o programa *script* recebe todos os eventos acumulados, trata-os e gera os eventos de saída correspondentes. O interpretador VRML somente passa a execução ao programa *script* quando não está muito ocupado com outros afazeres, como desenhar o mundo.

Todos os sensores e interpoladores padrão de VRML 2.0 permitem o acúmulo de eventos de entrada, retardando a sua execução. Porém, existem situações em que o programa *script* deve ser executado imediatamente após o recebimento de um evento de entrada por sua interface. O nó *Script* que tiver esta necessidade deve ter em seu campo *mustEvaluate* o valor verdadeiro. O valor *default* é falso, ou seja, permite o adiamento da execução e o acúmulo de eventos de entrada.

Este recurso deve ser usado moderadamente, pois a execução do programa *script* associado consome tempo e se o interpretador VRML for obrigado a executá-lo a cada evento de entrada recebido em sua interface a qualidade da navegação pode ficar sensivelmente prejudicada.

O campo *directOutput*

Normalmente, um programa *script* recebe os parâmetros necessários a sua execução através dos campos de sua interface e retorna valores ao mundo VRML alterando os campos de sua interface ou enviando eventos para outros nós através de seus eventos de saída. As rotas para estes eventos de saída estão previamente definidas no arquivo fonte VRML. Para a maioria dos nós *Script* esta interação é suficiente para estes alcançarem seus objetivos.

Colocando-se o valor verdadeiro no campo *directOutput* de um nó *Script*, este ganha, no entanto, muito mais poder sobre o cenário VRML em geral. Neste estado, o nó *Script* pode enviar valores para eventos de entrada de qualquer nó, monitorar o último valor enviado por qualquer evento de saída de qualquer nó e estabelecer dinamicamente novas rotas de eventos entre nós no cenário VRML. Como a modelagem do mundo pode mudar dinamicamente, o interpretador VRML perde boa parte de sua capacidade de otimizar a execução do cenário e com isto, pode haver queda de desempenho na navegação do usuário.

III.3.2 O Pacote VRML

O interpretador VRML fornece algumas características, classificadas como parte da API (*Application Programming Interface*) Java-VRML, ao programa *script* em Java. A API para Java provê um conjunto de classes e métodos que um programa *script* pode chamar para acessar campos e eventos de saída de sua interface. Os métodos *getField* e *getEventOutput* fornecem, respectivamente, manipuladores (*handles*) para campos ou eventos de saída da interface do programa *script*. Nesta API também existe um conjunto de classes para os tipos de dados encontrados em VRML. Métodos destas classes permitem a conversão entre os tipos de dados de VRML e Java. Estas classes estão no pacote VRML, acessível pelo comando `import vrml.*;`.

Um programa *script* responde a três tipos de chamada: inicialização, destruição e recebimento de eventos. A inicialização ocorre quando o programa *script* é primeiramente chamado e nenhum evento foi enviado a ele. Um método chamado *initialize* pode ser criado em Java para executar neste momento, inicializando variáveis e preparando-se para o recebimento dos eventos por vir. A destruição ocorre quando o programa *script* é descartado, pela saída do interpretador ou a leitura de um novo arquivo VRML. O método a ser usado é o *shutdown* em Java. Toda vez que é recebido um evento por algum evento de entrada na interface do programa *script* é acionado um método correspondente a cada evento, nomeado com o mesmo nome do evento de entrada. Alternativamente, pode-se ter um método chamado *processEvent* que trate todos os eventos recebidos.

III.3.3 O Pacote *Liquid Reality*

O pacote *Liquid Reality* [34] da DimensionX, recentemente adquirida pela Microsoft, é um conjunto de classes que formam um interpretador de arquivos VRML 2.0 fornecido atualmente de forma gratuita por estar em fase de testes. Pode-se com seu uso incluir a visualização de um ambiente virtual descrito em VRML em um programa Java desenvolvido pelo usuário.

Também pode-se manipular o ambiente virtual descrito alterando-se dinamicamente o arquivo VRML gerador do mundo virtual. Todas estas classes estão agrupadas em um pacote, que pode ser incluídas pelo comando `import dnx.*;`.

Capítulo IV

A Aplicação

Neste capítulo serão apresentados todos os componentes que formam a aplicação proposta, que foi desenvolvida nos laboratórios do Grupo de Teleinformática e Automação (GTA) da UFRJ.

O trabalho desenvolvido visou a implementação de uma interface gráfica com o usuário que permita que este interaja com outros usuários em um ambiente virtual compartilhado. Serão descritos os passos realizados para a descrição do mundo virtual, para a criação da interface gráfica, para o gerenciamento dos diferentes usuários e para a intermediação da troca de informações entre os participantes. O código fonte dos programas citados estão listados no Apêndice A.

IV.1 O Mundo Virtual

Foi criado um mundo virtual descrito em VRML, para ser o cenário de atuação dos usuários. Uma visão panorâmica do mundo virtual desenvolvido está apresentada na Figura IV.1. Há um grande espaço aberto, com um monumento e diversas árvores, e um espaço confinado, uma galeria de arte com representações de quadros clássicos. A Figura IV.2 ilustra um ângulo de visão interno da galeria de arte.

A descrição do mundo virtual é feita pelo arquivo *mundo.wrl*, que aciona arquivos

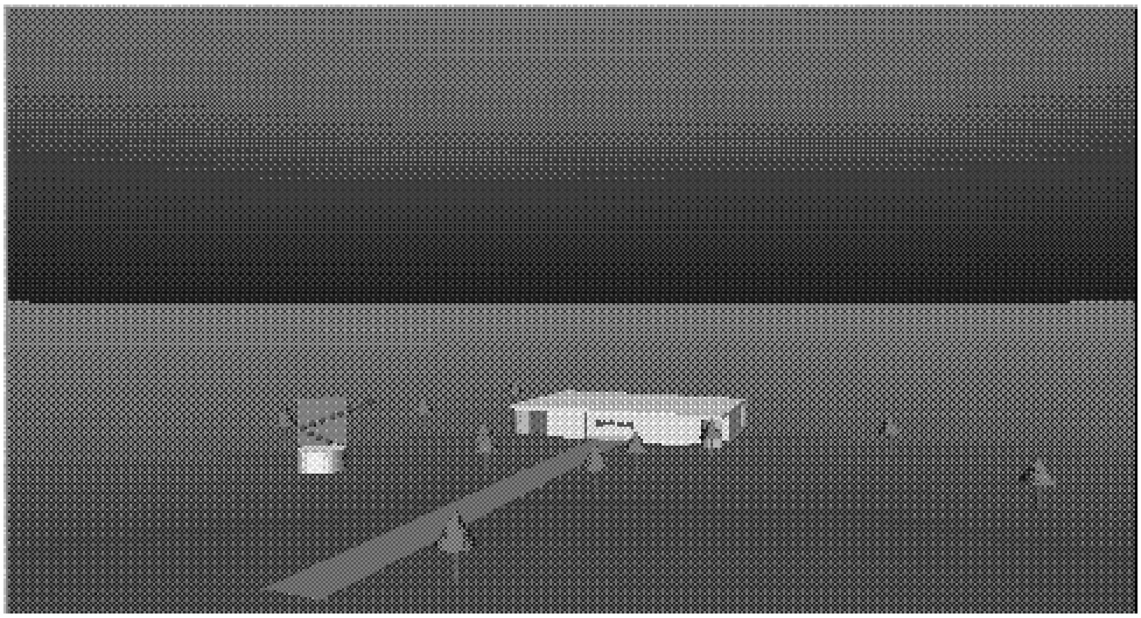

Figura IV.1: Visão Panorâmica do Mundo Virtual.

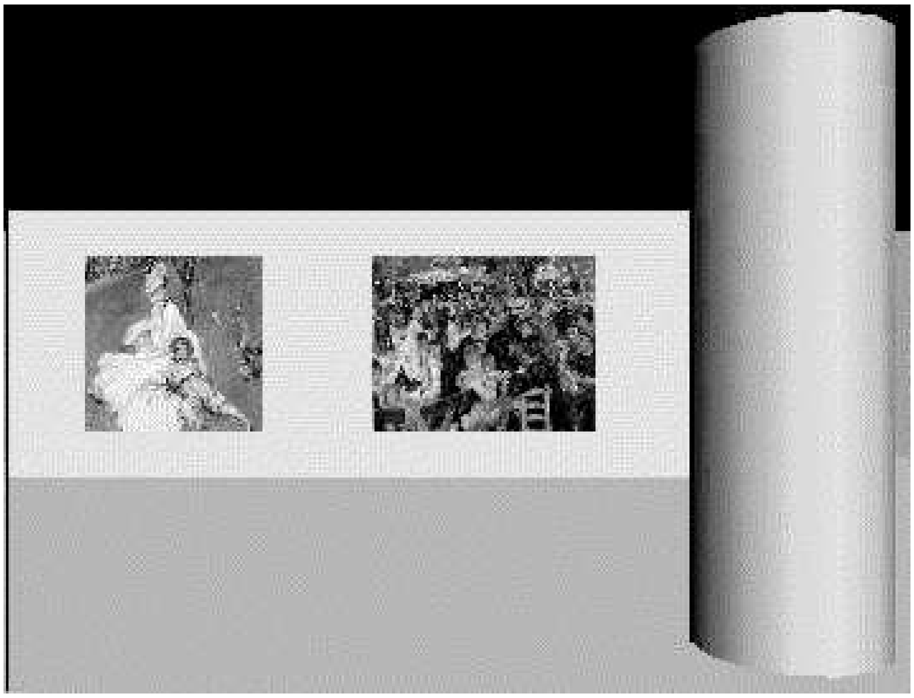

Figura IV.2: Visão Interna da Galeria de Arte Virtual.

externos que contêm, por exemplo, o protótipo das árvores e a descrição detalhada da galeria de arte. Desta forma, a manutenção e alteração do mundo virtual é facilitada, pois pode-se alterar ou acrescentar módulos simples ao invés de lidar com arquivos longos.

IV.1.1 Características Gerais do Mundo Virtual

O cenário virtual é iluminado por um nó de luz direcional (*DirectionalLight*) com o vetor de referência perpendicular ao plano do solo. Cria-se assism uma iluminação semelhante a encontrada em um dia claro de Sol ao meio-dia. O céu azul e o solo gramado são obtidos pela definição do nó *Background*.

As árvores são instâncias do protótipo externo ARVORE definido no arquivo externo *arvore.wrl*. A galeria de arte e todos os detalhes de seus interior são definidos no arquivo *galeria.wrl* que é incluído no mundo pelo uso do comando Inline. O monumento que contêm a logomarca do GTA girando sobre um pedestal é implementado de forma similar incorporando o arquivo *logo_gta.wrl*. Todo o mundo virtual foi criado considerando-se a unidade de distância como metro para efeito de manutenção de escala entre os diversos objetos.

IV.1.2 Definição do Avatar

A geometria do avatar é definida utilizando-se um nó LOD. Como visto na Seção III.1.2 pode-se com o uso deste nó definir níveis de detalhe diferentes a serem utilizados para intervalos de distância diferentes. O avatar é definido em 3 níveis de detalhe. Para o intervalo de 0 a 40 metros, é definido uma versão mais detalhada do avatar, com o corpo inteiro. Para o intervalo de 40 a 80 metros, há uma versão simplificada, sem braços e pernas, e após 80 metros o avatar torna-se invisível, saindo do alcance de visão. Na Figura IV.3 pode-se ver o avatar em sua versão inteira e na simplificada. Assim, quando avatares afastam-se ou aproximam-se uns dos outros, seus usuários vêem-se menos ou mais detalhadamente, respectivamente.

O interpretador VRML pode, assim, otimizar a modelagem do cenário virtual apresentado, pois os avatares dos demais usuários não precisam ser totalmente modelados em função da distância entre estes e o usuário conectado.

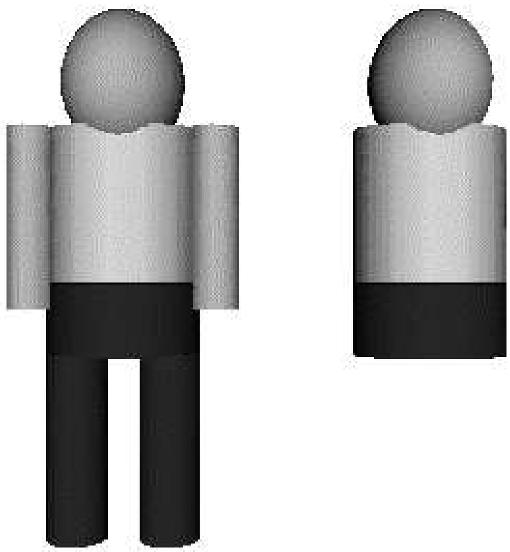

Figura IV.3: Geometria do Avatar em Dois Níveis de Detalhe.

IV.1.3 Monitorando as Ações do Usuário

Há um nó sensor de proximidade (*ProximitySensor*) chamado RASTREADOR que por sua definição engloba todo a área de interesse do mundo virtual. Um nó *Script* chamado RASTREADOR_SCRIPT recebe as alterações de posição e orientação do usuário detectadas pelo nó sensor de proximidade e armazena estas alterações em dois campos de sua definição. O campo *mustEvaluate* do nó *Script* é mantido em verdadeiro. Assim as alterações de posição ou orientação do usuário são imediatamente atualizadas nos respectivos campos visando diminuir a latência entre a mudança de estado do avatar e a chegada desta informação aos outros participantes. Posteriormente, estes campos serão lidos para o envio das atualizações aos demais usuários.

Para o mundo virtual as únicas ações possíveis para o usuário são sua mudança de posição ou orientação. Outras ações como pedido de desconexão ou envio de mensagens aos outros participantes são de responsabilidade da interface gráfica.

IV.2 A Interface Gráfica com o Usuário

A interface gráfica com o usuário é feita por meio de um *applet* Java. Assim que o participante aciona o *applet*, é apresentada a ele a imagem da Figura IV.4. Esta interface gráfica desenvolvida com componentes do *Abstract Windows Toolkit* (AWT) possui a amigabilidade necessária para o suporte a navegação e a troca de mensagens entre os participantes. A execução do *applet* é realizada pelo comando:

```
appletviewer avd_gta.html
```

O arquivo *avd_gta.html* invoca o arquivo *AVD_GTA.class*, que contêm os *bytecodes* referentes ao *applet*, e passa como parâmetro o nome do arquivo que contém a descrição do cenário virtual, no caso *mundo.wrl*.

Há vários componentes nesta interface. A mensagem de boas vindas é implementada por um Rótulo (*Label*). Os outros componentes são descritos a seguir.

IV.2.1 Janela para o Mundo Virtual

A visão que o usuário tem do mundo virtual é um objeto do tipo Painel (*Panel*), que foi associado a algumas classes do pacote *Liquid Reality* para a interpretação da descrição do mundo virtual. Esta janela funciona como um interpretador VRML 2.0 incorporado ao aplicativo.

IV.2.2 Botões de Comando

Existem quatro botões na interface gráfica que permitem ao usuário as ações descritas na Tabela IV.1.

O botão Conectar faz surgir uma nova janela, mostrada na Figura IV.5, que questiona o usuário sobre seu nome e a máquina na qual o servidor está executando.

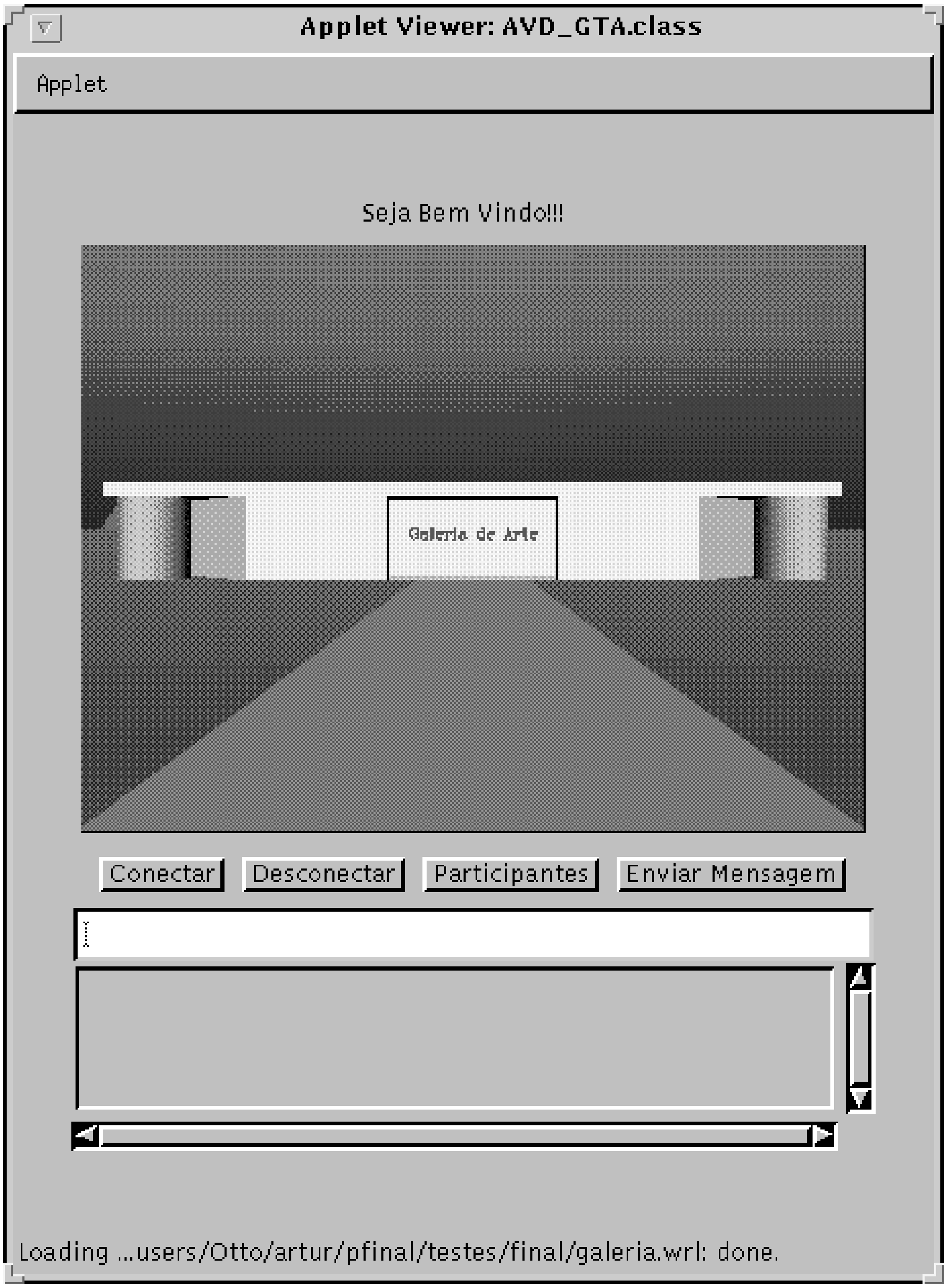

Figura IV.4: Imagem Inicial da Aplicação.

Tabela IV.1: Botões da Interface Gráfica.

Botão	Ação Associada
Conectar	Estabelece a conexão com o servidor
Desconectar	Desfaz a conexão com o servidor
Participantes	Fornece o nome dos usuários conectados
Enviar Mensagem	Envia aos outros usuários a mensagem especificada

Figura IV.5: Janela para Conexão.

IV.2.3 Texto

Há um campo de texto (*TextField*), que permite ao usuário editá-lo para escrever as mensagens a serem enviadas aos outros participantes. Também há uma área de texto (*TextArea*), não-editável, que mostra as mensagens destinadas ao usuário, tanto as oriundas do aplicativo quanto de outros usuários. Utilizando o recurso da troca de mensagens, o usuário pode manter-se em contato com os demais participantes favorecendo a colaboração entre todos na exploração do ambiente virtual apresentado.

IV.3 Filtragens de Dados Utilizadas

Na Seção II.1.2 foram discutidos diversos aspectos pertinentes à ambientes virtuais distribuídos. Entre eles, foram relatadas técnicas de filtragem de dados que poderiam ser aplicadas para reduzir a transmissão de dados irrelevantes para determinados usuários.

Neste trabalho foram incorporadas as técnicas de filtragem por distância e por oclusão da maneira descrita a seguir.

IV.3.1 Filtragem por Distância

Quando um usuário envia uma atualização de sua posição ou orientação ao servidor, este verifica a distância atual entre o usuário que moveu-se e os demais. Caso esta distância seja superior a um determinado valor, o servidor não repassa a informação de atualização ao usuário distante. Considera-se que eles estão fora do alcance de visão um do outro. Na implementação, este limiar ficou estabelecido em 100 metros.

IV.3.2 Filtragem por Oclusão

No mundo virtual implementado há um espaço confinado, a galeria de arte. Quando um usuário está no interior da galeria, ele não precisa receber atualizações sobre usuários que estejam no espaço aberto e estes não precisam das informações do usuário atualmente no espaço confinado.

Ao receber uma atualização de posição ou orientação de um usuário, o servidor verifica quais usuários estão no mesmo espaço, aberto ou confinado, que ele. As informações de atualização não são enviadas a usuários que estejam em um espaço distinto do transmissor da atualização.

IV.3.3 Uso Combinado das Filtragens

As filtragens descritas acima são utilizadas de forma combinada. É necessário atender a ambos os requisitos de filtragem para que as atualizações de um usuário sejam recebidas por outro. Resumindo, ambos os usuários devem estar no mesmo espaço, aberto ou confinado, e ter uma distância entre si inferior ao limiar estabelecido para receberem atualizações mútuas.

O uso das filtragens por distância e por oclusão diminui o volume de dados transmitidos dependendo do comportamento dos usuários no mundo virtual. Todos sempre transmitem suas atualizações ao servidor, porém este somente repassa as informações de atualização pertinentes ao usuário destino.

IV.4 Gerenciamento dos Usuários

O programa servidor, implementado por *Servidor.class*, aguarda permanentemente o pedido de conexão de novos clientes. Quando um usuário envia ao servidor um pedido de conexão, este o aceita, abre um *socket* ao cliente e inicia uma nova *thread* para cada novo cliente. A esta nova *thread* é repassado o *socket* para sua inicialização. Esta *thread* dedicada a cada cliente é implementada pela classe *Intermediario.class* e é responsável por toda intermediação de informação entre os clientes. A cada novo usuário é atribuído um número seqüencial de identificação chamado de *id*.

O sistema funciona de forma distribuída com as estações conectando-se a um servidor central que faz a filtragem e distribuição das informações trocadas entre os usuários participantes. Cada cliente tem a base de dados do mundo virtual replicada em sua estação e somente dados de atualização de posição ou orientação, aviso de conexão e desconexão são trocados entre os participantes.

A Figura IV.6 exemplifica o processo de troca de informações entre os clientes conectados. Quando o usuário A envia uma mensagem ao servidor, ela é tratada pela *thread* associada ao usuário A. Esta, de acordo com a natureza da mensagem, filtra-a e repassa aos outros usuários conectados (B, C ou nenhum deles) através das suas *threads*. Ou seja, todos os clientes comunicam-se exclusivamente com a *thread* a qual ele está associado no servidor.

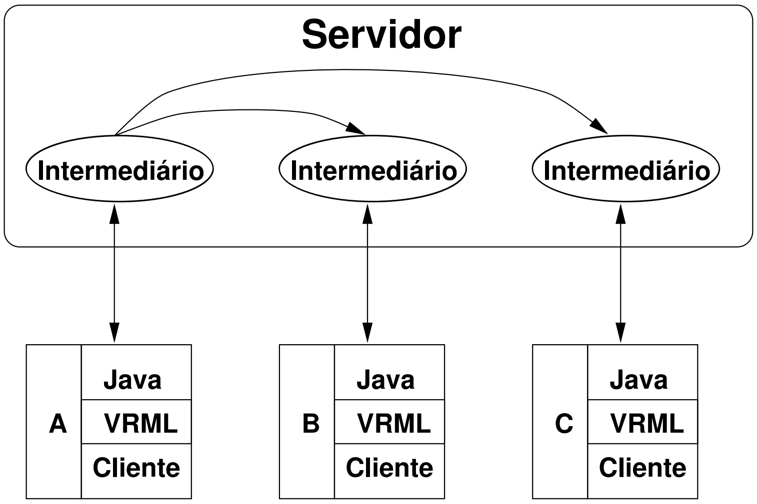

Figura IV.6: Estrutura de Comunicação.

As mensagens que o cliente pode enviar ao servidor e suas funções estão descritas na Tabela IV.2. Deve-se lembrar que a posição de um avatar é descrita pelas coordenadas (x, y, z) e sua orientação por parâmetros (x, y, z, r), onde r é um valor que deve ser multiplicado por cada um dos valores x, y e z para obter-se o valor do ângulo em radianos aplicado em cada eixo x, y e z, respectivamente.

Por outro lado, a *thread* Intermediário envia informações a seu cliente através das mensagens descritas na Tabela IV.3.

Tabela IV.2: Mensagens do Cliente ao Intermediário.

Comando	Parâmetros	Função
GIRE	nome x y z r	Comunica que o usuário girou para a nova orientação (x, y, z, r).
MOVA	nome x y z	Comunica que o usuário moveu-se para a nova posição (x, y, z).
MSG	nome mensagem	Envia aos demais usuários uma mensagem.
NOVO	id nome x y z x y z r	Comunica aos usuários já conectados a nossa conexão recém estabelecida, junto com nosso nome, id, posição e orientação.
SAI	nome id	Faz o pedido de desconexão do cliente ao servidor.
SOU	id nome x y z x y z r	Comunica ao novo usuário a nossa presença com nosso nome, id, posição e orientação.

Tabela IV.3: Mensagens do Intermediário ao Cliente.

Código	Parâmetros	Função
100	sem parâmetros	Requisição do nome do cliente.
150	id	Confirma a aceitação do cliente e informa o seu id.
200	id nome x y z x y z r	Informa que um novo usuário entrou no ambiente virtual.
250	nome x y z x y z r	Informa sobre um usuário já existente quando o usuário conectou-se.
300	nome x y z	Informa que o usuário chamado nome moveu-se para (x, y, z).
350	nome x y z r	Informa que o usuário chamado nome girou para (x, y, z, r).
400	nome mensagem	Usuário nome enviou mensagem
900	sem parâmetros	Confirmação de desconexão.
950	nome fatorDecremento	Avisa que usuário chamado nome retirou-se do sistema e que o id do usuário deve ser decrementado pelo valor fatorDecremento.

IV.5 Percepção dos Outros Usuários

Todos os usuários conectados percebem a movimentação dos outros, desde que estes estejam em sua área de interesse, ou seja, estejam no mesmo espaço a uma distância inferior ao limiar de visão. As mensagens enviadas são repassadas a todos os usuários independentemente de sua localização no mundo virtual.

Quando conecta-se ao servidor, o usuário primeiro envia seus dados (nome, posição e orientação) a seu intermediário para que este informe aos outros participantes a sua presença e características. Por sua vez, os usuários que já estavam conectados respondem ao aviso de novo usuário enviando seus dados exclusivamente ao novo usuário. Ao final deste processo todos os usuários, tanto o novo quanto os já conectados, possuem uma visão consistente do mundo virtual. Para cada aviso de novo cliente ou de usuário pré-existente o aplicativo replica a geometria do avatar e a renomeia com o nome do novo usuário. Assim, todas as ações realizadas por um usuário serão refletidas pelo avatar nomeado com seu nome.

Processo semelhante ocorre quando um usuário faz um pedido de desconexão do sistema. Seu intermediário confirma a desconexão ao usuário solicitante, avisa aos demais usuários para que estes possam retirar o avatar do usuário ausente do mundo virtual e finalmente o *thread* intermediária destrói-se.

Capítulo V

Conclusão

Este trabalho tratou do desenvolvimento de um ambiente virtual baseado no uso combinado das linguagens Java e VRML. Este ambiente permite que um número limitado de usuários interaja entre si no cenário virtual apresentado. A interação ocorre pela percepção da presença e ações dos outros usuários participantes no sistema e pela possibilidade de troca de mensagens com estes.

A topologia adotada, com um servidor central, atendeu as necessidades do sistema para a proposta de um número limitado de participantes simultâneos. No entanto, ficou claro que esta topologia não é escalável a um maior número de usuários devido à sobrecarga de tráfego no servidor. As técnicas de filtragem de dados adotadas, por distância e por oclusão, diminuem o tráfego oriundo do servidor, dependendo do comportamento dos usuários conectados, mas em nada contribuem para a redução do tráfego que chega ao servidor. Uma alternativa é a utilização de uma configuração sem o servidor central, com os usuários comunicando-se diretamente entre si. No entanto, esta abordagem impossibilita a filtragem total dos dados espacialmente, pois deveriam existir atualizações periódicas para que os usuários controlem a posição dos outros participantes para manter a filtragem consistente. Outra abordagem possível estudada defende a manutenção de um servidor central para o controle do mundo virtual: conexão de novos usuários, concorrência por recursos comuns e desconexão. Estas informações são enviadas por meio de protocolos confiáveis como TCP. As atualizações de posição e orientação são feitas por mensagens enviadas diretamente aos outros usuários, por meio de protocolos não confiáveis como UDP, que compartilhem a mesma área de interesse. Havendo a perda de um pacote, outra atualização mais recente mantém a consistência do mundo virtual. O controle da área de interesse de cada usuário é feito pelo servidor central, que atualiza em cada usuário quais são os outros participantes que devem receber mensagens de atualização. O uso de protocolos *multicast* para difusão das informações de atualização entre os usuários também contribuiria para a redução do tráfego imposto à rede de comunicação.

Foi possível verificar que o uso da linguagem VRML facilita o acréscimo de novos objetos ao mundo virtual apresentado podendo-se, inclusive, modularizar o projeto e a criação do ambiente virtual. Facilita-se, assim, a manutenção e a expansão do mundo a ser proposto para exploração. No decorrer do projeto pode-se constatar que a utilização de Java como suporte à linguagem VRML expande enormemente o potencial de aplicação desta última. O nó *Script* permite esta integração e abre aos mundos virtuais descritos em VRML 2.0 uma gama bastante grande de aplicações que exijam interatividade que não seriam implementáveis com o uso do mecanismo de roteamento de eventos oferecido pelo VRML puro. Novos nós podem ser criados, expandindo a linguagem, com seu comportamento determinado pela programação desenvolvida no programa *script* associado ao nó *Script*.

O pacote *Liquid Reality*, além de um interpretador VRML 2.0, oferece classes que permitem a manipulação dinâmica do arquivo VRML. Desta forma, obtem-se um poder maior de interatividade com o usuário, pois pode-se alterar o grafo de cena descrito pelo arquivo em tempo de execução. Na aplicação desenvolvida é utilizado este recurso, por exemplo, para criar-se a réplica do avatar que representará o novo usuário recém-conectado.

A aplicação desenvolvida mostrou a realidade virtual como uma nova técnica de interface homem-máquina. Exemplificou-se o acesso pelos usuários a um ambiente virtual compartilhado e algumas das maneiras pelas quais os usuários podem intergir com o cenário virtual que os envolve e com os demais participantes. A interface gráfica que complementa a visualização do mundo virtual com os comandos controle e espaços para a troca de mensagens mostrou-se amigável para com o usuário. O acréscimo de técnicas de filtragens de dados melhorou o desempenho do sistema. Uma vantagem do sistema é a sua implementação em máquinas não dedicadas, diferentemente de grandes sistemas de simulação que exigem terminais dedicados de alto custo.

A abordagem adotada possui grande potencial em áreas como trabalho cooperativo, em que alguns usuários podem compartilhar a visão de um ambiente modelado em realidade virtual, trocando comentários e idéias sobre o espaço no qual todos estão inseridos. A mesma estrutura pode ser aplicada, modificando-se o cenário virtual, para aplicações em outras áreas como entretenimento e simulação.

V.1 Trabalhos Futuros

Para trabalhos futuros, pode-se sugerir:

- utilizar técnicas de *multicast* do servidor aos clientes, visando a redução do tráfego enviado;
- estudar a eliminação do servidor central e como seriam feitas as filtragens devidas sem a sua presença;
- incorporar outras técnicas para diminuição do tráfego, como *dead-reckoning*;
- acrescentar a possibilidade de concorrência por um recurso comum no ambiente virtual e assim motivar a introdução de técnicas para manutenção de consistência;
- introduzir efeitos sonoros no ambiente de forma a aumentar o nível de imersão experimentado pelo usuário e aumentar o seu grau de percepção de outros participantes a sua volta.

Referências Bibliográficas

[1] J. F. Morie, "Inspiring the future: Merging mass communication, art, entertainment and virtual environments", *Computer Graphics*, vol. 28, no. 2, pp. 135–138, fevereiro 1994.

[2] R. Azuma, "Tracking requirements for augmented reality", *Communications of the ACM*, vol. 36, no. 7, pp. 50–51, julho 1993.

[3] W. Mackay, G. Velay, K. Carter, C. Ma e D. Pagani, "Augmenting reality: Adding computacional dimensions to paper", *Communications of the ACM*, vol. 36, no. 7, pp. 96–97, julho 1993.

[4] R. Zajtchuk e R. M. Satava, "Medical applications of virtual reality", *Communications of the ACM*, vol. 40, no. 9, pp. 63–64, setembro 1997.

[5] D. Strickland, L. Hodges, M. North e S. Weghorst, "Overcoming phobias by virtual exposure", *Communications of the ACM*, vol. 40, no. 8, pp. 34–39, agosto 1997.

[6] R. Macredie, S. J. E. Taylor, X. Yu e R. Keeble, "Virtual reality and simulation: an overview", in *Proc. of the 1996 Winter Simulation Conference*, pp. 669–674, 1996.

[7] S. Bayarri, M. Fernandez e M. Perez, "Virtual reality for driving simulation", *Communications of the ACM*, vol. 39, no. 5, pp. 72–76, maio 1996.

[8] C. R. Karr, D. Reece e R. Franceschini, "Synthetic soldiers", *IEEE Spectrum*, vol. 34, no. 3, pp. 39–45, março 1997.

[9] S. Bryson, "Virtual reality in scientific visualization", *Communications of the ACM*, vol. 39, no. 5, pp. 62–71, maio 1996.

[10] S. R. Ellis, "What are virtual environments?", *IEEE Computer Graphics and Applications*, vol. 14, no. 1, pp. 17–22, janeiro 1994.

[11] R. Braham e R. Comerford, "Sharing virtual worlds", *IEEE Spectrum*, vol. 34, no. 3, pp. 18–19, março 1997.

[12] R. B. de Araújo, *Especificação e Análise de um Sistema Distribuído de Realidade Virtual.* Tese de Doutorado, Escola Politécnica da USP, São Paulo, SP, junho 1996.

[13] R. C. Waters e J. W. Barrus, "The rise of shared virtual environments", *IEEE Spectrum*, vol. 34, no. 3, pp. 20–25, março 1997.

[14] M. R. Stytz, "Distributed virtual environments", *IEEE Computer Graphics and Applications*, vol. 16, no. 3, pp. 19–31, maio 1996.

[15] B. Roehle, "Channeling the data flood", *IEEE Spectrum*, vol. 34, no. 3, pp. 32–38, março 1997.

[16] R. C. Teixeira e O. C. M. B. Duarte, "Requisitos de ambientes virtuais distribuídos de larga escala", em *Anais do* 1° *Workshop de Realidade Virtual, WRV'97*, São Carlos, SP, pp. 26–35, novembro 1997.

[17] R. B. de Araújo, M. A. M. S. de Oliveira, R. P. Swensson, R. R. Cronemberger, R. C. Colossi, P. M. Eleutério e W. da Silva Bila, "Exploração de ambientes virtuais distribuídos de grande porte", em *Anais do* 1° *Workshop de Realidade Virtual, WRV'97*, São Carlos, SP, pp. 11–18, novembro 1997.

[18] J. Locke, "An introduction to the Internet networking environment and SIMNET/DIS", relatório técnico, Naval PostGraduate School, agosto 1995.

[19] M. R. Macedonia, M. J. Zyda, D. R. Pratt, D. P. Brutzman e P. T. Barham, "Exploiting reality with multicast groups", *IEEE Computer Graphics and Applications*, vol. 15, no. 5, pp. 38–45, setembro 1995.

[20] M. R. Macedonia e M. J. Zyda, "A taxonomy for networked virtual environments", *IEEE Multimidia*, vol. 4, no. 1, janeiro 1997.

[21] O. Hagsand, "Interactive multiuser VEs in DIVE system", *IEEE Multimidia*, vol. 3, no. 1, pp. 30–39, janeiro 1996.

[22] C. Greenhalgh e S. Benford, "MASSIVE: A collaborative virtual environment for teleconferencing", *ACM Transactions on Computer-Human Interaction*, vol. 2, no. 3, pp. 239–261, setembro 1995.

[23] M. R. Macedonia, D. R. Pratt e M. J. Zyda, "NPSNET: A network software architecture for large-scale virtual environments", *Presence: Teleoperators and Virtual Environments*, vol. 3, no. 4, pp. 265–287, 1994.

[24] J. W. Barrus, R. C. Waters e D. B. Andersen, "Locales: Supporting large multiuser virtual environments", *IEEE Computer Graphics and Applications*, vol. 16, no. 6, pp. 50–57, novembro 1996.

[25] G. Bell, A. Parisi e M. Pesce, *The Virtual Reality Modeling Language - Version 1.0 Specification*, maio 1995.

[26] G. Bell, R. Carey e C. Marrin, *The Virtual Reality Modeling Language Specification - Version 2.0*, agosto 1996.

[27] A. L. Ames, D. R. Nadeau e J. L. Moreland, *VRML 2.0 Sourcebook*. New York, NY: John Wiley & Sons, 2a. ed., 1997.

[28] J. Gosling, B. Joy e G. L. Steele, *The Java Language Specification*. Reading, MA: Addison-Wesley Publishing Company, 1a. ed., 1996.

[29] M. Campione e K. Walrath, *The Java Tutorial - Object Oriented Programming for the Internet*. Reading, MA: Addison-Wesley Publishing Company, 1a. ed., 1996.

[30] P. Naughton e H. Schildt, *Java: The Complete Reference*. Berkeley, CA: McGraw-Hill, 1997.

[31] R. Lea, K. Matsuda e K. Miyashita, *Java for 3D and VRML Worlds*. Indianapolis, IN: New Riders Publishing, 1a. ed., 1996.

[32] F. S. Tamiosso, A. B. Raposo, L. P. Magalhães e I. M. Ricarte, "Building interactive animations using VRML and Java", em *X Simpósio Brasileiro de Computação Gráfica e Processamento de Imagens, SIBGRAPI'97*, Campos do Jordão, SP, outubro 1997.

[33] A. Descartes, "Interfacing Java and VRML", in *UK Java Developer's Conference*, novembro 1996.

[34] DimensionX, "Liquid reality". http://www.microsoft.com/dimensionx/lr/.

Apêndice A

Listagens

Aqui estão listadas as versões finais de todos os programas escritos durante o desenvolvimento do projeto. Primeiramente estão divididos entre os programas que compõe a interface do cliente e os que devem ficar na máquina servidora. Os arquivos listados são códigos fonte de Java, arquivos VRML 2.0 ou HTML.

A.1 Cliente

A.1.1 AVD_GTA.java

Este é o código fonte do *applet* que implementa a interface com o usuário e faz a comunicação com o programa servidor.

```
// AVD_GTA.java
import java.applet.*;
import java.awt.*;
import java.io.*;
import java.net.*;
import java.util.*;
```

A.1 Cliente

```
// Inclui classes do pacote Liquid Reality da DimensionX
import dnx.lr.*;
import dnx.lr.app.*;
import dnx.lr.field.*;
import dnx.lr.node.*;

public class AVD_GTA extends Applet implements Runnable {
    // Dados do usuario
    String conta = null;

    // Numero de identificacao no servidor
    int id;

    // Armazena os nomes dos participantes
    Vector participantes = new Vector();

    // thread que liga-se com a rede
    Thread threadRede = null;

    // Definicao dos sockets
    Socket vrmlSocket = null;
    DataInputStream networkIStream = null;
    PrintStream networkOStream = null;

    // Posicao atual e previa do avatar no mundo
    SFVec3f posicaoAtual = null;
    SFVec3f posicaoPrevia = null;

    // Orientacao atual e previa do avatar no mundo
    SFRotation orientacaoAtual = null;
    SFRotation orientacaoPrevia = null;
```

```
// Containers da interface grafica (AWT)
AppletDocument documento;
Scene cena = null;
SceneContents componentes = null;

// Componentes da interface grafica (AWT)
Label Apresentacao = new Label( "Seja Bem Vindo!!!" );

// Janela para mundo virtual, painel de botoes e
// campos de texto.
painelVRML PAINELVRML = new painelVRML();
Panel painelComandos = new Panel();
TextField mensagemUsuario;
TextArea mensagemRede;

// Inicializacao do Applet - Construcao da Interface Grafica
public void init() {
    // Abre e verifica a sintaxe do arquivo VRML passado
    // como parametro para o applet.
    try {
        componentes = SceneContents.read(new ReadAction (
            new URL (getDocumentBase(), getParameter("vrml"))));
        cena = new Scene (componentes);
    } catch (VRMLSyntaxException e) {
        System.out.println ("Erro de sintaxe em : " + e);
        return;
    } catch (java.io.IOException e) {
        System.out.println ("Erro lendo : " + e);
        return;
    }
```

```
// Cria o documento e a visualizacao do cenario VRML
documento = new AppletDocument(this,cena);
View3D visao = new View3D( documento );
visao.setRenderQuality( View3D.QUALITY_SMOOTH_SHADED );
BrowserEngineAdapter adaptador =
    new BrowserEngineAdapter( visao );
documento.addFrameUpdateListener( adaptador );
documento.addSceneObserverEventListener( adaptador );
visao.addView3DListener( adaptador );

// Cria o 'layout' do applet
GridBagLayout gridbag = new GridBagLayout();
GridBagConstraints limites = new GridBagConstraints();

// Janela VRML
PAINELVRML.resize( 410, 310 );
PAINELVRML.add( visao.getComponent() );
PAINELVRML.layout();

// Definicao dos botoes
Button botaoConexao = new Button( "Conectar" );
Button botaoDesconexao = new Button( "Desconectar" );
Button botaoQuem = new Button( "Participantes" );
Button botaoMSG = new Button( "Enviar Mensagem" );

painelComandos.add( botaoConexao );
painelComandos.add( botaoDesconexao );
painelComandos.add( botaoQuem );
painelComandos.add( botaoMSG );
```

```
// Janela para envio de mensagens aos outros usuarios
mensagemUsuario = new TextField ( 42 );

// Janela para exibicao das mensagens recebidas
mensagemRede = new TextArea ( 5, 40 );
mensagemRede.setEditable(false);

// Organizar o layout da janela
setLayout( gridbag );

// Acrescenta o titulo da tela
limites.gridx = 0;
limites.gridy = 0;
gridbag.setConstraints( Apresentacao, limites );
add( Apresentacao );

// Acrescente a 'janela' para o cenario VRML
limites.gridx = GridBagConstraints.REMAINDER;
limites.gridwidth = GridBagConstraints.REMAINDER;
limites.gridy = GridBagConstraints.RELATIVE;
limites.fill = GridBagConstraints.HORIZONTAL;
gridbag.setConstraints( PAINELVRML, limites );
add( PAINELVRML );

// Acrescenta o painel de comandos com os botoes
limites.gridx = GridBagConstraints.REMAINDER;
limites.gridwidth = GridBagConstraints.REMAINDER;
limites.gridy = GridBagConstraints.RELATIVE;
limites.fill = GridBagConstraints.HORIZONTAL;
gridbag.setConstraints( painelComandos, limites );
add( painelComandos );
```

```
        // Acrescenta o campo para comunicacao entre usuarios
        gridbag.setConstraints( mensagemUsuario, limites );
        add( mensagemUsuario );

        // Acrescenta a area de mensagens para o usuario
        gridbag.setConstraints( mensagemRede, limites );
        add( mensagemRede );

        // Dimensiona a 'janela' para o cenario VRML
        visao.getComponent().reshape( 0, 0, 400, 300 );

        // Inicia o documento
        documento.start();
        repaint();
    }

    // Rotina principal
    public void run() {
        System.out.println( "AVD_GTA : pronto" );
        Apresentacao.setText( conta );
        while( vrmlSocket != null && networkIStream != null ) {
        String linhaAtual = null;
        try {
        while( networkIStream.available() > 0 ) {
            linhaAtual = networkIStream.readLine();
            System.out.println( "Linha lida: " + linhaAtual );

            // Confirmacao de desconexao
            if ( linhaAtual.regionMatches( 0, "900", 0, 3 ) ) {
                System.err.println("Recebi 900
```

```
            - Fechando o socket.");
        vrmlSocket.close();
        vrmlSocket = null;
        networkIStream = null;
        networkOStream = null;

        // Procurando Avatar
        Node rootNode = cena.getRootNode();
        Node achado = null;
        TransformNode noAvatar = null;

        // Removendo os avatares dos outros usuarios
        for ( int i = 0; i < participantes.size(); i++ ) {
            // Vasculha o cenario
            achado = cena.lastNodeWithName(
                (String)participantes.elementAt(i) );
            if ( achado instanceof TransformNode ) {
                noAvatar = (TransformNode)achado;
                System.err.println("No Avatar: "
                    + noAvatar.getName() );
                // Remove o Avatar do cenario
                System.out.println( "Retirando "
                    + noAvatar.getName()
                        + " do nosso sistema." );
                ((GroupNode)rootNode).children.remove(
                    noAvatar );
            }
        }
        participantes.removeAllElements();
        break;
    } else {
```

```
// Pedido de envio de conta (nome)
if ( linhaAtual.regionMatches( 0, "100", 0, 3 ) ) {
    mensagemRede.appendText( linhaAtual + "\n" );
    System.err.println( "Recebi 100
        - Requisitando conta.");
    networkOStream.println( "Conta: " + conta );
} else {

// Confirmacao de aceitacao
if ( linhaAtual.regionMatches( 0, "150", 0, 3 ) ) {
    mensagemRede.appendText( linhaAtual + "\n" );
    int index = linhaAtual.indexOf(":", 0 );
    int indexPost = linhaAtual.indexOf( " ", index );
    Integer idInteger = new Integer (
        linhaAtual.substring( index + 1, indexPost ) );
    id = idInteger.intValue();
    System.err.println( "Recebi 150 -
        Fomos aceitos. Somos o usuario " + id );
    networkOStream.println( "NOVO " + id + " "
        + conta + " " + posicaoAtual.toString()
            + " " + orientacaoAtual.toString() );
} else {

// Novo usuario no sistema
if ( linhaAtual.regionMatches( 0, "200", 0, 3 ) ) {
    mensagemRede.appendText( linhaAtual + "\n" );
    System.out.println( "Criando novo no’ de avatar
        e inserindo-o no cenario." );
    int index = linhaAtual.indexOf( " ", 4 );
    Double iDouble = new Double(
```

```
    linhaAtual.substring( 4, index ) );
int id = iDouble.intValue();
// Informa nossa existencia para o novo usuario
networkOStream.println( "SOU " + id + " "
    + conta + " " + posicaoAtual.toString()
        + " " + orientacaoAtual.toString() );

// Procurando Avatar
Node rootNode = cena.getRootNode();
Node achado = null;
TransformNode noAvatar = null;
// Vasculha o cenario
achado = cena.lastNodeWithName( "AVATAR" );
if ( achado instanceof TransformNode ) {
    noAvatar = (TransformNode)achado;
    System.err.println("No Avatar: "
        + noAvatar.getName() );
    // Replica geometria do avatar
    TransformNode novoAvatar =
        (TransformNode)noAvatar.deepClone();

    int indexNome = linhaAtual.indexOf(
        " ", index + 1 );
    System.err.println( "Nome do Avatar: " +
        linhaAtual.substring(index+1, indexNome));

    // Batiza o novo avatar com a conta de seu dono
    novoAvatar.setName( linhaAtual.substring(
        index + 1, indexNome ));

    // Registra o nome do usuario que conectou-se
```

```
participantes.insertElementAt(
new String( linhaAtual.substring(
    index + 1, indexNome)),participantes.size());

int indexX = linhaAtual.indexOf(
    " ", indexNome + 1 );
Double xDouble = new Double(
    linhaAtual.substring(indexNome+1,indexX));
float newX = xDouble.floatValue();

int indexY = linhaAtual.indexOf(
    " ", indexX + 1 );
Double yDouble = new Double(
    linhaAtual.substring(indexX+1,indexY));
float newY = yDouble.floatValue();

int indexZ = linhaAtual.indexOf(
    " ", indexY + 1 );
Double zDouble = new Double(
    linhaAtual.substring(indexY+1,indexZ));
float newZ = zDouble.floatValue();

int indexOX = linhaAtual.indexOf(
    " ", indexZ + 1 );
Double xODouble = new Double(
    linhaAtual.substring(indexZ+1,indexOX));
float newOX = xODouble.floatValue();

int indexOY = linhaAtual.indexOf(
    " ", indexOX + 1 );
Double yODouble = new Double(
```

```
                linhaAtual.substring(indexOX+1,indexOY));
            float newOY = yODouble.floatValue();

            int indexOZ = linhaAtual.indexOf(
                " ", indexOY + 1 );
            Double zODouble = new Double(
                linhaAtual.substring(indexOY+1,indexOZ));
            float newOZ = zODouble.floatValue();

            Double rODouble = new Double(
                linhaAtual.substring( indexOZ + 1,
                    linhaAtual.length() ) );
            float newOR = rODouble.floatValue();

            // Posiciona e orienta o novo Avatar
            novoAvatar.translation.setValue(newX,newY,newZ );
            novoAvatar.rotation.setValue(
                newOX, newOY, newOZ, newOR );

            // Acrescenta o novo Avatar ao cenario
            ((GroupNode)rootNode).children.addValue(
                novoAvatar );
        }
    } else {

    // Algum outro usuario moveu-se
    if ( linhaAtual.regionMatches( 0, "300", 0, 3 ) ) {
        mensagemRede.appendText( linhaAtual + "\n" );
        int indexNome = linhaAtual.indexOf( " ", 4 );
        String nomeAvatar = new String (
            linhaAtual.substring( 4, indexNome ) );
```

```
int indexX = linhaAtual.indexOf(" ",indexNome+1);
Double xDouble = new Double(
    linhaAtual.substring( indexNome + 1, indexX ) );
float newX = xDouble.floatValue();

int indexY = linhaAtual.indexOf( " ", indexX + 1 );
Double yDouble = new Double(
    linhaAtual.substring( indexX + 1, indexY ) );
float newY = yDouble.floatValue();

Double zDouble = new Double(
    linhaAtual.substring( indexY + 1,
        linhaAtual.length() ) );
float newZ = zDouble.floatValue();

System.err.println( "Avatar: " + nomeAvatar
    + " moveu-se para " + newX +
        ", " + newY + ", " + newZ );

// Procurando Avatar
Node rootNode = cena.getRootNode();
Node achado = null;
TransformNode noAvatar = null;
// Vasculha o cenario
achado = cena.lastNodeWithName ( nomeAvatar );
if ( achado instanceof TransformNode ) {
    noAvatar = (TransformNode)achado;
    System.err.println( "Mover : "
        + noAvatar.getName() );
    // Reposiciona o Avatar
```

```
            noAvatar.translation.setValue(newX,newY,newZ);
        }
    } else {

    // Algum outro usuario girou
    if ( linhaAtual.regionMatches ( 0, "350", 0, 3 ) ) {
        mensagemRede.appendText( linhaAtual + "\n" );
        int indexNome = linhaAtual.indexOf( " ", 4 );
        String nomeAvatar = new String (
            linhaAtual.substring( 4, indexNome ) );

        int indexX = linhaAtual.indexOf(" ",indexNome+1);
        Double xDouble = new Double(
            linhaAtual.substring( indexNome + 1, indexX ) );
        float newX = xDouble.floatValue();

        int indexY = linhaAtual.indexOf( " ", indexX + 1 );
        Double yDouble = new Double(
            linhaAtual.substring( indexX + 1, indexY ) );
        float newY = yDouble.floatValue();

        int indexZ = linhaAtual.indexOf( " ", indexY + 1 );
        Double zDouble = new Double(
            linhaAtual.substring( indexY + 1, indexZ ) );
        float newZ = zDouble.floatValue();

        Double rDouble = new Double(
            linhaAtual.substring( indexZ + 1,
                linhaAtual.length() ) );
        float newR = rDouble.floatValue();
```

```
        System.err.println( "Avatar: " + nomeAvatar
            + " moveu-se para " + newX +
                ", " +newY +", " +newZ +", r = " +newR);

        // Procurando Avatar
        Node rootNode = cena.getRootNode();
        Node achado = null;
        TransformNode noAvatar = null;
        // Vasculha o cenario
        achado = cena.lastNodeWithName ( nomeAvatar );
        if ( achado instanceof TransformNode ) {
            noAvatar = (TransformNode)achado;
            System.err.println(
                "Girar : " + noAvatar.getName() );
            // Gira o Avatar
            noAvatar.rotation.setValue(newX,newY,newZ,newR);
        }
    } else {

    // Informacao de usuario anterior quando nos conectamos
    if ( linhaAtual.regionMatches( 0, "250", 0, 3 ) ) {
        mensagemRede.appendText( linhaAtual + "\n" );
        System.out.println( "Criando novo no' de avatar
            e inserindo-o no cenario." );

        // Procurando Avatar
        Node rootNode = cena.getRootNode();
        Node achado = null;
        TransformNode noAvatar = null;
        // Vasculha o cenario
        achado = cena.lastNodeWithName( "AVATAR" );
```

```
        if ( achado instanceof TransformNode ) {
            noAvatar = (TransformNode)achado;
            System.err.println("No Avatar: "
                + noAvatar.getName() );
            TransformNode novoAvatar =
                (TransformNode)noAvatar.deepClone();

            // Batiza o novo avatat com a conta de seu dono
            int indexNome = linhaAtual.indexOf( " ", 4 );
            System.err.println( "Nome do Avatar: "
                + linhaAtual.substring( 4, indexNome ));
            novoAvatar.setName(
                linhaAtual.substring( 4, indexNome ));

            // Registra o nome dos usuario que
            // ja estavam conectados quando cheguei
            participantes.insertElementAt(
            new String(linhaAtual.substring(4,indexNome)),
                participantes.size());

            int indexX = linhaAtual.indexOf(
                " ", indexNome + 1 );
            Double xDouble = new Double(
                linhaAtual.substring( indexNome + 1,indexX));
            float newX = xDouble.floatValue();

            int indexY = linhaAtual.indexOf(
                " ", indexX + 1 );
            Double yDouble = new Double(
                linhaAtual.substring( indexX + 1,indexY));
            float newY = yDouble.floatValue();
```

```
int indexZ = linhaAtual.indexOf( " ",indexY+1);
Double zDouble = new Double(
    linhaAtual.substring( indexY + 1,indexZ));
float newZ = zDouble.floatValue();

int indexOX = linhaAtual.indexOf( " ",indexZ+1);
Double xODouble = new Double(
    linhaAtual.substring( indexZ + 1,indexOX));
float newOX = xODouble.floatValue();

int indexOY = linhaAtual.indexOf( " ",indexOX+1);
Double yODouble = new Double(
    linhaAtual.substring( indexOX + 1,indexOY));
float newOY = yODouble.floatValue();

int indexOZ = linhaAtual.indexOf( " ",indexOY+1);
Double zODouble = new Double(
    linhaAtual.substring( indexOY + 1,indexOZ));
float newOZ = zODouble.floatValue();

Double rODouble = new Double(
    linhaAtual.substring( indexOZ + 1,
        linhaAtual.length() ) );
float newOR = rODouble.floatValue();

// Posiciona e orienta o novo Avatar
novoAvatar.translation.setValue(newX,newY,newZ);
novoAvatar.rotation.setValue(
    newOX, newOY, newOZ, newOR );
```

```
            // Acrescenta o novo Avatar ao cenario
            ((GroupNode)rootNode).children.addValue(
                novoAvatar );
        }
    } else {

    // Informe de desconexao de outro usuario
    if ( linhaAtual.regionMatches( 0, "950", 0, 3 ) ) {
        mensagemRede.appendText( linhaAtual + "\n" );
        int index = linhaAtual.indexOf( " ", 4 );
        String nomeAvatar = new String (
            linhaAtual.substring( 4, index ) );

        Integer fatorInteger = new Integer (
            linhaAtual.substring( index + 1,
                linhaAtual.length() ) );
        id = id - fatorInteger.intValue();
        System.out.println( "Nosso novo id e' : " + id );

        // Pegando o no' do Avatar
        Node rootNode = cena.getRootNode();
        Node achado = null;
        TransformNode noAvatar = null;

        // Vasculha o cenario
        achado = cena.lastNodeWithName( nomeAvatar );
        if ( achado instanceof TransformNode ) {
            noAvatar = (TransformNode)achado;
            System.err.println("No Avatar: "
                + noAvatar.getName() );
            // Remove o Avatar ao cenario
```

```
                System.out.println( "Retirando "
                    + noAvatar.getName()+" do nosso sistema.");
                ((GroupNode)rootNode).children.remove(
                    noAvatar );
            }

            for ( int i = 0; i < participantes.size(); i++) {
                if (((String)participantes.elementAt(i)).equals(
                            noAvatar.getName() ) ) {
                    participantes.removeElementAt(i);
                }
            }
        } else {

        // Recebimento de mensagem
        if ( linhaAtual.regionMatches( 0, "400", 0, 3 ) ) {
            mensagemRede.appendText( linhaAtual + "\n" );
            int index = linhaAtual.indexOf( " ", 4 );
            String avatarRemetente = new String (
                linhaAtual.substring( 4, index ) );
            String msg = new String ( linhaAtual.substring(
                index + 1, linhaAtual.length() ) );
            mensagemRede.appendText(
                avatarRemetente + ": " + msg + "\n");
        }

        } } } } } } } }
    }
    } catch (IOException e) {
        System.err.println("IOException: " + e.toString() );
    }
```

```
// Ate' este ponto, os dados recebidos da rede foram lidos.
// Agora, vai se verificar se o usuario moveu-se e em
// caso afirmativo enviar a atualizacao dos dados.
Node rootNode = cena.getRootNode();
Node achado = null;
ScriptNode noRastreador = null;
float x, y, z;

// Procura no' Rastreador (Script)
achado = cena.lastNodeWithName( "RASTREADOR_SCRIPT" );
if ( achado instanceof ScriptNode ) {
    noRastreador = (ScriptNode)achado;
    // Pegar a posicao atual do Avatar
    posicaoAtual = (SFVec3f)noRastreador.getField(
                   "posicao_avatar" );
    if ( posicaoPrevia != null ) {
        if (!posicaoPrevia.equals( posicaoAtual ) ) {
            System.out.println( "Nos nos movemos.
                Enviando atualizacao." );
            networkOStream.println( "MOVA " + conta
                +" " +id +" " +posicaoAtual.toString());
        }
        posicaoPrevia.copy( posicaoAtual );
    } else {
        posicaoPrevia = new SFVec3f();
        posicaoPrevia.copy( posicaoAtual );
    }

    // O mesmo procedimento para a orientacao do Avatar
    orientacaoAtual = (SFRotation)noRastreador.getField(
```

```
                        "orientacao_avatar" );
            if ( orientacaoPrevia != null ) {
                if ( !orientacaoPrevia.equals( orientacaoAtual ) ) {
                    System.out.println( "Nos giramos.
                        Enviando atualizacao." );
                    networkOStream.println( "GIRE " + conta
                        +" " +id +" " +orientacaoAtual.toString());
                }
                orientacaoPrevia.copy( orientacaoAtual );
            } else {
                orientacaoPrevia = new SFRotation();
                orientacaoPrevia.copy( orientacaoAtual );
            }
        } else {
            System.err.println("Nao e' um no' Script." );
        }

        // Retarda um pouco a 'thread'
        try {
            threadRede.sleep( 100 );
        } catch (InterruptedException e) {
            System.out.println("InterruptedException: "
                + e.toString() );
        }
        } // Fim do while principal
        threadRede = null;
    }     // Fim da funcao run

    // Tratamento dos eventos (pressao nos botoes)
    public boolean handleEvent( Event e ) {
        switch ( e.id ) {
```

```
        case Event.ACTION_EVENT: {
            if ( e.arg.toString().equals( "Conectar" ) ) {
                AVD_GTAconta dialogo = new AVD_GTAconta( this );
                dialogo.show();
                return true;
            } else {

            if ( e.arg.toString().equals( "Desconectar" ) ) {
                mensagemRede.appendText( "Desconectando...\n" );
                if ( vrmlSocket != null ) {
                    networkOStream.println( "SAI "
                        + conta + " " + id );
                }
                return true;
            } else {

            if ( e.arg.toString().equals( "Participantes" ) ) {
                mensagemRede.appendText( "Participantes: "
                    + participantes.toString() + "\n" );
                return true;
            } else {

            if ( e.arg.toString().equals( "Enviar Mensagem" )
                    ||  e.target instanceof TextField ) {
                System.out.println( "MSG " + conta + " "
                    + mensagemUsuario.getText() );
                if ( vrmlSocket != null ) {
                     networkOStream.println( "MSG " + conta
                         + " " + mensagemUsuario.getText() );
                }
                mensagemUsuario.setText("");
```

```
                }
                } } }
            break;
            }
        }
    return false;
    }

    // Tenta a conexao com o servidor
    public boolean conectaServidor ( String nomeServidor ) {
        System.out.println( "Conectando ao servidor..." );
        if ( vrmlSocket == null ) {
            try {
                vrmlSocket = new Socket( nomeServidor, 4900 );
                networkIStream = new DataInputStream(
                    vrmlSocket.getInputStream() );
                networkOStream = new PrintStream(
                    vrmlSocket.getOutputStream() );
                threadRede = new Thread( this, "threadRede" );
                threadRede.start();
            } catch ( IOException e ) {
                System.out.println("Excecao de I/O acusada: "
                    + e.toString() );
                return false;
            }
        }
        return true;
    }
}
```

A.1.2 AVD_GTAconta.java

Este é o programa que cria a janela para perguntar ao usuário seu nome e a máquina aonde deseja conectar-se.

```
// AVD_GTAconta.java
import java.awt.*;
import java.net.*;
import java.io.*;
import java.applet.*;

public class AVD_GTAconta extends Frame {
    // Variaveis
    String conta = "";
    String servidor = "";

    // Define componentes do applet
    Label contaLabel, servidorLabel;
    TextField contaCampo;
    Choice servidorCampo;
    Button ok, cancela;

    // Applet 'pai'
    AVD_GTA GUIprincipal = null;

    // Define Gerenciador de Diagramacao (Layout Manager)
    GridBagLayout gridbag;
    GridBagConstraints constraints;

    public AVD_GTAconta ( AVD_GTA p ) {
        GUIprincipal = p;
```

```
// Cria Gerenciador de Diagramacao (Layout Manager)
gridbag = new GridBagLayout();
constraints = new GridBagConstraints();
setLayout( gridbag );
constraints.weightx = 1.0;
constraints.weighty = 1.0;

// Titulo
setTitle( "Login - AVD_GTA" );

// Conta
constraints.gridwidth = GridBagConstraints.RELATIVE;
contaLabel = new Label( "Conta: " );
gridbag.setConstraints( contaLabel, constraints );
add( contaLabel );

// Campo da conta
constraints.gridwidth = GridBagConstraints.REMAINDER;
contaCampo = new TextField( 15 );
gridbag.setConstraints( contaCampo, constraints );
add( contaCampo );

// Servidor
constraints.gridwidth = GridBagConstraints.RELATIVE;
servidorLabel = new Label( "Servidor: " );
gridbag.setConstraints( servidorLabel, constraints );
add( servidorLabel );

// Campo do servidor
constraints.gridwidth = GridBagConstraints.REMAINDER;
```

```
        servidorCampo = new Choice();
        servidorCampo.addItem( "buzios" );
        servidorCampo.addItem( "lumiar" );
        servidorCampo.addItem( "parati" );
        gridbag.setConstraints( servidorCampo, constraints );
        add( servidorCampo );

        // Botao de Ok
        constraints.gridwidth = GridBagConstraints.RELATIVE;
        constraints.ipadx = 20;
        ok = new Button( "OK" );
        gridbag.setConstraints( ok, constraints );
        add( ok );

        // Botao de cancela
        constraints.gridwidth = GridBagConstraints.REMAINDER;
        constraints.ipadx = 0;
        cancela = new Button( "Cancela" );
        gridbag.setConstraints( cancela, constraints );
        add( cancela );

        // Dimensoes da janela de LOGIN
        resize( new Dimension( 250, 200 ) );
    }

    // Trata toque nos botoes
    public boolean action ( Event e, Object arg ) {
        if ( arg.equals( "Cancela" ) ) {
            dispose();
            return true;
        }
```

```
        if ( arg.equals( "OK" ) ) {
            conta = new String( contaCampo.getText() );
            servidor = new String(
                servidorCampo.getItem(
                    servidorCampo.getSelectedIndex() ) );

            GUIprincipal.mensagemRede.appendText(
                "Conectando " + conta +
                     " ao servidor " + servidor + "...\n" );
            dispose();

            GUIprincipal.conta = new String( conta );
            GUIprincipal.conectaServidor( servidor );

            return true;
        }
        return false;
    }
}
```

A.1.3 painelVRML.java

Este painel (*Panel*) permite que o interpretador do *Liquid Reality* seja acrescido sem ter o tamanho comprometido por seu redimensionamento errado.

```
// painelVRML.java
import java.awt.*;

public class painelVRML extends Panel {
    public synchronized Dimension minimumSize() {
        return new Dimension( 410, 310 );
```

```
    }
    public synchronized Dimension preferredSize() {
        return new Dimension( 410, 310 );
    }
}
```

A.1.4 rastreador.java

Aqui está apresentado o nó *Script* que verifica se o usuário moveu-se e registra sua nova posição e/ou orientação.

```
// rastreador.java
import java.io.*;
import java.lang.*;
import java.net.*;

import vrml.*;
import vrml.node.*;
import vrml.field.*;

public class rastreador extends Script {
    private ConstSFVec3f posicao;
    private ConstSFRotation orientacao;
    private SFVec3f posicao_avatar;
    private SFRotation orientacao_avatar;

    public void initialize() {
        System.err.println( "Inicializando o Script Rastreador." );
        posicao_avatar = (SFVec3f)getField( "posicao_avatar" );
        orientacao_avatar =
            (SFRotation)getField( "orientacao_avatar" );
```

```
    }

    public void processEvent( Event e ) {
        if ( e.getName().equals( "posicao_mudou" ) ) {
            posicao = (ConstSFVec3f)e.getValue();
            posicao_avatar.setValue( (ConstSFVec3f)e.getValue() );
            return;
        }

        if ( e.getName().equals( "orientacao_mudou" ) ) {
            orientacao = (ConstSFRotation)e.getValue();
            orientacao_avatar.setValue(
                (ConstSFRotation)e.getValue() );
            return;
        }
    }
}
```

A.1.5 avd_gta.html

Arquivo HTML necessáio ao *appletviewer* para acionar o *applet* AVD_GTA.class.

```
<!-- avd_gta.html -->
<html>
<applet code="AVD_GTA.class" width=470 height=570>
<param name=vrml value="mundo.wrl">
</applet>
</html>
```

A.1.6 mundo.wrl

Arquivo VRML que descreve o mundo virtual no qual serão inseridos os usuários.

```
#VRML V2.0 utf8
# mundo.wrl
EXTERNPROTO ARVORE [ field SFVec3f localizacao ] "arvore.wrl"

# Ceu e solo
Background {
    skyColor    [ 0.0 1.0 0.8, 0.0 0.5 0.8, 0.0 0.0 0.8 ]
    skyAngle    [ 1.309, 1.571 ]
    groundColor [ 0.5 0.5 0.0, 0.0 0.5 0.0, 0.18 0.54 0.34 ]
    groundAngle [ 1.309, 1.571 ]
}
# Informacoes sobre o avatar
NavigationInfo {
    avatarSize [ 0.5, 1.5, 0.75 ]
    speed 2
}
# Ponto de visao do usuario
DEF AVATARLOC Viewpoint {
    position 0 1.50 0
}
# Sensor que verifica a posicao do usuario
DEF RASTREADOR ProximitySensor {
    center 0 0 0
    size 1000 1000 1000
}
# Ilumina o cenario -> "Sol a pino"
DirectionalLight {
```

```
    direction 0 -1 0
    intensity 0.8
}

# Componentes do mundo agrupados aqui
DEF RAIZ Transform {
   children [
     # Script para rastrear a movimentacao do avatar
     DEF RASTREADOR_SCRIPT Script {
         url "rastreador.class"
            mustEvaluate TRUE
         eventIn SFVec3f posicao_mudou
         eventIn SFRotation orientacao_mudou
         field SFVec3f posicao_avatar 0 0 0
            field SFRotation orientacao_avatar 0 0 1 0
        }

        # Posiciona as arvores
        ARVORE { localizacao -5 0 -5 }
        ARVORE { localizacao 7 0 -2 }
        ARVORE { localizacao -34 0 -10 }
        ARVORE { localizacao 40 0 -10 }
        ARVORE { localizacao 13 0 -20 }
        ARVORE { localizacao -25 0 -50 }
        ARVORE { localizacao 7 0 20 }
        ARVORE { localizacao -4 0 60 }
        ARVORE { localizacao 30 0 -30 }
        ARVORE { localizacao -26 0 -26 }
        ARVORE { localizacao -8 0 -10 }
        ARVORE { localizacao 8 0 -10 }
```

```
# Insere caminho de terra ate' a galeria
Transform {
translation 0 .01 0
    children Shape {
        appearance Appearance {
            material Material { diffuseColor 0.8 0.6 0 }
        }
        geometry Box { size 4 .02 50 }
    }
}

# Insere a Galeria de Arte
DEF GALERIA Transform {
translation -13 0 -25
   children Inline { url "galeria.wrl" }
}

# Insere pedestal e simbolo do GTA girando
DEF GTA Transform {
translation -20 0 0
children [
    Transform {
    translation 0 1 0
        children Shape {
            appearance Appearance {
                material Material { diffuseColor .8 1 1 }
            }
            geometry Cylinder {
                height 2
                radius 2.25
            }
```

```
        }
    }
    Transform {
    translation 0 4.5 0
        children Inline { url "logo_gta.wrl" }
    }
] }

# Define a geometria do Avatar em 3 niveis de detalhe
DEF AVATAR Transform {
children LOD {
    range [ 40, 80 ]
    level [
    # Primeiro nivel de detalhe
        Transform {
        children [
        # CABECA
            DEF CABECA Transform {
            scale 1 1.2 1
            children Shape {
                appearance DEF PELE Appearance {
                    material Material {
                        diffuseColor 0.82 0.71 0.55
                    }
                }
                geometry Sphere {
                    radius 0.2
                }
            } }
        # TRONCO
            Transform {
```

```
        translation 0 -.400 0
        scale 1 1 0.5
        children DEF PEITO Shape {
            appearance Appearance {
                material Material {
                    diffuseColor 1 1 0
                }
            }
            geometry Cylinder {
                height 0.5
                radius 0.25
            }
        } }
    # BRACO DIREITO
        Transform {
        translation -0.3 -.440 0
        children DEF BRACO Shape {
            appearance USE PELE
            geometry Cylinder {
                height .588
                radius 0.08
            }
        } }
    # BRACO ESQUERDO
        Transform {
        translation 0.3 -.440 0
        children USE BRACO
        }
    # QUADRIL
        Transform {
        translation 0 -.770 0
```

```
        scale 1 1 0.5
        children DEF QUADRIL Shape {
            appearance Appearance {
                material Material {
                    diffuseColor 0 0 1
                }
            }
            geometry Cylinder {
                height 0.236
                radius 0.25
            }
        } }
    # PERNA DIREITA
        Transform {
        translation -0.15 -1.18 0
        children DEF PERNA Shape {
            appearance Appearance {
                material Material {
                    diffuseColor 0 0 1
                }
            }
            geometry Cylinder {
                height 0.608
                radius 0.1
            }
        } }
    # PERNA ESQUERDA
        Transform {
        translation 0.15 -1.18 0
        children USE PERNA
        }
```

```
    ] } # Fim do primeiro nivel de detalhe

# Segundo nivel de detalhe
    Transform {
    children [
    # CABECA
        DEF CABECA Transform {
        scale 1 1.2 1
        children Shape {
            appearance DEF PELE Appearance {
                material Material {
                    diffuseColor 0.82 0.71 0.55
                }
            }
            geometry Sphere {
                radius 0.2
            }
        } }
    # TRONCO
        Transform {
        translation 0 -.400 0
        scale 1 1 0.5
        children DEF PEITO Shape {
            appearance Appearance {
                material Material {
                    diffuseColor 1 1 0
                }
            }
            geometry Cylinder {
                height 0.5
                radius 0.25
```

```
                    }
                } }
            # QUADRIL
                Transform {
                translation 0 -.770 0
                scale 1 1 0.5
                children DEF QUADRIL Shape {
                    appearance Appearance {
                        material Material {
                            diffuseColor 0 0 1
                        }
                    }
                    geometry Cylinder {
                        height 0.236
                        radius 0.25
                    }
                } }
            ] } # Fim do segundo nivel
        # Terceiro nivel de detalhe (nenhum)
            Transform { }
    ] # Fim do campo level
} # Fim do no’ LOD
} # Fim da definicao do Avatar
] # Fim do children do no’ raiz

# Roteamento dos eventos
ROUTE RASTREADOR.position_changed
    TO RASTREADOR_SCRIPT.posicao_mudou
ROUTE RASTREADOR.orientation_changed
    TO RASTREADOR_SCRIPT.orientacao_mudou
ROUTE RASTREADOR.position_changed TO AVATAR.set_translation
```

```
    ROUTE RASTREADOR.orientation_changed TO AVATAR.set_rotation
} # Fim da definicao do no' raiz
```

A.1.7 arvore.wrl

Arquivo VRML que define o protótipo da árvore que é utilizada no mundo virtual.

```
#VRML V2.0 utf8
# arvore.wrl
PROTO ARVORE [ field SFVec3f localizacao 0 0 0 ] {
    Transform {
        translation IS localizacao
        children [
    Transform {
    translation 0 1 0
    # Tronco
            children Shape {
                appearance Appearance {
                    material Material {
                        diffuseColor 0.63 0.32 0.18 # Marron (Tronco)
                    }
                }
                geometry Cylinder {
                    height 2
                    radius 0.2
                }
            }       # Fim do Shape
            }       # Fim do 1o. Transform
            Transform {
            translation 0 3 0
            # Copa da arvore
```

```
            children Shape {
                appearance Appearance {
                    material Material {
                        diffuseColor 0.13 0.54 0.13 # Verde (Folhas)
                    }
                }
                geometry Cone {
                    height 2
                    bottomRadius 1.2
                }
            }       # Fim do Shape
            }       # Fim do 2o. Transform
        ]           # Fim do children node da Arvore
    }
}                   # Fim da Definicao do PROTO Arvore
```

A.1.8 logo_gta.wrl

Neste arquivo é definida a geometria da logomarca do GTA que fica rotacionando permanentemente sobre seu eixo.

```
#VRML V2.0 utf8
# logo_gta.wrl
DEF LOGOTIPO Transform {
  children [
    Transform { # Placa de Fundo Verde
      translation 0 0 -.5
      children [
        Shape {
          geometry Box { size 4.5 4.5 .2 }
          appearance Appearance {
```

```
        material Material { diffuseColor .2 .7 .6 }
      }
    }
  ]
}

# Inicio das Esferas
Transform { # Define 1a. Esfera
  translation .6 -1.8 .5
  children [
    DEF Esfera Shape { # Nomeia como Esfera
      geometry Sphere { radius .3 }
      appearance Appearance {
        material Material { diffuseColor 1 0 0 }  # Vermelho
      }
    }
  ]
}
Transform { # 2a. Esfera
  translation -.2 -1.4 .5
    children USE Esfera # Usa Modelo Esfera
}
Transform { # 3a. Esfera
  translation -1 -1 .5
    children USE Esfera # Usa Modelo Esfera
}
Transform { # 4a. Esfera
  translation -1.7 -.5 .5
    children USE Esfera # Usa Modelo Esfera
}
Transform { # 5a. Esfera
```

```
      translation -.8 -.1 .5
        children USE Esfera # Usa Modelo Esfera
    }
    Transform { # 6a. Esfera
      translation .1 .2 .5
        children USE Esfera # Usa Modelo Esfera
    }
    Transform { # 7a. Esfera
      translation 1 .5 .5
        children USE Esfera # Usa Modelo Esfera
    }
    Transform { # 8a. Esfera
      translation 1.8 .8 .5
        children USE Esfera # Usa Modelo Esfera
    }
    Transform { # 9a. Esfera
      translation 2.7 1.2 .5
        children USE Esfera # Usa Modelo Esfera
    }
    Transform { # 10a. Esfera
      translation 3.6 1.5 .5
        children USE Esfera # Usa Modelo Esfera
    }
    Transform { # 11a. Esfera
      translation 4.5 1.8 .5
        children USE Esfera # Usa Modelo Esfera
    }
    # Fim das Esferas

  # Sensores
  DEF Tempo TimeSensor {
```

```
        loop TRUE           # Gera eventos em loop
        cycleInterval 20    # 20 segundos para giro sobre o eixo
  }
  # Posicoes intermediarias no giro
  DEF Animacao OrientationInterpolator {
        key      [ 0,    .2,        .4,
                   .6         .8        1 ]
        keyValue [ 0 1 0 0, 0 1 0 1.2, 0 1 0 2.5,
                   0 1 0 3.8, 0 1 0 5, 0 1 0 0 ]
  }
  ] # children do logotipo
}
#Roteamento dos eventos para provocar o giro permanente
ROUTE Tempo.fraction_changed TO Animacao.set_fraction
ROUTE Animacao.value_changed TO LOGOTIPO.rotation
```

A.1.9 galeria.wrl

Aqui está definida a geometria que compõe a galeria de arte que é inserida no mundo virtual. As chamadas às imagens dos quadros clássicos podem ser substituídas por quaisquer arquivos do padrão GIF ou JPEG.

```
#VRML V2.0 utf8
# galeria.wrl
DEF GALERIA Group {
    children [
        # Paredes externas
        Shape {
            appearance DEF PAREDE Appearance {
                material Material {
                    diffuseColor 1.0 0.9 0.7 # Cor Beje
```

```
            }
        }
        geometry Extrusion {
            convex FALSE
            crossSection [
                # Contorno da galeria
                10 0, 10 -0.5, 5 -0.5, 0.5 -5,
                0.5 -13, 5 -17.5, 21 -17.5, 25.5 -13,
                25.5 -5, 21 -0.5, 16 -0.5, 16 0,
                21 0, 26 -5, 26 -13, 21 -18,
                5 -18, 0 -13, 0 -5, 5 0, 10 0,
            ]
            spine [
                # Linha reta para subida
                0.0 0.0 0.0,
                0.0 3 0.0
            ]
        }
    }
# Parede Interna (Anteparo para porta)
Transform {
translation 13 1.5 -3
    children Shape {
        appearance USE PAREDE
        geometry Box {
            size 12 3 0.4
        }
    }
}
# COLUNAS INTERNAS
Transform {
```

```
translation 8 1.5 -9
    children DEF COLUNA Shape {
        appearance USE PAREDE
        geometry Cylinder {
            height 3
            radius 0.5
        }
    }
}
Transform {
translation 18 1.5 -9
    children USE COLUNA
}

# TETO
Transform {
translation 13 3.25 -9
    children Shape {
       appearance USE PAREDE
       geometry Box { size 26 0.5 18 }
    }
}

# COLUNAS EXTERNAS
Transform {
translation 1.25 1.5 -1.25
    children DEF COLUNA_EXTERNA Shape {
        appearance Appearance {
            material Material { diffuseColor 1 .63 .48 }
        }
        geometry Cylinder {
```

```
            height 3
            radius 1.25
        }
    }
}
Transform {
translation 1.25 1.5 -16.75
    children USE COLUNA_EXTERNA
}
Transform {
translation 24.75 1.5 -16.75
    children USE COLUNA_EXTERNA
}
Transform {
translation 24.75 1.5 -1.25
    children USE COLUNA_EXTERNA
}

# Letreiro
Transform {
translation 10.5 1.5 -2.4
children Shape {
    appearance Appearance {
        material Material { diffuseColor 1 0 0 }
    }
    geometry Text {
        string [ "Galeria de Arte" ]
        length 5
    }
} }
```

```
# QUADROS
Transform {
translation 13 1.5 -17.45
    children Shape {
        appearance Appearance {
            texture ImageTexture { url "quadros/mona.jpg" }
        }
        geometry Box {
            size 1.37 2 0.05
        }
    }
}
Transform {
translation 18 1.5 -17.45
    children Shape {
        appearance Appearance {
            texture ImageTexture { url "quadros/vinci1.jpg" }
        }
        geometry Box {
            size 1.37 2 0.05
        }
    }
}
Transform {
translation 0.55 1.5 -7
rotation 0 1 0 1.57
    children Shape {
        appearance Appearance {
            texture ImageTexture { url "quadros/van_gogh1.jpg" }
        }
        geometry Box {
```

```
                size 1.57 2 0.05
            }
        }
    }
    Transform {
    translation 0.55 1.5 -11
    rotation 0 1 0 1.57
        children Shape {
            appearance Appearance {
                texture ImageTexture { url "quadros/van_gogh2.gif" }
            }
            geometry Box {
                size 1.5 2 0.05
            }
        }
    }
    Transform {
    translation 25.45 1.5 -11
    rotation 0 1 0 -1.57
        children Shape {
            appearance Appearance {
                texture ImageTexture { url "quadros/dali1.jpg" }
            }
            geometry Box {
                size 2 2 0.05
            }
        }
    }
    Transform {
    translation 25.45 1.5 -7
    rotation 0 1 0 -1.57
```

```
        children Shape {
            appearance Appearance {
                texture ImageTexture { url "quadros/dali2.jpg" }
            }
            geometry Box {
                size 2 2 0.05
            }
        }
    }
    Transform {
    translation 13 1.5 -3.25
        children Shape {
            appearance Appearance {
                texture ImageTexture { url "quadros/renoir1.jpg" }
            }
            geometry Box {
                size 2 2 0.05
            }
        }
    }
    Transform {
    translation 9.5 1.5 -3.25
        children Shape {
            appearance Appearance {
                texture ImageTexture { url "quadros/renoir2.jpg" }
            }
            geometry Box {
                size 2.5 2 0.05
            }
        }
    }
```

```
    Transform {
    translation 16.5 1.5 -3.25
        children Shape {
            appearance Appearance {
                texture ImageTexture { url "quadros/renoir3.jpg" }
            }
            geometry Box {
                size 1.5 2 0.05
            }
        }
    }
    # Fim dos quadros
    ]
}
```

A.2 Servidor

A.2.1 Servidor.java

Este programa aguarda a chegada de novos clientes ao mundo virtual. Quando recebe uma solicitação de conexão este cria uma nova *thread* implementada por Intermediario.class para o novo cliente.

```
// Servidor.java
import java.net.*;
import java.io.*;
import java.util.*;
import vrml.field.*;

class Servidor {
    // Vetor onde serao identificados os clientes conectados
```

```
static Vector clientes = new Vector();
static Vector posClientes = new Vector();
// Distancia limite de atualizacao (Filtragem por distancia)
static double limiteDistancia = 100.0;

// Porta do servidor
public final static int PORT = 4900;

// Rotina principal
public static void main(String[] args){
    ServerSocket server_socket = null;
    Socket client_socket = null;
    int id;     // Numero de identificacao do cliente

    System.out.println("Iniciando servidor: " + PORT);

    try {
        server_socket = new ServerSocket(PORT);
    } catch (IOException e) {
        System.out.println( "Nao foi possivel criar
            criar socket em : " + PORT + ", " + e);
        System.exit(1);
    }
    System.out.println("Socket criado: " + PORT);
    System.out.println("Aguardando clientes...");

    while(true) {
        // aceita o pedidos dos clientes
       try {
           client_socket = server_socket.accept();
       } catch (IOException e) {
```

```
                System.out.println("Erro ao aceitar o
                    cliente em: " + PORT + ", " + e);
                System.exit(1);
            }
            System.out.println("Conexao estabelecida: "
                 + client_socket.getInetAddress());
            id = clientes.size();

            // Armazena posicao de cada cliente
            posClientes.insertElementAt(new SFVec3f (0, 0, 0), id);
            // Cria nova thread para o cliente novo
            clientes.insertElementAt( new Intermediario(
                 client_socket, clientes, posClientes, id), id);

            System.out.println(" id = " + id);
        }
    }
}
```

A.2.2 Intermediario.java

Este código implementa a *thread* que faz a comunicação com cada cliente.

```
// Intermediario.java
import java.net.*;
import java.io.*;
import java.util.*;
import java.lang.*;

import vrml.field.*;
```

```
class Intermediario extends Thread {
    DataInputStream in = null;
    PrintStream out = null;

    Vector clientes;
    Vector posClientes;

    Intermediario(Socket s, Vector v, Vector p, int id) {
        super("Intermediario");
        clientes = v;
        posClientes = p;

        try {
            in = new DataInputStream( s.getInputStream() );
            out = new PrintStream( s.getOutputStream() );

            out.println( "100 - Envie sua conta" );
            String resposta = new String ( in.readLine() );

            int index = resposta.indexOf(" ", 0);
            String nomeCliente = new String (
                resposta.substring( index + 1, resposta.length() ) );
            System.out.println( "Usuario : " + nomeCliente );

            out.println("150 - Usuario:" + id + " - Bem vindo!!!" );

            this.start();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

```
public void run() {
    int id = 0;
    String linhaAtual = null;

    while (in != null ) {
    try {
    while ( in.available() > 0 ) {
    System.out.println("Lendo dados do cliente...");
    linhaAtual = in.readLine();
    System.out.println("Linha lida: " + linhaAtual );

    if ( linhaAtual.regionMatches( 0, "SAI", 0, 3 ) ) {
        System.err.println(
            "Recebi pedido de saida. Enviando 900." );
        out.println("900 - Confirmando desconexao.");
        int indexNome = linhaAtual.indexOf( " ", 4 );
        String nomeAvatar = new String (
            linhaAtual.substring( 4, indexNome ) );

        Double idDouble = new Double( linhaAtual.substring(
            indexNome + 1, linhaAtual.length() ) );
        int idNew = idDouble.intValue();

        id = idNew;

        // Envia aos outros usuarios o meu nome
        // para a retirada do meu avatar
        // e o fator de decrementacao do seu id
        int fatorDecrementoId = 0;
```

```
        // Retira regitro de posicao do avatar
        // que esta saindo do sistema
        posClientes.removeElementAt(id);

        for ( int i = 0; i < clientes.size(); i++ ) {
            if ( i > id ) {
                fatorDecrementoId = 1;
            }
            if ( i != id ) {
                ((Intermediario)clientes.elementAt(i)).envie(
                     nomeAvatar, fatorDecrementoId );
            }
        }

        in = null;
        out = null;
        clientes.removeElementAt(id);
        this.stop();
        break;
    } else {

    if ( linhaAtual.regionMatches( 0, "MOVA", 0, 4 ) ) {
        int indexNome = linhaAtual.indexOf( " ", 5 );
        String nomeAvatar = new String (
            linhaAtual.substring( 5, indexNome ) );
        int indexId = linhaAtual.indexOf( " ",  indexNome + 1);
        Integer idInteger = new Integer (
            linhaAtual.substring( indexNome + 1, indexId ) );
        id = idInteger.intValue();

        int indexX = linhaAtual.indexOf( " ", indexId + 1 );
```

```
Double xDouble = new Double(
    linhaAtual.substring( indexId + 1, indexX ) );
float newX = xDouble.floatValue();

int indexY = linhaAtual.indexOf( " ", indexX + 1 );
Double yDouble = new Double(
    linhaAtual.substring( indexX + 1, indexY ) );
float newY = yDouble.floatValue();

Double zDouble = new Double(
    linhaAtual.substring( indexY + 1,
        linhaAtual.length() ) );
float newZ = zDouble.floatValue();

System.err.println( "Movendo " + nomeAvatar
    + " para " + newX + ", " + newY + ", " + newZ );

// Atualiza posicao do avatar no vetor posClientes
((SFVec3f)(posClientes.elementAt(id))).setValue(
    newX, newY, newZ );

for (int i = 0; i < clientes.size(); i++ ) {
  if ( i != id ) {
    if ( estaoProximos( i, id) ) {
      if (estaoMesmaArea( i, id ) ) {
        ((Intermediario)clientes.elementAt(i)).envie(
             nomeAvatar, newX, newY, newZ );
      }
    }
  }
}
```

```
} else {

if ( linhaAtual.regionMatches( 0, "NOVO", 0, 4 ) ) {
    int index = linhaAtual.indexOf( " ", 5 );
    Integer idInteger = new Integer (
        linhaAtual.substring( 5, index ) );
    id = idInteger.intValue();

    int indexNome = linhaAtual.indexOf( " ", index + 1 );
    String nomeCliente = new String (
        linhaAtual.substring( index + 1, indexNome ) );

    int indexX = linhaAtual.indexOf( " ", indexNome + 1 );
    Double xDouble = new Double(
        linhaAtual.substring( indexNome + 1, indexX ) );
    float newX = xDouble.floatValue();

    int indexY = linhaAtual.indexOf( " ", indexX + 1 );
    Double yDouble = new Double(
        linhaAtual.substring( indexX + 1, indexY ) );
    float newY = yDouble.floatValue();

    int indexZ = linhaAtual.indexOf( " ", indexY + 1 );
    Double zDouble = new Double(
        linhaAtual.substring( indexY + 1, indexZ ) );
    float newZ = zDouble.floatValue();

    int indexOX = linhaAtual.indexOf( " ", indexZ + 1 );
    Double xODouble = new Double(
        linhaAtual.substring( indexZ + 1, indexOX ) );
    float newOX = xODouble.floatValue();
```

```
            int indexOY = linhaAtual.indexOf( " ", indexOX + 1 );
            Double yODouble = new Double(
                linhaAtual.substring( indexOX + 1, indexOY ) );
            float newOY = yODouble.floatValue();

            int indexOZ = linhaAtual.indexOf( " ", indexOY + 1 );
            Double zODouble = new Double(
                linhaAtual.substring( indexOY + 1, indexOZ ) );
            float newOZ = zODouble.floatValue();

            Double rODouble = new Double(
                linhaAtual.substring( indexOZ + 1,
                    linhaAtual.length() ) );
            float newOR = rODouble.floatValue();

            // Atualiza posicao do avatar no vetor posClientes
            ((SFVec3f)(posClientes.elementAt(id))).setValue(
                newX, newY, newZ );

            for ( int i = 0; i < clientes.size(); i++ ) {
                if ( i != id ) {
                    ((Intermediario)clientes.elementAt(i)).envie(
                        nomeCliente, id, newX, newY, newZ,
                            newOX, newOY, newOZ, newOR );
                }
            }
        } else {

        if ( linhaAtual.regionMatches( 0, "GIRE", 0, 4 ) ) {
            int indexNome = linhaAtual.indexOf( " ", 5 );
```

```
String nomeAvatar = new String (
    linhaAtual.substring( 5, indexNome ) );

int indexId = linhaAtual.indexOf( " ",  indexNome + 1);
Integer idInteger = new Integer (
    linhaAtual.substring( indexNome + 1, indexId ) );
id = idInteger.intValue();

int indexX = linhaAtual.indexOf( " ", indexId + 1 );
Double xDouble = new Double(
    linhaAtual.substring( indexId + 1, indexX ) );
float newX = xDouble.floatValue();

int indexY = linhaAtual.indexOf( " ", indexX + 1 );
Double yDouble = new Double(
    linhaAtual.substring( indexX + 1, indexY ) );
float newY = yDouble.floatValue();

int indexZ = linhaAtual.indexOf( " ", indexY + 1 );
Double zDouble = new Double(
    linhaAtual.substring( indexY + 1, indexZ ) );
float newZ = zDouble.floatValue();

Double rDouble = new Double(
    linhaAtual.substring( indexZ + 1,
        linhaAtual.length() ) );
float newR = rDouble.floatValue();

System.err.println( "Girando " + nomeAvatar +
    " para " + newX + ", " + newY + ", "
         + newZ + ", r = " + newR );
```

```
        for (int i = 0; i < clientes.size(); i++ ) {
          if ( i != id ) {
            if ( estaoProximos( i, id) ) {
              if (estaoMesmaArea( i, id ) ) {
                ((Intermediario)clientes.elementAt(i)).envie(
                    nomeAvatar, newX, newY, newZ, newR );
              }
            }
          }
        }
    } else {

    if ( linhaAtual.regionMatches( 0, "SOU", 0, 3 ) ) {
        int index = linhaAtual.indexOf( " ", 4 );
        Double iDouble = new Double(
            linhaAtual.substring( 4, index ) );
        int idNovoAvatar = iDouble.intValue();

        int indexNome = linhaAtual.indexOf( " ", index + 1 );
        String antigoAvatar = new String (
            linhaAtual.substring( index + 1, indexNome ));
        System.out.println("Recebendo informacoes sobre " +
            antigoAvatar + " e enviando para usuario: "
                + idNovoAvatar );

        int indexX = linhaAtual.indexOf( " ", indexNome + 1 );
        Double xDouble = new Double(
            linhaAtual.substring( indexNome + 1, indexX ) );
        float newX = xDouble.floatValue();
```

```
int indexY = linhaAtual.indexOf( " ", indexX + 1 );
Double yDouble = new Double(
    linhaAtual.substring( indexX + 1, indexY ) );
float newY = yDouble.floatValue();

int indexZ = linhaAtual.indexOf( " ", indexY + 1 );
Double zDouble = new Double(
    linhaAtual.substring( indexY + 1, indexZ ) );
float newZ = zDouble.floatValue();

int indexOX = linhaAtual.indexOf( " ", indexZ + 1 );
Double xODouble = new Double(
    linhaAtual.substring( indexZ + 1, indexOX ) );
float newOX = xODouble.floatValue();

int indexOY = linhaAtual.indexOf( " ", indexOX + 1 );
Double yODouble = new Double(
    linhaAtual.substring( indexOX + 1, indexOY ) );
float newOY = yODouble.floatValue();

int indexOZ = linhaAtual.indexOf( " ", indexOY + 1 );
Double zODouble = new Double(
    linhaAtual.substring( indexOY + 1, indexOZ ) );
float newOZ = zODouble.floatValue();

Double rODouble = new Double(
    linhaAtual.substring( indexOZ + 1,
        linhaAtual.length() ) );
float newOR = rODouble.floatValue();

((Intermediario)clientes.elementAt(idNovoAvatar)).envie(
```

```
        antigoAvatar, newX, newY, newZ,
            newOX, newOY, newOZ, newOR );
} else {

if ( linhaAtual.regionMatches( 0, "MSG", 0, 3 ) ) {
    int indexNome = linhaAtual.indexOf( " ", 4 );
    String avatarRemetente = new String (
        linhaAtual.substring( 4, indexNome ));
    String msg = new String (
        linhaAtual.substring( indexNome + 1,
            linhaAtual.length() ) );
    System.out.println(
        "MSG " + avatarRemetente + ": " + msg );

    for (int i = 0; i < clientes.size(); i++ ) {
        if ( i != id ) {
            ((Intermediario)clientes.elementAt(i)).envie(
                avatarRemetente, msg );
        }
    }
} } } } } }
} // Fim do while
} catch (IOException e) {
    System.out.println("Nao posso ler dados.");
    System.exit(1);
}
// Retarda um pouco a thread para que as outras threads
// possam executar
try {
    Intermediario.sleep( 100 );
} catch (InterruptedException e) {
```

```
            System.out.println("Fui interrompido.");
        }
        } // Fim do while
    } // Fim do run

    public boolean estaoProximos( int id1, int id2 ) {
        // Pega os dois argumentos
        SFVec3f arg1 = (SFVec3f)posClientes.elementAt(id1);
        SFVec3f arg2 = (SFVec3f)posClientes.elementAt(id2);
        // Calcula a distancia d entre os dois avatares
        // d = sqrt( (x1-x0)^2 + (y1-y0)^2 + (z1-z0)^2 );
        double d = Math.sqrt( Math.pow(
          (double)arg1.getX() - (double)arg2.getX(), 2.0 )
          + Math.pow((double)arg1.getY()-(double)arg2.getY(),2.0)
          + Math.pow((double)arg1.getZ()-(double)arg2.getZ(),2.0));

        System.out.println("Distancia entre "
            + id1 + " e " + id2 + " e’ " + d );

        // Retorna verdadeiro se a distancia
        // entre os avatares for menor que o limite.
        // Caso contrario, retorna falso.
        return ( d < Servidor.limiteDistancia );
    }

    public boolean estaoMesmaArea( int id1, int id2 ) {
        // Pega os dois argumentos
        SFVec3f arg1 = (SFVec3f)posClientes.elementAt(id1);
        SFVec3f arg2 = (SFVec3f)posClientes.elementAt(id2);

        // naGaleria(arg1) XOR naGaleria(arg2)
```

```
        if ( naGaleria(arg1) && naGaleria(arg2) ) {
            return true;
        } else {
            return ( !naGaleria(arg1) && !naGaleria(arg2) );
        }
    }

    // Verifica se arg's indicam presenca na galeria
    public boolean naGaleria(SFVec3f arg) {
        return ( ((((double)arg.getX() > -13.0)
            && ((double)arg.getX() < 13.0))
            && (((double)arg.getZ() > -42.5)
            && ((double)arg.getZ() < -27.8))) );
    }

   // Ha' uma sobrecarga da funcao envie() para cada uma
   // das mensagens possiveis do intermediario ao cliente.
    public synchronized void envie( String nome,
                int fatorDecremento ) {
        out.println("950 " + nome + " " + fatorDecremento );
    }

    public synchronized void envie( String nome,
                float posX, float posY, float posZ,
                float oriX, float oriY, float oriZ, float oriR ) {
        out.println("250 " + nome + " "
            + posX + " " + posY + " " + posZ + " "
            + oriX + " " + oriY + " " + oriZ + " " + oriR );
    }

    public synchronized void envie( String nome,
```

```
            int i, float posX, float posY, float posZ,
            float oriX, float oriY, float oriZ, float oriR ) {
        out.println("200 " + i + " " + nome + " "
            + posX + " " + posY + " " + posZ + " "
            + oriX + " " + oriY + " " + oriZ + " " + oriR );
    }

    public synchronized void envie( String nome,
            float x, float y, float z ) {
        out.println("300 " + nome + " " + x + " " + y + " " + z );
    }

    public synchronized void envie( String nome,
            float x, float y, float z, float r ) {
        out.println("350 " + nome + " " + x + " " + y
            + " " + z + " " + r);
    }

    public synchronized void envie( String nome, String msg ){
        out.println( "400 " + nome + " " + msg );
    }
}
```