RICARDO NAKAMURA

MODELAGEM E DESENVOLVIMENTO DE UMA BIBLIOTECA PARA A CONSTRUÇÃO DE AMBIENTES VIRTUAIS TRIDIMENSIONAIS MULTIUSUÁRIOS

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo, para obtenção do título de Mestre em Engenharia

São Paulo

2002

UNIVERSIDADE DE SÃO PAULO

ESCOLA POLITÉCNICA

Modelagem e Desenvolvimento de uma Biblioteca para a Construção de Ambientes Virtuais Tridimensionais Multiusuários

Ricardo Nakamura

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo, para obtenção do título de Mestre, pelo curso de Pós-Graduação em Engenharia Elétrica – Área de Concentração: Sistemas Digitais.

Orientador: Prof. Dr. Romero Tori

São Paulo

2002

Este trabalho é dedicado aos meus pais, Tutomu e Ana Maria, por todo o apoio que sempre me deram.

Agradecimentos

Ao meu orientador, professor Romero Tori, por toda a ajuda ao longo deste trabalho e pela confiança depositada em mim.

Aos colegas do Interlab, Enio Ribeiro Salles, Andrea Zotovici, Roberto Bianchini e Robson Augusto Siscoutto, pela sua amizade.

À Mariza Ushijima Leone, pela sua amizade, presteza e eficiência ao resolver dificuldades administrativas do Mestrado.

À Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pela bolsa de pesquisa.

À minha família, pela compreensão e suporte durante todos esses anos.

ERRATA

Página 56, parágrafo 4, última frase: substituir por "Um mesmo computador poderá simular um ou mais objetos ativos."

Sumário

Lista de Figuras

Lista de Abreviaturas e Siglas

AOIM – *Area of Interest Management*

API – *Application Programming Interface*

AVM – Ambiente Virtual Multiusuário

CAVE – *CAVE Automatic Virtual Environment*

CORBA – *Common Object Request Broker Architecture*

CRT – *Cathode Ray Tube*

DLL – *Dynamic Link Library*

DCOM – *Distributed Component Object Model*

FTP – *File Transfer Protocol*

IP – *Internet Protocol*

LOD – *Levels of Detail*

RMI – *Remote Method Invocation*

RV – Realidade Virtual

TCP – *Transmission Control Protocol*

UDP – *User Datagram Protocol*

UML – *Unified Modeling Language*

Resumo

O trabalho apresentado neste texto consistiu na modelagem e especificação de uma biblioteca de classes para a construção de ambientes virtuais tridimensionais multiusuários para uso em diferentes aplicações, denominada DAVI3D. Os conceitos relacionados ao desenvolvimento de ambientes virtuais multiusuários foram estudados, assim como os problemas encontrados neste processo e as soluções existentes para os mesmos. Um protótipo desta biblioteca de classes foi produzido. A partir deste protótipo, foi criado um Ambiente Virtual Multiusuário integrado a outro projeto do Laboratório de Tecnologias Interativas (Interlab), o Museu Virtual, para testar a sua aplicação.

Abstract

The research presented in this text consisted in the modeling and specification of a class library named DAVI3D that should allow building three-dimensional, multi-user virtual environments, which could be used in different applications. The concepts related to the development of multi-user virtual environments were studied, along with the problems found in such process and their solutions. A prototype for the class library was developed. A Multi-user Virtual Environment was created with this prototype and integrated with the Virtual Museum, another project from the Interactive Technologies Laboratory (Interlab), to test its applicability.

1. Introdução

O presente trabalho consiste em um projeto na área de Realidade Virtual. Trata-se da modelagem e desenvolvimento de uma biblioteca de classes para auxiliar a criação de Ambientes Virtuais Multiusuários, que foi chamada de DAVI3D (Biblioteca para o Desenvolvimento de Ambientes Virtuais 3D). Na modelagem da biblioteca buscou-se criar uma arquitetura flexível, baseada nos conhecimentos adquiridos durante a pesquisa, sobre o desenvolvimento de Ambientes Virtuais Multiusuários.

De forma simplificada, um Ambiente Virtual Multiusuário (AVM) é um espaço virtual simulado por computadores interligados em rede, no qual duas ou mais pessoas podem interagir. Dependendo da simulação realizada e dos recursos disponíveis, podem ser atingidos vários graus de realismo e o ambiente virtual pode ser aplicado para várias tarefas, entre as quais treinamento, educação e entretenimento.

Um AVM também pode ser chamado de Ambiente Virtual Compartilhado (AVC). Um caso particular desses ambientes são os Ambientes Virtuais

Colaborativos, que apresentam recursos especialmente projetados para facilitar o trabalho colaborativo entre duas ou mais pessoas.

1.1. Objetivos

O principal objetivo deste trabalho é a modelagem de uma biblioteca de classes para a construção de ambientes virtuais multiusuários que possa ser utilizada em diferentes aplicações. Este objetivo inclui também o desenvolvimento de um protótipo desta biblioteca.

O segundo objetivo é a integração do protótipo desenvolvido com outro projeto do Laboratório de Tecnologias Interativas, o Museu Virtual 3D, com a finalidade de demonstrar uma aplicação da biblioteca.

1.2. Motivação

Na década de 90 surgiram diferentes aplicações de Realidade Virtual (RV), principalmente para educação e entretenimento; surgiram também diferentes ambientes virtuais em grupos de pesquisa acadêmicos e de empresas privadas. Nesta mesma década, diferentes autores como Burdea & Coiffet (1994), Pimentel & Teixeira (1995) e Vince (1995) previram o crescimento e popularização das aplicações de RV com o desenvolvimento de computadores de maior capacidade e redução dos custos de dispositivos de interface específicos.

Atualmente, o que se observa é que o crescimento não ocorreu tão rapidamente quanto foi esperado. Em termos de tecnologia e equipamentos, duas mudanças foram especialmente significativas:

- O surgimento de adaptadores gráficos de alto desempenho e baixo custo para a exibição de gráficos tridimensionais em computadores pessoais.

Estes dispositivos permitem que aplicações de RV sejam desenvolvidas em plataformas de custo mais baixo e portanto, acessíveis a um maior número de usuários;

- A perspectiva do aumento da capacidade de transporte de dados através da Internet, o que deverá permitir o desenvolvimento de ambientes virtuais compartilhados mais complexos e com capacidade para mais usuários simultâneos, aumentando a gama de aplicações para os mesmos.

Por outro lado, o custo de outros equipamentos especiais para uso em RV continua elevado, principalmente para usuários domésticos.

Outro fato importante é que ainda persiste o problema observado por Ellis (1994): faltam aplicações que identifiquem a utilidade da RV para a sociedade em geral. O surgimento destas aplicações poderia efetivamente levar a um maior crescimento desta área.

Existem várias aplicações atraentes para os Ambientes Virtuais Compartilhados, tais como: trabalho cooperativo envolvendo equipes situadas em locais diferentes, educação à distância, teleconferência e entretenimento. Uma delas pode se tornar a aplicação que venha a popularizar a área de Realidade Virtual. Os avanços tecnológicos discutidos anteriormente também favorecem o desenvolvimento de aplicações de AVMs.

Atualmente existe um grande interesse em pesquisas e desenvolvimento de ambientes virtuais compartilhados, em parte devido aos motivos apresentados acima. Uma característica importante desta área de pesquisa é a sua dependência de várias áreas distintas de conhecimento. Surgem muitos desafios e problemas na integração entre estas áreas, e não existe uma solução universal.

O projeto descrito nesta Dissertação se insere no contexto exposto acima. A importância do estudo do desenvolvimento de Ambientes Virtuais Compartilhados se encontra no fato de ser um campo novo, em que freqüentemente ocorrem mudanças em função da tecnologia. Portanto, existe o interesse em buscar soluções versáteis e arquiteturas abrangentes, capazes de se adaptar a estas mudanças. A elaboração de tais soluções poderá surgir somente a partir da compreensão dos elementos envolvidos e das dificuldades existentes.

Ao propor a modelagem de uma biblioteca para a criação de ambientes virtuais, este projeto tem como objetivo realizar esses estudos. Espera-se que os resultados deste trabalho possam fornecer uma visão abrangente da área pesquisada, vindo a motivar novas pesquisas e contribuições.

1.3. Organização do Texto

O restante deste texto se encontra dividido em cinco capítulos: o capítulo 2 apresenta conceitos básicos sobre Realidade Virtual e outras áreas de conhecimento relacionadas, que não sejam exclusivamente aplicadas a Ambientes Virtuais, tema este que será tratado no capítulo 3. O capítulo 2 inclui conceitos de Computação Gráfica, Simulação e Comunicação em Redes de Computadores.

O capítulo 3 traz uma introdução aos Ambientes Virtuais, acompanhada das definições relevantes sobre o assunto e específicas desta área, além de uma discussão sobre os problemas encontrados na literatura. São abordados os temas de Arquitetura de Dados, Interface de Usuário e Desenvolvimento de ambientes virtuais. No mesmo capítulo se encontra uma revisão sobre alguns dos diversos ambientes virtuais multiusuários encontrados na literatura e que foram utilizados como referência para o desenvolvimento da biblioteca de classes.

O capítulo 4 trata do desenvolvimento da arquitetura e biblioteca para construção de ambientes virtuais multiusuários propostas neste trabalho. O capítulo inclui uma discussão detalhada da modelagem, especificação e implementação de um protótipo da biblioteca.

O capítulo 5 apresenta o projeto do Museu Virtual desenvolvido no Laboratório de Tecnologias Interativas (Interlab) e o processo de integração entre aquele projeto e este, com a finalidade de disponibilizar os seus produtos em um ambiente virtual multiusuários. Este processo de integração foi realizado com a intenção de demonstrar uma aplicação do protótipo.

Por último, o capítulo 6 traz uma avaliação dos resultados do trabalho e discute futuras pesquisas e aplicações que podem se originar dele.

2. Conceitos Básicos

Este capítulo apresenta alguns conceitos fundamentais sobre Realidade Virtual e áreas relacionadas, como Computação Gráfica, Simulação e Comunicação em Redes de Computadores.

O foco deste capítulo são os conceitos gerais, que se aplicam não somente à ambientes virtuais. O próximo capítulo será dedicado a técnicas mais específicas.

2.1. Realidade Virtual

Realidade Virtual (RV) é nome dado a uma área de pesquisa e aplicação que envolve diferentes ramos da Ciência da Computação, tais como a computação gráfica, sistemas de tempo real, bancos de dados e sistemas de simulação, além de pesquisa de *hardware*, principalmente dispositivos para interação com o usuário.

Segundo Ellis (1994), o campo da Realidade Virtual surgiu das pesquisas sobre simuladores de veículos e teleoperação, na década de 60. Na década de 70, surgiram os primeiros dispositivos especiais de entrada, como luvas sensíveis aos movimentos dos dedos e sistemas de rastreamento magnético de posição. Na década de 80 surgiram aplicações de RV principalmente nas áreas de treinamento e

simulação; entretanto o custo de tais sistemas era extremamente alto e seu uso restrito. A partir da metade daquela década surgiram as versões comerciais de dispositivos especiais para RV (Vince, 1995).

Vince (1995) observa que embora o nome "Realidade Virtual" tenha se tornado popular nos meios de comunicação, alguns autores e pesquisadores o consideram inadequado, por sugerir que os resultados produzidos tenham um grau de realismo ou verossimilhança extremo. Neste caso, preferem o termo Ambiente Virtual, que é mais adequado às limitações tecnológicas e permite a inclusão de sistemas nos quais o realismo não é fundamental.

Burdea & Coiffet (1994) propõem que a Realidade Virtual seja definida por três características: Interatividade, Imersão e Imaginação. A interatividade vem da capacidade do sistema em reconhecer ações do usuário e fornecer respostas em tempo real (ou pelo menos, com um pequeno atraso de tempo). A imersão é a capacidade do sistema em prover estímulos sensoriais ao usuário para que este se sinta envolvido no espaço sintético da simulação. A imaginação é o fator mais abstrato entre os três, e corresponde à resposta do usuário ao sistema, em função das outras duas características; é efetivamente o trabalho que será feito pelo usuário através do sistema. Nesse sentido, pode-se dizer que o objetivo da Realidade Virtual é a geração de espaços sintéticos, em geral tridimensionais, com os quais o usuário possa interagir e nos quais ele possa se sentir como um participante, para a realização ou simulação de tarefas específicas.

De modo geral, um Ambiente Virtual pode ser definido como um sistema computacional no qual se utilizam técnicas e recursos de Realidade Virtual; em outras palavras, é o sistema computacional que implementa o espaço tridimensional

citado anteriormente. Para uma definição mais exata, será utilizada aquela dada por Ellis:

> *"... we can define virtual environments as interactive, virtual image displays enhanced by special processing and by nonvisual display modalities, such as auditory and haptic, to convince users that they are immersed in a synthetic space."* (Ellis, 1994)

Analogamente, um Ambiente Virtual Multiusuário (AVM) ou Ambiente Virtual Compartilhado pode ser definido como um Ambiente Virtual do qual dois ou mais usuários participam simultaneamente. Cada usuário sabe da presença dos demais através de algum indicador do sistema (como por exemplo, uma representação visual de cada usuário) e cada usuário pode interagir com os demais através do sistema – por exemplo, através de gestos, texto ou voz.

Ao longo deste texto, o termo "participante" será usado de maneira equivalente a "usuário". Entretanto, um participante do ambiente virtual pode tanto ser um usuário humano interagindo através de um computador, como um agente autônomo, ou uma simulação por computador.

Freqüentemente, o termo "avatar" é utilizado para designar o indicador ou representação do usuário no ambiente virtual ou, de acordo com Kirner (2001), "um avatar é uma representação, humanóide ou não, de um usuário dentro do ambiente virtual".

O termo "objeto" é usado para indicar um elemento qualquer que compõe o ambiente virtual. Desta forma, em um ambiente virtual reproduzindo um escritório, as paredes, mesas, cadeiras e os avatares dos usuários do ambiente seriam todos objetos. Em alguns contextos, este termo pode ser confundido com o conceito de

objeto das linguagens de programação orientadas a objetos. Ainda assim, o termo será adotado por ser o mais intuitivo para se referir aos elementos presentes no AVM.

O desenvolvimento de um Ambiente Virtual, como sistema de realidade virtual, envolve a aplicação de conhecimentos de diferentes áreas como a computação gráfica, simulação numérica e bancos de dados. No caso de ambientes virtuais compartilhados, conhecimentos sobre comunicação em redes de computadores também se tornam necessários. Estes aspectos de desenvolvimento serão discutidos no restante deste capítulo e também no próximo.

2.2. Computação Gráfica

No desenvolvimento de ambientes virtuais compartilhados, a computação gráfica é utilizada para gerar representações visuais do ambiente para os participantes. Na maioria dos casos, estas representações são construídas utilizando gráficos tridimensionais, embora alguns sistemas suportem também representações bidimensionais do ambiente (Johnson et al., 1998).

O campo da computação gráfica é muito extenso e os seus principais conceitos e algoritmos já foram abordados de forma extensiva por vários autores, como Foley et al. (1996) e Hearn & Baker (1994). Não é um dos objetivos deste trabalho discutir estes algoritmos, que em geral são bem estabelecidos e conhecidos, e implementados em várias bibliotecas de programação. Por outro lado, uma visão geral de alguns tópicos da computação gráfica pode auxiliar na discussão das técnicas específicas para ambientes virtuais que serão apresentadas em seguida. Desta forma, os conceitos fundamentais serão apresentados primeiramente.

2.2.1. Conceitos e Técnicas de Computação Gráfica

Representação de modelos geométricos: Para gerar a imagem de um objeto tridimensional, é preciso ter uma representação deste objeto. Existem diferentes modelos e estruturas de dados que podem ser utilizados para este fim, conforme o tipo de informação que precisa ser armazenada sobre o objeto.

Podem ser identificadas duas grandes categorias de modelos geométricos: representações exatas e aproximadas. A primeira categoria utiliza descrições matemáticas de volumes e superfícies para descrever o objeto em questão. Em geral, estas representações são muito mais precisas, mas também requerem mais capacidade de processamento. A segunda categoria utiliza superfícies mais simples para aproximar a forma do objeto. Num caso extremo e freqüentemente utilizado, a superfície do objeto é aproximada por uma malha de polígonos conectados (Hearn & Baker, 1994; Mäntylä, 1988). Por uma questão de desempenho, em geral as bibliotecas gráficas para geração de imagens em tempo real utilizam este tipo de representação. A figura 2-1 mostra um exemplo deste tipo de representação.

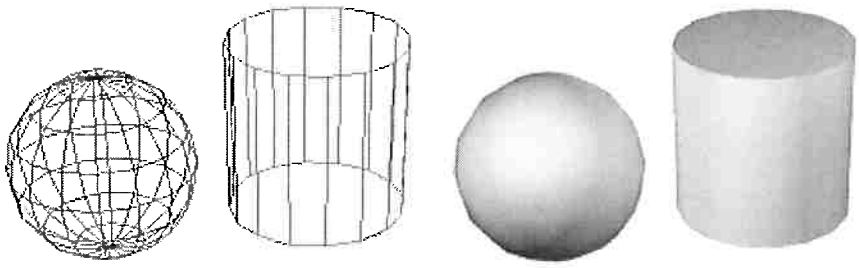


Figura 2-1: Representação geométrica por polígonos

Modelos de iluminação: O olho humano capta a imagem de um objeto a partir da luz que é refletida por ele. As diferentes cores surgem a partir da capacidade da superfície do objeto de absorver determinadas freqüências de luz. Os modelos de iluminação utilizados em computação gráfica procuram reproduzir este

comportamento. Modelos de *ray-tracing*, por exemplo, simulam a trajetória de raios de luz, seguindo um percurso inverso, do olho do observador até as fontes de luz.

Entretanto, modelos de *ray-tracing* exigem muita capacidade computacional e não são adequados para a geração de imagens em tempo real. Neste caso, usam-se modelos aproximados como os modelos de luz ambiente, reflexão de luz difusa, reflexão de Phong entre outros (Hearn & Baker, 1994).

Em geral, os modelos de iluminação procuram calcular a intensidade da luz refletida por um ponto na superfície do objeto, a partir dos raios de luz incidentes. A cor da luz refletida é obtida a partir dos modelos de *shading*.

Modelos de *shading* e texturas: Os modelos de iluminação determinam a intensidade da luz incidente sobre um ponto da superfície de um objeto. A partir desta informação, os modelos de *shading* determinam a luz refletida e portanto a cor daquele ponto. Consequentemente, modelos de *shading* precisam levar em conta as propriedades do material que compõe a superfície do objeto. Diferentes modelos representam de modos distintos estas propriedades.

Uma dificuldade nos modelos de iluminação e *shading* está no fato de que é computacionalmente muito caro realizar os cálculos desses modelos para cada ponto da superfície do objeto. Consequentemente, alguns modelos realizam simplificações para reduzir este custo. Por exemplo, dois algoritmos muito populares, criados por Phong e Gouraud, fazem os cálculos apenas para alguns pontos e usam interpolação para obter as cores dos demais (Hearn & Baker, 1994).

O uso de texturas é outra técnica para reduzir o custo computacional da geração de imagens tridimensionais. Uma textura é uma imagem aplicada sobre a superfície do objeto para determinar as cores de cada ponto daquela superfície. É

necessário definir uma função de mapeamento entre os pontos da textura e os pontos da superfície do objeto, conforme ilustrado na figura 2-2. As coordenadas (x,y) da figura se referem a pontos na superfície da esfera.

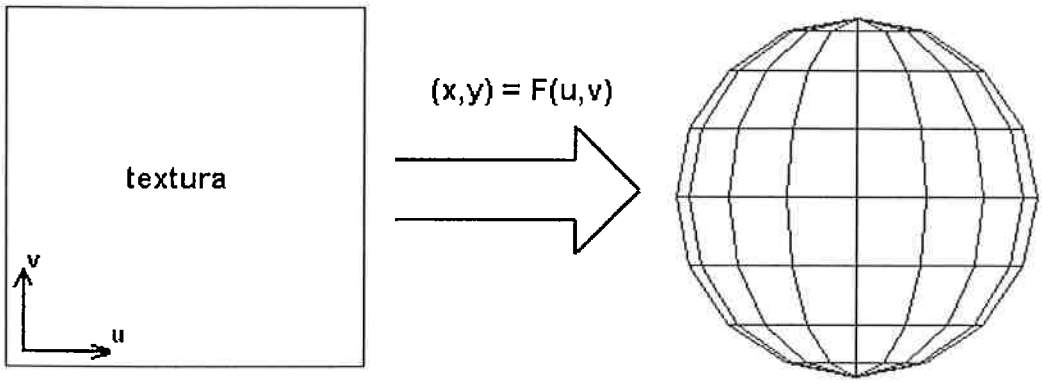


Figura 2-2: Mapeamento de Textura

A vantagem do uso de texturas é que a imagem da textura pode conter detalhes que teriam de ser modelados no objeto. Desta forma, pode-se ter um objeto representado por um modelo geométrico mais simples. Por outro lado, o uso excessivo de texturas pode levar a um consumo de grande quantidade de memória (para armazenar as imagens utilizadas nas texturas).

2.2.2. Técnicas utilizadas em Ambientes Virtuais

Geração de Imagens Estereoscópicas

Um processo de interesse para o uso em ambientes virtuais é a geração de imagens estereoscópicas. A estereoscopia é uma propriedade das imagens observadas pelo olho humano, que apresentam uma disparidade lateral devida à distância entre os olhos. Esta disparidade é interpretada pelo cérebro e utilizada como uma medida para inferir a distância de um objeto.

A geração de imagens estereoscópicas consiste em uma técnica para reproduzir este efeito de disparidade nas imagens visualizadas em um ambiente virtual, adicionando uma sensação de profundidade e imersão do usuário. Esta técnica consiste em gerar duas imagens a partir de dois pontos de vista próximos mas separados horizontalmente de uma distância equivalente à separação entre os olhos do observador. Cada imagem é exibida apenas para o olho correspondente, fazendo com que a imagem final percebida pelo usuário tenha o efeito estereoscópico.

Existem diferentes métodos e equipamentos para realizar a exibição das imagens estereoscópicas. Um deles faz uso de um visor usado como um capacete, no qual são montadas dois dispositivos de exibição na frente dos olhos. Estes dispositivos podem ser telas de cristal líquido ou mesmo tubos de raios catódicos (CRTs). Neste método, as imagens são apresentadas simultaneamente para os dois olhos, mas a construção do capacete faz com que cada olho veja apenas a imagem correspondente (Vince, 1995).

Outro método consiste em usar óculos com obturadores de cristal líquido. Neste caso, um monitor de vídeo exibe alternadamente as imagens para os olhos esquerdo e direito, e envia sinais sincronizados para que os obturadores fiquem abertos ou fechados, permitindo que somente o olho correspondente veja a imagem.

Redução da complexidade na geração das imagens

Um dos maiores desafios da computação gráfica aplicada a AVMs está no fato de que os gráficos devem ser gerados em tempo real e freqüentemente, com um alto grau de detalhamento e "realismo". Isto significa que todos os algoritmos envolvidos nas etapas descritas anteriormente precisam ser executados várias vezes por segundo, para gerar imagens sucessivas do ambiente virtual. Este desafio tem

originado diferentes pesquisas por soluções mais eficientes. Pode-se distinguir dois grandes grupos de soluções para o problema: redução da complexidade da imagem a ser gerada, e redução do esforço computacional para gerar a imagem.

No primeiro caso, procura-se simplificar os modelos geométricos que descrevem os objetos do ambiente virtual. O resultado é que modelos mais simples implicam em menos cálculos nos algoritmos e portanto, menos tempo de processamento. Um exemplo comum deste tipo de solução é uma técnica chamada *levels of detail* (LOD), na qual um objeto possui vários modelos geométricos com diferentes níveis de detalhamento. Quanto mais distante o objeto se encontra do observador, mais simples é o modelo utilizado na geração da imagem, uma vez que os detalhes de um modelo mais complexo não seriam visíveis, devido à distância. (Singhal & Zyda, 1999).

A figura 2-3 mostra um exemplo de objeto com três níveis de detalhe progressivamente menores, da esquerda para a direita. O último nível de detalhe, a ser usado quando o objeto se encontrar a grandes distâncias do observador, é efetivamente apenas um polígono com uma textura.

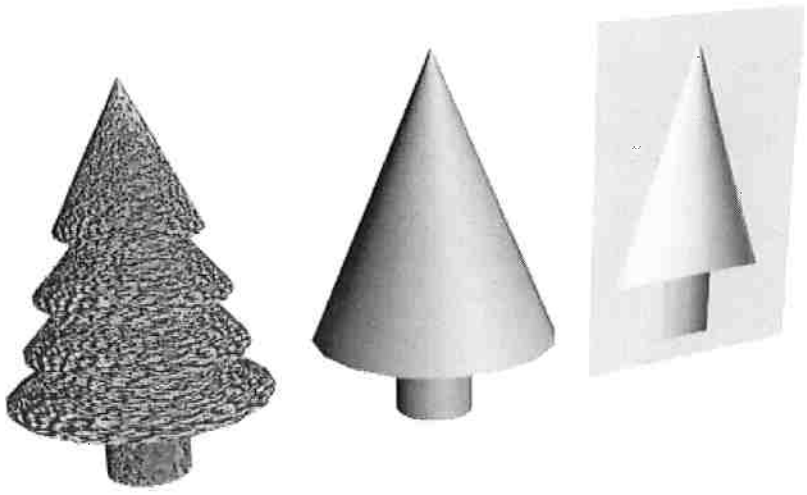


Figura 2-3: Objeto com diferentes níveis de detalhamento

Funkhouser & Sequin (1993) apresentam uma versão adaptativa desta técnica, na qual os níveis de detalhe para cada objeto são escolhidos em tempo de execução com o objetivo de conseguir a imagem mais detalhada possível, em função do número de imagens que devem ser geradas por segundo e da complexidade dos objetos.

No segundo caso, as soluções buscam maneiras de reduzir o número de cálculos necessários na geração de cada imagem. Uma das soluções nesta categoria consiste em utilizar processamento paralelo, dividindo a tarefa de geração das imagens entre vários processadores. Neste caso, o esforço computacional de cada processador é reduzido.

Outras soluções utilizam a propriedade conhecida como *frame-to-frame coherence*, ou coerência entre quadros, que surge em imagens geradas a partir de simulações em tempo real, como no caso dos ambientes virtuais. Essencialmente, duas imagens correspondentes a dois instantes separados por um intervalo de tempo pequeno tendem a ser similares. Uma nova imagem pode, então, ser construída aproveitando elementos da imagem anterior, ou pelo menos informações que foram usadas para gerar aquela imagem. O resultado desta reutilização é que menos processamento precisa ser realizado. Duchaineau et al. (1997) apresentam um algoritmo para visualização de terrenos a partir de mapas de grandes dimensões que utiliza diferentes técnicas, entre as quais a coerência entre quadros para melhorar o desempenho.

2.2.3. Bibliotecas Gráficas

Existe uma série de bibliotecas gráficas disponíveis que implementam os recursos discutidos acima. Nesta seção, serão apresentadas as bibliotecas que foram analisadas como alternativas para uso no desenvolvimento do projeto.

OpenGL: Esta biblioteca consiste em um conjunto extenso de funções para a comunicação com dispositivos de exibição gráfica, que podem ser usadas para a exibição de imagens bidimensionais, mas são especialmente adequadas para imagens 3D. Ela foi criada em 1992 pela empresa *Silicon Graphics*. Atualmente, a funcionalidade desta biblioteca é definida por um padrão especificado por um grupo de fabricantes de *hardware* e empresas de *software* (OpenGL, 2002).

A arquitetura do OpenGL utiliza um modelo de *pipeline* que por sua vez reflete a arquitetura dos dispositivos de exibição. Pode-se dizer que é um modelo com um baixo nível de abstração, sobre o qual pode-se construir camadas de serviços mais complexos. Esta biblioteca está disponível para diferentes plataformas, em geral sob a forma de bibliotecas de sistema como por exemplo, *dynamic link libraries* (DLLs) no Windows.

DirectX: Esta é uma biblioteca para aplicações multimídia, especialmente entretenimento, criada pela empresa Microsoft. A biblioteca usa um modelo orientado a objetos que inclui componentes para a exibição de imagens 2D e 3D. Entretanto, este modelo de objetos não inclui abstrações sobre os dispositivos de *hardware*, sendo comparável ao OpenGL. Um dos problemas desta biblioteca é a sua disponibilidade, que se limita à plataforma Windows (DirectX, 2002).

Open Scene Graph: Esta biblioteca cria uma abstração de alto nível e orientada a objetos, sobre a biblioteca OpenGL, sendo específica para o desenvolvimento de

software na linguagem C++. Durante a elaboração deste texto, esta biblioteca se encontrava disponível para as plataformas Windows, Linux, FreeBSD, IRIX e Mac OSX (Open, 2002).

A abstração utilizada por esta biblioteca é conhecida como "grafo de cena". O grafo de cena é um modelo que descreve uma cena, composta de objetos tridimensionais e outros componentes (tais como fontes de luz) através de um grafo direcionado acíclico. Este modelo permite que o desenvolvimento seja feito em termos dos objetos que compõem a cena, e não dos seus componentes (tais como representação geométrica e matrizes de transformação de coordenadas).

Java 3D: O Java3D é uma API criada pela Sun Microsystems em 1998, com o objetivo de permitir a criação de aplicações Java com gráficos tridimensionais. Assim como o Open Scene Graph, o Java 3D também utiliza o modelo de grafo de cena para atingir um maior nível de abstração. A figura 2-4 mostra um exemplo de grafo de cena, usando a notação sugerida na especificação do Java 3D (Java 3D, 2002). Trata-se de um grafo direcional acíclico, formando portanto uma estrutura em árvore. Na raiz do grafo (representado por um retângulo com bordas arredondadas) encontra-se o "universo", que contém todos os elementos da cena.

Sob o universo podem existir um ou mais Locales (representados por um losango), que representam sistemas de coordenadas dentro do universo. Os grupos (representados por círculos) possuem duas funções principais: organizar a estrutura do grafo de cena e aplicar transformações geométricas sobre os elementos conectados a eles. Por último, as folhas (representadas por triângulos) correspondem aos elementos visuais e de animação da cena.

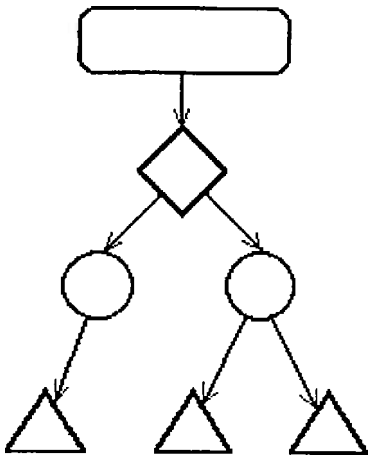


Figura 2-4: Grafo de cena do Java 3D

Esta biblioteca utiliza internamente o OpenGL ou o DirectX para se comunicar com os dispositivos de exibição. No momento da elaboração deste texto, esta biblioteca estava disponível para as plataformas UNIX, Linux, Solaris e Windows (Java 3D, 2002).

Uma característica interessante do Java 3D é que a sua arquitetura prevê a utilização de dispositivos de entrada e saída não convencionais, sendo portanto adequada para o uso em aplicações de RV com estes dispositivos.

2.3. Simulação

Simulação, em um ambiente virtual, se refere aos algoritmos utilizados para definir o comportamento dos objetos que fazem parte do ambiente. Neste caso, são incluídos os métodos de controle de posição e movimentação dos avatares dos usuários.

As técnicas de simulação empregadas no ambiente virtual dependem, evidentemente, dos comportamentos que se deseja simular. Um problema que pode ocorrer na simulação está no fato de que em geral, o comportamento dos objetos é

descrito por uma ou mais equações diferenciais que devem ser integradas com o tempo. Conforme a complexidade destas equações, podem surgir problemas de estabilidade na resolução das mesmas. Vince (1995) e Baraff (1995) discutem técnicas para realizar a simulação do comportamento de objetos e mecanismos segundo as leis da física, em tempo real.

No contexto de ambientes virtuais, a movimentação de um avatar pelo seu usuário também é geralmente considerada um caso de simulação.

Um outro tema que está diretamente ligado com a simulação é a detecção de colisão entre objetos. Conforme Lin & Gottschalk (1998) existem muitas técnicas diferentes, que dependem da representação geométrica adotada para os objetos. Na detecção de colisão existem dois tipos de problemas distintos: verificar a colisão entre dois objetos determinados, e determinar a ocorrência de uma ou mais colisões em um conjunto de objetos. Existem algoritmos apropriados para tratar de cada um destes problemas.

2.4. Comunicação

A comunicação entre computadores é um componente fundamental para os ambientes virtuais compartilhados, afinal é o que permite a utilização do sistema por vários usuários simultâneos situados em locais diferentes.

Além disso, a escalabilidade do sistema, em termos de número de usuários simultâneos, depende diretamente da capacidade do sistema de comunicação em utilizar eficientemente recursos limitados de comunicação, como a largura de banda.

Devido à importância deste tópico, pode-se encontrar na literatura vários estudos sobre a eficiência e viabilidade de diferentes alternativas para a comunicação

em um AVM, bem como formas de classificação dos ambientes de acordo com as suas características (Brutzman et al., 1995; Macedonia & Zyda, 1997; Kubo, 2000).

Neste trabalho, dois aspectos de comunicação entre computadores foram analisados: protocolos de comunicação, abordando as questões relativas a infra-estrutura e transporte de dados e arquiteturas de comunicação, abordando a questão da conectividade e acesso entre os computadores. Estes são os temas das próximas seções. Antes, entretanto, serão apresentados alguns conceitos gerais e dificuldades presentes na área de redes de computadores.

2.4.1. Dificuldades Inerentes ao uso de Redes de Computadores

O uso de redes de computadores na implementação de ambientes virtuais compartilhados traz algumas limitações ou dificuldades que precisam ser levadas em consideração. Estas limitações têm origem nas leis da física e como tal, estarão sempre presentes. As duas principais limitações desta natureza são largura de banda e latência ou atraso.

Largura de banda: Essencialmente, a largura de banda corresponde ao máximo volume de informações que pode ser transmitido através de uma conexão de rede, por unidade de tempo.

Tanenbaum (1996) explica que a largura de banda é função da forma como uma unidade de informação é representada para ser transmitida e da velocidade com que estas representações podem ser transmitidas através de determinado meio físico.

Por exemplo, no caso de uma rede de computadores convencional, utilizando fios de cobre, as informações são codificadas sob a forma de um sinal elétrico. Este sinal, ao ser transmitido através do fio, sofre atenuação devido à resistência elétrica do mesmo. Existe uma freqüência máxima para o sinal transmitido, acima da qual a

atenuação torna o sinal indistinguível de interferências (ruído) externas, e a informação transmitida não pode ser recuperada. Consequentemente, existe um limite para a quantidade de informação que pode ser codificada no sinal elétrico em função daquela freqüência máxima, e isto determina a largura de banda.

No caso de fibras ópticas, atualmente ocorre o contrário. A atenuação do sinal em uma fibra é muito pequena: em teoria, a largura de banda de uma fibra óptica é da ordem de 50Tbps. Na prática, o limite fica em cerca de 1Gbps, porque ainda não existem equipamentos capazes de converter sinais elétricos em sinais luminosos (e vice versa) com maior velocidade (Tanenbaum, 1996).

A partir do que foi exposto, pode-se concluir que a largura de banda pode ser aumentada através de novas tecnologias de transmissão, incluindo novos materiais, mas sempre poderá se tornar um fator limitante.

A largura de banda se torna um problema pelo fato de impor um limite na quantidade de informação que pode ser transmitida. Cada participante de um AVM precisa enviar informações sobre o seu estado atual e receber informações dos outros participantes, resultando em um determinado volume de dados transmitido por participante. Portanto, excluídos outros fatores, a largura de banda determina o número máximo de participantes simultâneos.

Latência ou Atraso: Outro fator fundamental no desenvolvimento de sistemas que utilizam redes de computadores, a latência ou atraso corresponde ao intervalo de tempo entre o envio de um conjunto de informações e a sua recepção e processamento. Em um AVM os atrasos são importantes porque em geral, os dados transmitidos são usados como parâmetros para simular o comportamento dos

participantes. Se o atraso for muito elevado, os participantes do AVM podem observar situações discrepantes.

Existem várias causas para o surgimento de latência em uma rede de computadores. O primeiro fator é físico, proveniente da velocidade finita da luz que introduz um atraso a todo sinal transmitido, em função da distância percorrida. Outra causa de atraso se encontra na transmissão do sinal através da rede, devido ao processamento realizado em equipamentos como roteadores. Existe ainda outra fator, que é o processamento das informações no envio e recepção, incluindo o tempo de conversão do sinal nos equipamentos e no sistema operacional (Singhal & Zyda, 1999; Tanenbaum, 1996).

2.4.2. Protocolos de Comunicação

Existem diferentes protocolos ou mecanismos de comunicação que podem ser empregados para transportar dados entre os computadores envolvidos na simulação do ambiente virtual.

Nesta seção serão descritos quatro mecanismos diferentes de transmissão de dados entre os computadores, representados por diferentes protocolos de comunicação.

Unicast

No modelo de comunicação *unicast*, um computador envia dados a exatamente um destinatário. Se for preciso enviar dados a vários receptores, deverá existir uma conexão para cada um deles, e os dados devem ser enviados repetidamente através de cada conexão. Na Internet, o mecanismo *unicast* pode ser implementado através dos protocolos TCP ou UDP, normalmente adotando-se o primeiro.

O protocolo TCP é baseado no modelo de uma conexão mantida permanentemente entre dois computadores, através da qual um fluxo de dados é transmitido. O protocolo possui mecanismos que garantem a integridade dos dados (Tanenbaum, 1996).

O protocolo UDP se baseia na transmissão de "datagramas", pacotes de dados independentes. Neste caso, não existe uma conexão fixa entre os computadores que se comunicam. O protocolo segue uma estratégia de "fazer o melhor possível" (*best efforts*) para transmitir os datagramas; como resultado, não existem garantias de que todos os datagramas serão recebidos, nem que eles serão recebidos na mesma ordem em que foram enviados. Por outro lado, esta redução na confiabilidade resulta em uma maior velocidade na transmissão de cada datagrama. (Tanenbaum, 1996).

Pelas suas características, o *unicast* através de TCP e UDP têm aplicações diferenciadas. Em geral, dados que precisam ser transmitidos com grande confiabilidade utilizam o TCP. Um exemplo são os protocolos de transferência de arquivos, como FTP. Por outro lado, o UDP pode ser mais vantajoso quando os dados transmitidos podem ser segmentados e a sua recepção contínua e rápida é mais importante do que a confiabilidade. Um exemplo que se enquadra nesta categoria são os protocolos de *streaming* de áudio e vídeo. Neste caso, o conteúdo (som e imagens) é codificado em fragmentos para reprodução em tempo real. A transmissão destes fragmentos através do UDP permite que o requisito de tempo real seja satisfeito, e a perda de alguns fragmentos não causa uma deterioração significativa da informação transmitida.

A principal vantagem do *unicast* está no fato de que protocolos deste tipo estão disponíveis em virtualmente todos os tipos de redes de computadores – ao

contrário do que ocorre, por exemplo, com o *multicast*. Outra característica do *unicast* é que os dados são transferidos apenas para os computadores que devem recebê-los, sem interferir nos demais computadores presentes na mesma rede, como ocorre no *broadcast*.

A principal desvantagem do *unicast* está no volume de dados que precisa ser transmitido através da rede, visto que um conjunto de dados precisa ser enviado repetidamente para cada computador que deve recebê-lo. Este fato limita a escalabilidade de um AVM baseado no *unicast* em termos do número máximo de participantes, em função da largura de banda disponível.

Observa-se na bibliografia pesquisada que o mecanismo *unicast* é adotado na maioria dos protótipos de ambientes virtuais compartilhados juntamente com uma arquitetura do tipo cliente-servidor.

Broadcast

Na comunicação *broadcast*, um computador envia dados para todos os demais computadores presentes na rede. Cada receptor deve julgar se a informação recebida é importante ou não. Este fato traz conseqüências e limitações importantes para o protocolo.

Primeiramente, o envio de grandes quantidades de dados através do mecanismo de *broadcast* interfere nas demais operações da rede pois uma parcela da largura de banda de todos os computadores estará ocupada por estes dados.

Além disso, e como conseqüência do primeiro fato, equipamentos utilizados no transporte de dados em redes de computadores bloqueiam dados enviados através de broadcast para evitar que interfiram em outras redes. Por este motivo, o broadcast só funciona em redes locais.

Por último, Singhal & Zyda. (1999) observam que protocolos *broadcast* são inerentemente ineficientes, pois um computador precisa processar cada mensagem recebida para decidir se a mensagem é relevante, o que leva a um desperdício de tempo de processamento.

Concluindo, o *broadcast* pode ser uma solução simples para ambientes virtuais compartilhados utilizados em redes locais dedicadas, mas não é viável para AVMs que devem operar na Internet.

Multicast

No protocolo *multicast*, um computador envia dados para um canal de distribuição e outros computadores conectados a este mesmo canal recebem aqueles dados. Podem existir vários canais *multicast* independentes, permitindo a separação de diferentes tipos de dados em cada canal. Desta forma, no *multicast* um conjunto de dados é transmitido uma única vez, como no *broadcast*, mas apenas os computadores para os quais estes dados são relevantes os recebem (Singhal & Zyda, 1999).

O protocolo *multicast* está disponível na Internet na forma do protocolo *IP Multicast*; neste protocolo, os canais de distribuição de dados são chamados de **grupos *multicast***. É interessante observar que não é preciso estar conectado a um grupo para enviar dados, mas apenas para recebê-los. Este protocolo utiliza a mesma abordagem de "datagramas" do UDP, discutido anteriormente. Uma conseqüência disto é que o *multicast* não garante a integridade dos dados transmitidos. Koifman & Zabele (1996) discutem alguns mecanismos que podem ser implementados sobre o *multicast* para aumentar a confiabilidade da transmissão.

Diferentes autores, como Funkhouser (1995), Macedonia et al. (1995) e Broll (1998) concordam que o *multicast* é mais eficiente do que o *unicast* e o *broadcast* em termos de volume de dados que devem ser transmitidos e processamento necessário na transmissão e recepção de dados. Consequentemente, é o protocolo que permite maior escalabilidade em termos do número de usuários do AVM.

Por outro lado, o *multicast* também possui desvantagens. A principal delas está no fato deste protocolo não estar disponível amplamente na Internet nem em determinados tipos de rede. Outra limitação similar é que atualmente, equipamentos de rede possuem limitações no número máximo de grupos que podem ser conectados ao mesmo tempo. Entretanto, estas limitações se devem ao fato de que este protocolo é relativamente recente, e pode vir a ser atenuado com o passar do tempo (Funkhouser, 1995; Singhal & Zyda, 1999).

Broll (1997) observa que o *multicast*, por suas características, não é o protocolo mais adequado , para a transmissão confiável de grandes volumes de dados (como por exemplo, modelos geométricos e texturas usados na construção do AVM). Para estes casos, o autor recomenda o uso de um protocolo *unicast*.

Pode-se concluir que o protocolo *multicast* é o mais recomendável para AVMs de grande porte, ou que devem oferecer grande capacidade de escalabilidade em termos de número de usuários. Além disso, o *multicast* pode ser interessante para uso em redes locais, onde não apresenta os problemas de disponibilidade encontrados na Internet e nem os problemas de interferência do *broadcast*.

Modelo de Comunicação Orientado a Objetos

Este modelo difere dos demais protocolos porque constrói uma abstração sobre a camada de comunicação. Para o desenvolvedor, existem apenas objetos

(segundo o conceito de modelagem de software orientada a objetos) acessando uns aos outros através de chamadas de métodos. Não se faz distinção entre os casos em que os objetos se encontram na memória de um só computador, ou em computadores diferentes ligados em rede. Os padrões CORBA e Java RMI podem ser usados para implementar este modelo de comunicação.

Uma das vantagens deste modelo é exatamente o fato dele encapsular todos os detalhes de comunicação, permitindo ao desenvolvedor trabalhar com um modelo de objetos de alto nível.

A desvantagem deste modelo é o seu baixo desempenho comparado com outros protocolos, uma conseqüência direta da camada de abstração adicionada entre os protocolos de comunicação e o restante do sistema. Segundo Kubo et al. (1999), o tempo entre a solicitação e a execução de um método pode levar alguns milissegundos.

Oliveira et al. (1999) propõe o uso de um modelo de comunicação orientado a objetos específico para ambientes virtuais. Segundo os autores, modelos como CORBA e DCOM têm como principal objetivo a reutilização de *software*, mas não satisfazem os requisitos de tempo real dos Ambientes Virtuais Compartilhados.

Por outro lado, Sementille et al. (2000) apresenta um estudo de caso, comparando o uso de CORBA com os serviços de comunicação oferecidos pelo *software* WorldToolKit, destinado ao desenvolvimento de aplicações de Realidade Virtual. O autor observa, após a realização de diferentes testes com configurações diferentes, que "a arquitetura CORBA, em especial com a utilização do seu Serviço de Eventos, pode fornecer um desempenho aceitável para aplicações distribuídas."

Por outro lado, observa que os protótipos desenvolvidos seriam adequados apenas a ambientes virtuais de pequeno ou médio porte.

A conclusão a que se pode chegar é que o modelo de comunicação orientado a objetos pode vir a ser utilizado, no futuro, em AVMs de médio e grande porte, mas seu uso atualmente se restringe a sistemas experimentais ou de pequeno porte.

2.4.3. Arquitetura de Comunicação

As arquiteturas de comunicação se referem à forma em que os computadores envolvidos na simulação estão conectados entre si. Existem três arquiteturas fundamentais: cliente-servidor, ponto-a-ponto e híbrido.

Cliente-Servidor

Na arquitetura cliente-servidor, os computadores não se comunicam diretamente. Cada um dos computadores que participa do AVM (os **clientes**) se comunica com um computador central – o **servidor**.

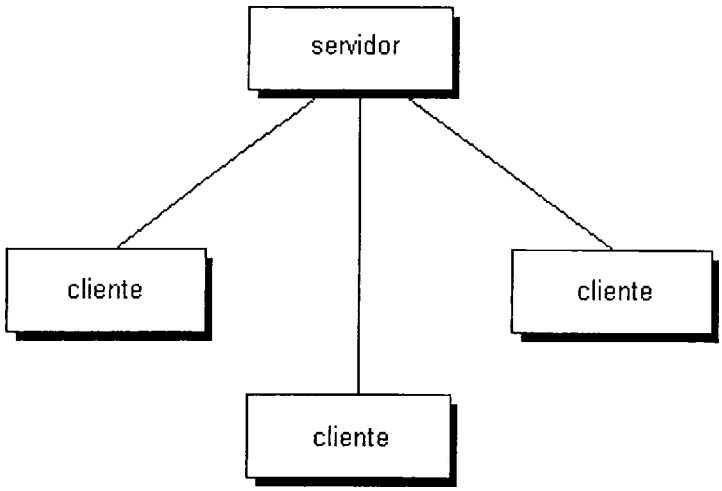


Figura 2-5: Arquitetura Cliente-Servidor

O servidor, por sua vez, é responsável por retransmitir as informações necessárias aos outros clientes. A figura 2-5 apresenta uma representação conceitual desta arquitetura. Implementações utilizando *unicast* seguem exatamente esta estrutura.

No caso de uma implementação usando *multicast*, não há a necessidade de existirem conexões diretas entre os clientes e o servidor. A figura 2-6 apresenta uma possível implementação utilizando dois grupos *multicast*. As linhas contínuas indicam que um participante recebe as mensagens do grupo; linhas tracejadas indicam que o participante somente envia dados para o grupo.

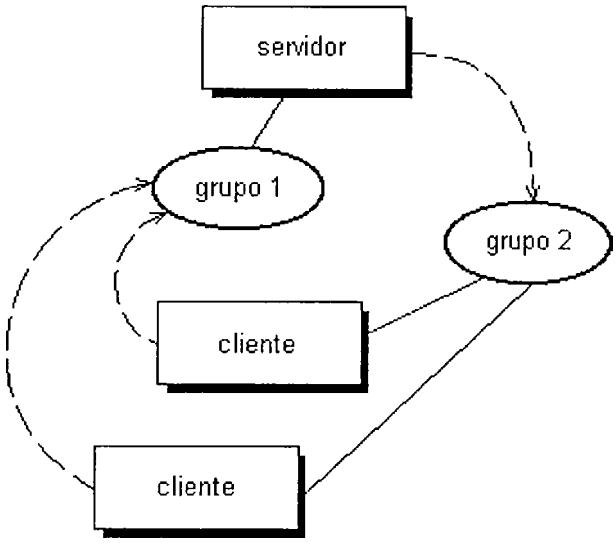


Figura 2-6: Cliente-Servidor com Multicast

Conforme observado por Singhal & Zyda (1999), a arquitetura cliente-servidor apresenta duas desvantagens: a primeira é a introdução de atrasos na transmissão das informações, uma vez que toda informação deve ser enviada primeiro ao servidor e depois seguir para os clientes. A segunda desvantagem é que a

escalabilidade do sistema depende da capacidade do servidor em processar e retransmitir as informações recebidas.

Por outro lado, os autores apontam uma vantagem desta arquitetura: como as informações passam sempre pelo servidor, é possível fazer algum processamento adicional antes que elas sejam retransmitidas. Funkhouser (1995), por exemplo, utiliza algoritmos de oclusão para determinar a visibilidade de cada usuário em relação aos demais. Com isso, o servidor transmite para cada cliente apenas informações sobre os usuários que estão visíveis. Kazman (1993) também propõe uma arquitetura na qual o servidor pode efetuar diferentes operações de "filtragem" de mensagens para distribuição seletiva aos clientes. Em ambos os casos, o resultado é uma economia na quantidade de informações transmitidas através da rede.

Ponto a Ponto

Na arquitetura ponto-a-ponto os computadores estão (pelo menos conceitualmente) todos conectados uns aos outros. Consequentemente, cada computador pode enviar dados diretamente para qualquer outro.

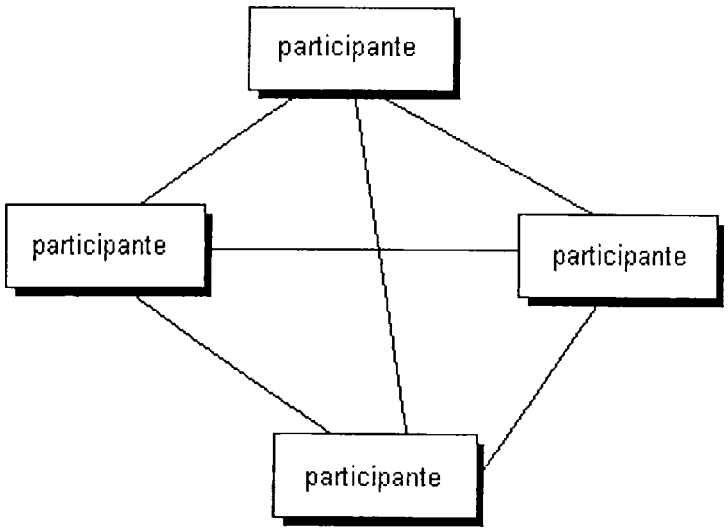


Figura 2-7: Arquitetura ponto-a-ponto

Existem duas implementações básicas desta arquitetura. A primeira utiliza conexões *unicast* entre todos os computadores. Neste caso a implementação coincide com a arquitetura conceitual. Cada computador precisa ter uma conexão com cada um dos demais, conforme ilustrado na figura 2-7.

A outra implementação utiliza os protocolos *broadcast* ou *multicast* para transmissão das informações. Neste caso, cada conjunto de informações é acompanhado de um identificador que determina quais os computadores a que ele se destina. A figura 2-8, a seguir, apresenta uma implementação desta arquitetura através do *multicast*. Neste caso, os participantes estão todos conectados através de um mesmo grupo.

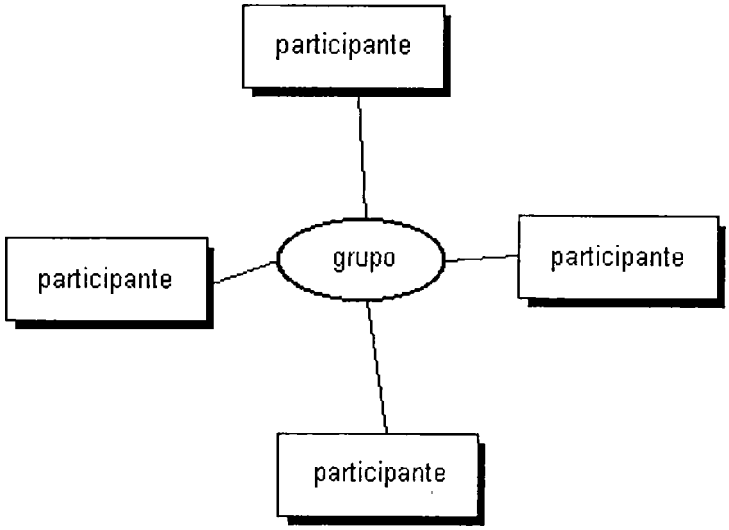


Figura 2-8: Ponto-a-ponto implementado com multicast

Autores como Singhal & Zyda (1999) e Broll (1998) concordam que a arquitetura ponto-a-ponto é mais eficientemente implementada desta maneira.

Arquitetura Híbrida

Arquiteturas híbridas apresentam uma combinação de cliente-servidor e ponto-a-ponto, buscando aproveitar as vantagens de cada uma.

Funkhouser (1995) apresenta uma arquitetura cliente-servidor onde dois ou mais servidores são interligados em uma arquitetura ponto-a-ponto. O objetivo de tal arquitetura é fazer com que cada servidor tenha que tratar de apenas parte do número total de clientes presentes na simulação. A figura 5 ilustra um exemplo desta arquitetura.

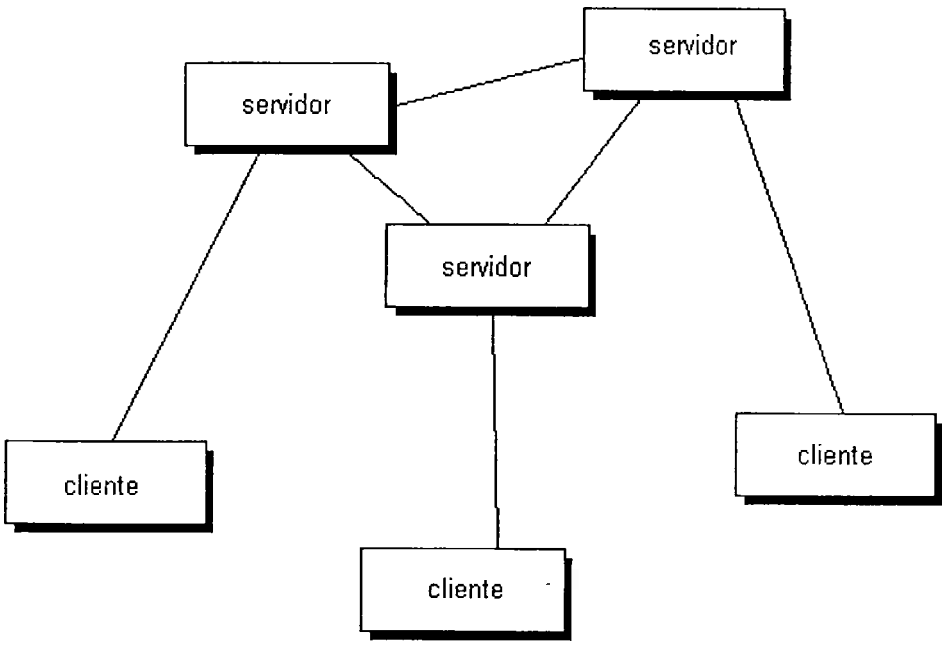


Figura 2-9: Arquitetura Híbrida

2.5. Conclusão

Foram apresentados neste capítulo os conceitos básicos das áreas de conhecimento envolvidas no desenvolvimento de Ambientes Virtuais. Mais informações podem ser obtidas em Hearn & Baker (1994), Tanenbaum (1996) e Singhal & Zyda (1999).

Quanto à arquitetura de comunicação, a biblioteca adota internamente um modelo ponto-a-ponto. Este modelo pode ser adaptado para uma arquitetura cliente-servidor através de uma camada de conversão.

3. Ambientes Virtuais Compartilhados

Este capítulo trata dos conceitos e técnicas envolvidos na construção de um AVM, sob três aspectos: gerenciamento de dados, interface homem-máquina e desenvolvimento ou metodologia de projeto.

O final do capítulo traz uma revisão de alguns dos diferentes ambientes virtuais compartilhados encontrados na literatura que foram utilizados como referência para o desenvolvimento da biblioteca.

3.1. Gerenciamento de Dados

Ambientes virtuais normalmente dependem de uma base de dados para o seu funcionamento. Estes dados incluem modelos geométricos para representação dos objetos no espaço virtual, imagens usadas como texturas e arquivos de som, entre outros. Além destes dados que são predominantemente estáticos, existem conjuntos de dados dinâmicos referentes ao estado das entidades participantes do sistema, incluindo os usuários e possivelmente outros objetos que possuem comportamentos simulados por computador. As informações de estado geralmente incluem a posição de cada entidade, e podem também englobar dados como velocidade, aceleração etc.

A arquitetura de dados, tema desta seção, se refere à localização dessa base de dados, relativa aos computadores participantes do AVM.

Existe um outro aspecto da arquitetura de dados de um ambiente virtual, que é a capacidade de oferecer persistência dos dados. Esta capacidade significa que alterações realizadas no ambiente virtual são armazenadas e podem ser recuperadas posteriormente. Esta capacidade é particularmente importante em ambientes colaborativos como, por exemplo, ambientes para aplicações de engenharia. Aquino & Kirner (2001) discutem uma arquitetura de dados com capacidade de persistência. O que se pode concluir é que, independentemente das arquiteturas que serão expostas a seguir, a capacidade de persistência requer um servidor ou agente centralizador capaz de registrar todas as alterações realizadas no ambiente virtual.

3.1.1. Descrição das Arquiteturas de Dados

Centralizado

No modelo centralizado, a base de dados é conservada em um único servidor. Os dados de natureza estática podem ser requisitados pelo cliente e armazenados localmente para uso posterior. No caso de dados dinâmicos, os clientes devem enviar pedidos de alteração e acesso para o servidor.

A figura 3-1 a seguir ilustra este processo: inicialmente, um dos participantes requisita o acesso a um elemento do banco de dados. O servidor verifica se o dado está sendo alterado por outro participante e, em caso negativo, habilita o participante a fazer alterações. Posteriormente, o participante notifica o servidor, liberando o acesso àquele elemento de dados.

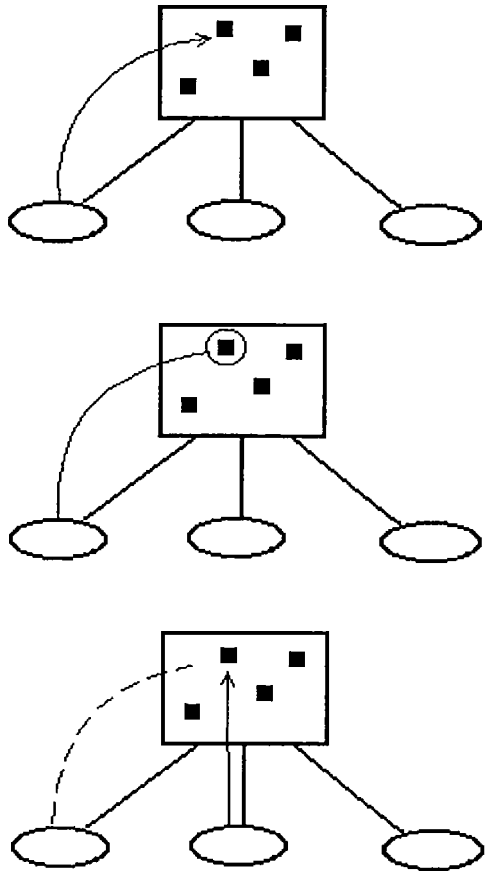


Figura 3-1: Arquitetura de dados centralizada

É importante notar que o desenvolvimento de um servidor de dados para um ambiente virtual é diferente de um servidor tradicional de banco de dados ou um servidor de Web. Pinto (2001) observa que a principal diferença entre um servidor Web e um servidor de ambientes virtuais está no número de transações feitas entre o cliente e o servidor.

Uma característica do modelo centralizado é que não existe a possibilidade de surgirem inconsistências entre os dados observados por diferentes computadores participantes do AVM, uma vez que os acessos aos dados são sincronizados no servidor. Consequentemente, a sua implementação é mais simples, porque não requer mecanismos de sincronização sofisticados.

Uma desvantagem deste modelo está exatamente no fato de que qualquer alteração deve ser autorizada pelo servidor, introduzindo um atraso no acesso aos dados. Este atraso se reflete numa redução do número máximo possível de alterações que podem ser realizadas em um elemento de dado, e consequentemente, em uma diminuição na interatividade do ambiente.

Distribuído Replicado

No modelo distribuído replicado existem cópias idênticas de todo o banco de dados em todos os clientes, incluindo dados estáticos e dinâmicos. Quando um cliente faz uma modificação em um dado, ele deve notificar os outros clientes desta modificação. É mais difícil garantir consistência entre os clientes nesta arquitetura, pois a princípio, dois clientes podem alterar o mesmo dado simultaneamente. Além disso, dois clientes podem armazenar dados incompatíveis, devido a atrasos na recepção das mensagens de atualização. A figura 3-2 apresenta o funcionamento deste modelo.

A maior vantagem do modelo distribuído é que ele permite uma maior interatividade em troca de uma menor consistência. Como os clientes podem alterar livremente os dados, não ocorrem os atrasos presentes em uma arquitetura de dados centralizada.

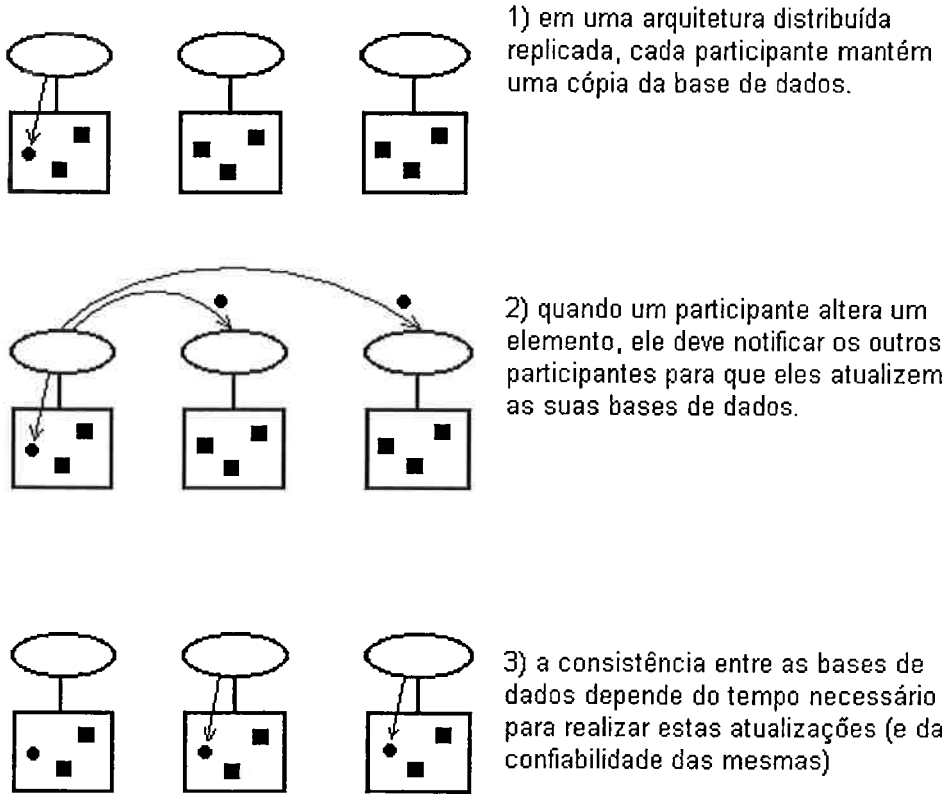


Figura 3-2: Arquitetura de dados distribuída replicada

A biblioteca de classes implementada adota este modelo de arquitetura de dados, devido às características expostas anteriormente.

Distribuído Particionado

O modelo distribuído particionado tem as mesmas propriedades do modelo distribuído replicado, com uma diferença: cada cliente armazena apenas informações relativas à uma parte do AVM que está atualmente acessível a ele.

Primeiramente, este modelo traz o problema de determinar o que significa "acessível". Em alguns casos, o espaço virtual do AVM é subdividido em áreas, e o cliente mantém uma base de dados correspondente apenas à área em que ele se

encontra no momento. Outros métodos incluem a determinação de que objetos estão visíveis para o cliente, ou uma combinação das duas técnicas.

Outro problema do modelo distribuído particionado é o gerenciamento da base de dados conforme o cliente se move dentro do AVM. É preciso determinar que informações podem ser descartadas, e que novas informações precisam ser armazenadas.

A vantagem que o modelo distribuído particionado traz em relação ao modelo distribuído replicado é que este modelo requer menos memória, visto que apenas parte da base de dados do ambiente virtual é armazenada. Esta economia de memória pode não ser significativa para ambientes de pequeno e médio porte, tendo em vista as dificuldades adicionais apresentadas. No caso de ambientes de grande porte, entretanto, esta arquitetura de dados pode ser necessária para viabilizar o AVM.

3.1.2. Gerenciamento de Áreas de Interesse

Um outro aspecto da arquitetura de dados, que tem sido pesquisada e aplicada, principalmente em ambientes virtuais compartilhados de médio e grande porte, são as técnicas de Gerenciamento de Áreas de Interesse ou *Area of Interest Management* (AOIM).

A motivação principal destas técnicas é a redução do consumo de largura de banda e capacidade de processamento dos participantes de um ambiente virtual. Isso é feito através da filtragem ou seleção das informações que cada participante precisa receber.

Em geral, técnicas de AOIM são aplicadas a ambientes virtuais com arquiteturas de dados distribuídos particionados, unindo as suas próprias vantagens a um menor consumo de memória causado pelo particionamento da base de dados.

Morse (1996) traz uma análise de sete implementações de técnicas de AOIM. Em geral, o critério usado para fazer a filtragem ou divisão em áreas de interesse é espacial. Por exemplo, o ambiente virtual pode ser dividido em "células". Cada participante recebe somente informações dos objetos que se encontram na mesma célula ou em células vizinhas. Diferentes técnicas de AOIM propõem células de tamanho e forma constante ou variável. Na figura 3-3, por exemplo, o participante indicado por um quadrado recebe apenas informações dos outros dois participantes situados na mesma célula.

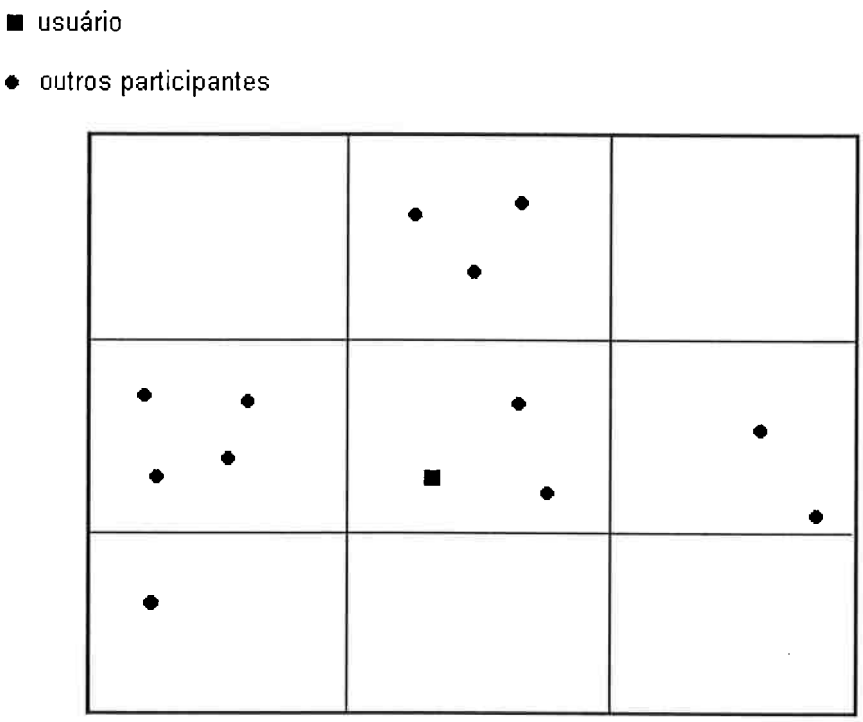


Figura 3-3: Exemplo de AOIM

Em geral, as técnicas de AOIM se utilizam de grupos de *multicast* para implementar a filtragem de mensagens. No exemplo anterior, cada célula poderia estar associada a um grupo distinto. Entretanto, Morse (1996) observa que esta

filtragem não permite controle muito preciso das informações que serão recebidas, nem garante uma distribuição homogênea dos objetos pelos grupos *multicast.*

Adicionalmente, Funkhouser (1995) observa que o processo de conexão ou desconexão de um grupo *multicast* é um processo relativamente lento, e portanto pode causar atrasos na recepção de dados, se o participante precisar trocar com freqüência de grupo (como no caso de um participante se movendo entre células diferentes).

No caso de arquiteturas cliente-servidor, técnicas semelhantes a AOIM podem ser implementadas através de algoritmos no próprio servidor. Os sistemas descritos por Kazman (1993) e Funkhouser (1995) são exemplos de tais técnicas. Obviamente, isso acarreta um maior uso da capacidade de processamento do servidor, o que deve ser levado em consideração.

A conclusão a que se pode chegar é que as técnicas de AOIM trazem vantagens para ambientes virtuais de grande porte, nos quais o número de participantes pode exceder a capacidade de processamento ou largura de banda disponíveis. Em ambientes menores, a sua utilidade precisa ser analisada com mais cautela. Não há previsão de incluir este tipo de técnica na biblioteca de classes.

3.1.3. Consistência

Em um AVM, a manutenção da consistência entre as bases de dados dos participantes é muito importante. Caso isso não ocorra, cada participante pode ter uma impressão diferente do ambiente virtual, o que reduz a sensação de imersão e interatividade e dificulta a realização de tarefas cooperativas.

Como foi observado na seção anterior, no caso de uma base de dados centralizada a manutenção da consistência não é um problema, mas no caso de bases

distribuídas é preciso definir mecanismos para garantir um certo nível de consistência entre os participantes, ou limitar a quantidade de inconsistência possível.

Funkhouser (1995) observa que o problema de manter a consistência em um AVM apresenta a dificuldade adicional de se ter um grande volume de atualizações que precisam ser propagadas rapidamente. O autor conclui que sistemas gerenciadores de bancos de dados distribuídos de uso geral não são adequados. Seguindo esta conclusão, pode-se encontrar na literatura diferentes soluções especializadas para o problema.

Singhal & Zyda (1999) apresentam três técnicas que podem ser utilizadas na manutenção de consistência: *shared repository*, *blind broadcasting* e *dead reckoning*.

No caso do *shared repository*, a idéia é simular uma base de dados centralizada. Quando um cliente deseja fazer uma alteração em um elemento do AVM, ele notifica os demais e a partir daí pode fazer as alterações. Depois ele envia uma outra notificação de que não pretende fazer mais alterações.

Galli & Luo (2000) apresentam um protocolo de consistência desenvolvido para uma aplicação de CAD colaborativo implementado usando VRML. O sistema abordado no artigo utiliza uma arquitetura de dados distribuídos replicados. No protocolo proposto pelos autores, um participante deve informar a todos os outros que pretende fazer alterações em um determinado objeto. Se todos os outros participantes responderem permitindo o acesso àquele objeto, aquele participante pode fazer as alterações que quiser naquele objeto enquanto for necessário.

No *blind broadcasting*, cada cliente envia com freqüência informações sobre o estado dos objetos que estão sendo simulados naquele computador. Não existe

garantia de que toda a informação será recebida por todos os participantes. O protocolo confia no fato de que a informação é retransmitida freqüentemente para fazer com que nenhum cliente fique com uma visão muito defasada do estado atual do AVM. Entretanto, já não existe a garantia de consistência. Uma vantagem desta técnica está na maior interatividade, em relação ao *shared repository*. Uma desvantagem está no uso de largura de banda.

Finalmente, o *dead reckoning* leva o *blind broadcasting* um passo adiante. Em vez de transmitir freqüentemente todas as suas informações, cada participante mantém um modelo de simulação simplificado (por exemplo, interpolação) e quando este modelo atinge um certo nível de discrepância em relação à simulação verdadeira, ele envia um novo conjunto de dados. Os outros participantes utilizam apenas o modelo simplificado para definir o comportamento das demais entidades, e atualizam este modelo quando recebem um novo conjunto de dados.

Lu et al. (1999) apresenta uma técnica para manter a consistência em AVMs grandes, no qual os três mecanismos discutidos anteriormente podem ser usados, dependendo do número de usuários presentes no sistema, com o objetivo de manter o desempenho do sistema dentro de um certo limite. Essencialmente, cada uma delas é utilizada conforme a largura de banda disponível e o número de usuários do sistema. Outro conceito interessante apresentado pelos autores é o de objetos ativos e passivos, com diferentes graus de importância e utilizando diferentes mecanismos de atualização.

A biblioteca de classes implementada utiliza a distinção entre objetos ativos e passivos para determinar os objetos que necessitam de controle de atualização e

consistência. A princípio, qualquer um dos três modelos de consistência discutidos acima poderia ser implementado através da biblioteca.

3.2. Interface de Usuário

O aspecto das interfaces de usuário nos ambientes virtuais tem sido pouco estudado, em particular sob o ponto de vista de usabilidade. Bowman & Hodges (1995) observam que as interfaces de usuário em AVs não foram alvo de muitas análises quantitativas ou testes com usuários. O resultado é que, em muitos casos, o sistema se torna difícil de ser utilizado, devido a uma interface inadequada.

Uma dificuldade adicional na especificação de interfaces para ambientes virtuais é que elas podem diferir completamente das interfaces tradicionais, em termos de operação e metáforas. Um exemplo de interface de usuário não convencional é o sistema de anotações apresentado por Harmon et al. (1996). Através deste sistema, um participante pode associar textos ou gravações de voz a objetos em um ambiente virtual, que podem ser acessados posteriormente. Neste caso, os autores indicam que realizaram alterações no formato da interface, depois de estudos iniciais de usabilidade.

As pesquisas ligadas a interfaces de usuário são mais intensas na área de ambientes virtuais colaborativos – aqueles em que os participantes devem participar conjuntamente da realização de uma tarefa, como por exemplo, ambientes para projetos de engenharia. Diversos autores relatam experiências com interfaces para este tipo de ambiente virtual (Fraser et al., 1999; Hindmarsh et al., 1998; Frécon & Nöu, 1998; Robertson et al., 1997;) e de modo geral, suas conclusões podem ser utilizadas em ambientes virtuais compartilhados colaborativos ou não.

Em Bowman & Hodges (1997) e Bowman et al. (1997), podem ser encontradas avaliações e recomendações sobre diferentes técnicas de interface de usuário em ambientes virtuais, incluindo movimentação do usuário e manipulação de objetos. Os autores apresentam algumas conclusões importantes, que podem ser aplicadas no desenvolvimento de novas interfaces de usuário:

- os processos de seleção de objetos e manipulação de um objeto devem ser considerados separadamente. Existem técnicas mais adequadas para cada um deles (Bowman & Hodges, 1997);
- a velocidade com que um usuário se movimenta no ambiente virtual não afeta muito o seu senso de localização, desde que haja continuidade na trajetória. Técnicas que envolvem o transporte instantâneo do usuário entre locais diferentes causam uma maior desorientação (Bowman et al., 1997);

Os resultados apresentados por Jacob et al. (2001) também concordam com as conclusões acima. Considerando todos os trabalhos envolvidos e os testes de usabilidade realizados, conclui-se que aquelas recomendações podem ser adotadas como base para a elaboração de uma interface de usuário para um AVM.

Como será discutido no capítulo 4, a modelagem da biblioteca de classes não levou em conta os problemas específicos de interface de usuário, que dependem das necessidades de cada aplicação em particular.

3.3. Desenvolvimento

Um dos problemas que surgem no desenvolvimento de ambientes virtuais é a pouca utilização de metodologias de projeto de *software*. Oliveira et al. (1999) observam que existem poucos esforços em se aplicar metodologias de

desenvolvimento a AVs, e o resultado são sistemas inflexíveis e de pouca qualidade. Os autores indicam que isto resulta de uma mentalidade de criar protótipos ou testes conceituais, e não produtos.

Kim et al. (1998) também observam a pouca estruturação e método no desenvolvimento de ambientes virtuais. Conforme apontado pelos autores, em geral a especificação de um ambiente virtual se limita à definição do seu conteúdo e comportamento dos objetos; pouca importância é dada à arquitetura do sistema como um todo. Neste mesmo artigo, os autores apresentam uma metodologia derivada das metodologias de programação orientadas a objetos, utilizando *statecharts*, para auxiliar no levantamento de requisitos para um ambiente virtual.

Alternativamente, Martins et al. (1999) relatam o uso de uma metodologia de projeto orientada a objetos utilizando a notação UML (*Unified Modeling Language*), tradicionalmente empregada no projeto de sistemas de software orientado a objetos, para a especificação e projeto de um ambiente virtual. Os autores concluem que tal metodologia é adequada para o projeto de Ambientes Virtuais Compartilhados.

Um dos objetivos principais deste trabalho é realizar a modelagem da biblioteca de classes quanto à sua arquitetura. Utilizou-se para isso uma metodologia de projeto de software, verificando assim a sua adequação e buscando-se evitar os problemas de desenvolvimento citados acima.

3.4. Revisão de Ambientes Virtuais

Nesta seção serão apresentados alguns ambientes virtuais e *toolkits* encontrados na literatura, que foram usados como referência para o desenvolvimento da biblioteca de classes.

3.4.1. SPLINE/Diamond Park

O *Diamond Park* foi uma demonstração do sistema SPLINE desenvolvido entre 1995 e 1996 pelos laboratórios de pesquisa da Mitsubishi (MERL). Trata-se de um ambiente virtual multiusuário dividido em regiões interligadas incluindo locais fechados (como prédios) e abertos (como parques). O sistema SPLINE foi especificado para o desenvolvimento de ambientes virtuais de grande escala, capazes de sofrerem alterações sem a necessidade de um reinício do sistema. A arquitetura adotada é ponto-a-ponto, com uma base de dados distribuída particionada (Anderson et al., 1995).

O sistema SPLINE introduziu algumas técnicas de particionamento do ambiente virtual e conexão de novos usuários em uma arquitetura ponto-a-ponto, denominadas *Locales* e *Beacons*. (Barrus et al., 1996).

Essencialmente, um *Locale* corresponde a um volume do espaço com forma arbitrária e um sistema de coordenadas local usado para definir a posição dos objetos dentro deste volume. Um ambiente virtual pode ser composto de um ou mais *Locales*, e a relação entre eles é dada por uma matriz homogênea de transformação de coordenadas (que no caso mais simples, corresponderá a apenas uma translação). O uso de *Locales* corresponde à aplicação de técnicas de AOIM para reduzir o número de mensagens que precisam ser enviadas através da rede de computadores. Entretanto, o critério utilizado para selecionar os *Locales* dos quais um participante deverá receber informações é baseado na visibilidade. Portanto, uma definição inadequada dos *Locales* pode resultar em uma aplicação ineficiente onde todos os participantes recebem dados de todos os *Locales*.

Um *Beacon* é uma entidade associada a um grupo de *multicast* dentro do ambiente virtual do SPLINE. Normalmente, existe pelo menos um *Beacon* por *Locale*. Como não existem servidores, os *Beacons* atuam como agentes centralizadores, registrando os objetos que entram e saem de um *Locale* e permitindo a sua localização. Este serviço é importante porque o SPLINE foi especificado para ambientes virtuais de grandes dimensões com muitos objetos e esta é a única maneira de localizar um determinado objeto rapidamente.

De um modo geral, pode-se concluir que o SPLINE foi, na época do seu desenvolvimento, um dos ambientes virtuais compartilhados mais avançados, com uma arquitetura bem definida.

A influência do SPLINE neste trabalho ocorre de maneira indireta, visto que alguns conceitos introduzidos naquele ambiente foram empregados no Java 3D, como por exemplo o conceito do *Locale* como referência local do sistema de coordenadas.

3.4.2. DWTP

O DWTP é um sistema desenvolvido para a criação de ambientes virtuais compartilhados pela Internet, utilizando protocolos como FTP, outros protocolos dedicados, e a linguagem VRML (Broll, 1998).

Uma das características interessantes deste protocolo é a utilização de diferentes protocolos para a transferência de tipos diferentes de dados. Desta forma, busca-se aproveitar as vantagens de cada protocolo. Por exemplo, grandes volumes de dados correspondentes aos modelos geométricos do ambiente virtual são transferidos por conexões do tipo FTP, usando *unicast* confiável. Por outro lado,

informações sobre as alterações dos objetos e participantes do ambiente virtual são transferidas através de *multicast*.

A arquitetura do DWTP é composta por uma série de agentes responsáveis por garantir a confiabilidade das transmissões de dados e permitir que participantes incapazes de realizar conexões *multicast* possam utilizar o ambiente virtual. Desta forma, a arquitetura suporta primariamente o modelo ponto-a-ponto, com conexões *multicast* e base de dados distribuída (replicada ou particionada).

A partir do DWTP foi retirada a idéia de abstrair a camada de rede, assumindo um modelo ponto-a-ponto que é compatibilizado com a arquitetura de redes real através de uma camada mais baixa.

3.4.3. jVE

O jVE é um sistema baseado em uma arquitetura para suporte a aplicações colaborativas (Harada, 2002). O objetivo do sistema é permitir a construção de ambientes colaborativos que podem ser acessados através de diferentes representações como, por exemplo, textual (HTML) e visual (objetos tridimensionais). A arquitetura do jVE é baseada em um modelo de dados centralizado, no qual um servidor faz o controle de alterações aos objetos do ambiente colaborativo.

Existem duas diferenças fundamentais entre o jVE e o projeto apresentado neste texto. Em primeiro lugar, a arquitetura de dados usada no presente projeto segue o modelo distribuído replicado. Adicionalmente, a pesquisa do jVE enfatizou a definição de uma arquitetura e protocolos para ambientes colaborativos, independente da representação visual. Por outro lado, este trabalho se destina

especificamente a ambientes multiusuários baseados em imagens 3D, o que leva a um maior enfoque nas técnicas de computação gráfica e interação.

3.4.4. MuseuVirtual

O MuseuVirtual é um Ambiente Virtual Colaborativo para uso através da Internet. O objetivo do projeto é servir como plataforma para o desenvolvimento de aplicações de ensino. Este sistema usa uma arquitetura cliente-servidor com conexões *unicast* com servidor centralizado de dados (Wazlawick et al., 2000). Conforme os diagramas apresentados em Kirner et al. (2001), o servidor atua como centralizador das mensagens dos clientes, servidor de dados e também armazena o estado do sistema para fins de registrar a sua utilização e permitir a persistência de dados.

Este sistema foi estudado para a compreensão dos mecanismos de persistência do ambiente virtual, embora estes mecanismos não tenham sido incluídos na biblioteca de classes.

3.4.5. NICE

O sistema NICE foi desenvolvido para ser utilizado em aplicações de ensino para crianças, com equipamentos especiais para realidade virtual imersiva, como o CAVE. Em termos de arquitetura, o NICE usa o modelo cliente-servidor, com conexões unicast e uma base de dados centralizada para garantir consistência total entre os participantes. O servidor do NICE também é responsável por armazenar dados alterados pelos usuários, permitindo a sua persistência (Johnson et al., 1998).

O sistema NICE serviu de inspiração para especificar uma interface de saída flexível permitindo o uso da biblioteca de classes tanto em dispositivos convencionais (monitor de vídeo) como em dispositivos especiais.

3.4.6. NPSNET

O NPSNET é um ambiente virtual de grande escala desenvolvido pela Naval Postgraduate School, para ser usado em simulações militares. Grande atenção é dada para tecnologias que garantam a escalabilidade do sistema, como AOIM e uso de Multicast (Brutzman et al., 1995; Macedonia et al., 1995).

A arquitetura do NPSNET segue o modelo ponto-a-ponto, com conexões *multicast* e base de dados distribuída replicada. Os protocolos são baseados no padrão DIS (vide SIMNET).

A biblioteca de classes utiliza um modelo de base distribuída replicada, tal como o NPSNET, ainda que sem a sofisticação encontrada nele (tendo em vista que o NPSNET se destina a ambientes de grandes dimensões).

3.4.7. RING

O RING é um AVM com arquitetura híbrida onde vários servidores estão ligados por conexões ponto-a-ponto (Funkhouser, 1995).

Esta arquitetura híbrida tem como objetivo dividir os clientes participantes do ambiente virtual entre os servidores. Com isso, a capacidade de processamento necessária em cada servidor é reduzida, assim como a utilização de largura de banda para cada servidor.

Adicionalmente, os servidores do RING utilizam algoritmos de determinação de visibilidade para selecionar os clientes que devem receber cada mensagem, de modo que um cliente não recebe atualizações de objetos que não estão visíveis para ele. Como resultado, o consumo de largura de banda nas conexões com os clientes é reduzido, às custas de um aumento da latência devido ao processamento das mensagens no servidor.

Este sistema foi estudado em relação aos mecanismos de filtragem de mensagens no servidor, embora esta funcionalidade não tenha sido incluída no protótipo da biblioteca.

3.4.8. SIMNET

O SIMNET foi o primeiro AVM de grande escala desenvolvido pelo exército dos Estados Unidos para simulações militares. Este ambiente virtual opera em uma rede de computadores isolada e utiliza o modelo ponto-a-ponto com conexões *broadcast*, sendo este um dos motivos para o uso da rede dedicada (Singhal & Zyda, 1999).

O SIMNET garante a consistência da simulação utilizando a técnica de *dead reckoning* discutida anteriormente e foi um dos primeiros ambientes virtuais a empregar este tipo de técnica.

O protocolo do SIMNET deu origem ao padrão Distributed Interactive Simulations (DIS) que foi normalizado pelo IEEE. Este padrão tem sido usado como base para outros protocolos de ambientes virtuais.

Apesar da definição do padrão DIS, poucos são os ambientes virtuais não-militares que usam este padrão diretamente, pelo fato de ser restrito e limitado a uso para simulações de veículos militares.

3.4.9. World ToolKit

O World ToolKit é um produto comercial produzido pela empresa Sense8, voltado para o desenvolvimento de ambientes virtuais. Essencialmente, trata-se de uma biblioteca para programação em C ou C++ e um conjunto de ferramentas para auxiliar na produção de modelos geométricos. O suporte para AVMs é acrescentado

através do módulo World2World, que utiliza uma arquitetura híbrida envolvendo uma hierarquia de servidores em dois níveis (Sense8, 2002).

Este sistema foi estudado brevemente a fim de compreender a sua arquitetura sob o ponto de vista de uma solução geral para ambientes virtuais.

3.5. Conclusão

Como pôde ser observado ao longo deste capítulo, existem muitas opções para a definição da arquitetura de um ambiente virtual, que dependem da sua aplicação, número de usuários e outros fatores.

Diversos ambientes virtuais multiusuários foram pesquisados, e algumas soluções foram utilizadas como referência para o desenvolvimento da arquitetura da biblioteca de classes. Entre estas soluções estão o uso de uma arquitetura de dados distribuída replicada, a flexibilidade na definição dos métodos de consistência e a definição de uma interface genérica para a construção da interface de usuário.

4. Desenvolvimento da Biblioteca para Criação de Ambientes Virtuais Compartilhados

Este capítulo apresenta a modelagem da arquitetura da biblioteca de classes, assim como os detalhes da implementação de um protótipo desta biblioteca.

Conforme observado anteriormente, um dos problemas no desenvolvimento de ambientes virtuais é a falta de metodologias neste processo. Por este motivo, buscou-se adotar uma metodologia de projeto de *software* orientado a objetos.

Foi escolhida a metodologia de Processo Unificado de Desenvolvimento de Software, conforme descrita por Jacobson et al. (1998). Esta metodologia divide o ciclo de desenvolvimento de *software* em quatro fases: concepção, elaboração, construção e transição. Dentro de cada fase, ocorrem uma ou mais iterações seguindo as etapas tradicionais de projeto: levantamento de requisitos, análise, *design*, implementação e testes. Entretanto, em cada fase algumas destas etapas são mais relevantes do que as outras. O objetivo da metodologia é promover o desenvolvimento progressivo, a partir do estabelecimento de uma arquitetura capaz de suportar as principais funções do sistema.

Este capítulo inicia com a descrição do sistema, de forma textual. Em seguida, se encontram restrições e parâmetros quantitativos do sistema. Estas informações são a base para a elaboração dos casos de uso, na primeira fase da metodologia. Por último, será apresentada a arquitetura desenvolvida para a biblioteca, seguida de uma discussão sobre a sua implementação.

4.1. Proposta

A proposta deste projeto é a criação de uma biblioteca para a construção de ambientes virtuais para múltiplos usuários simultâneos. Esta biblioteca será constituída de componentes de infra-estrutura e componentes de integração (para utilizar o ambiente virtual em uma aplicação específica). Os componentes de infra-estrutura deverão refletir uma arquitetura flexível, capaz de suportar o uso da biblioteca em diferentes aplicações.

Deve-se observar que esta biblioteca de classes não constitui um *framework* porque apesar de estabelecer uma arquitetura para o ambiente virtual, ela não estabelece uma estrutura para a aplicação como um todo.

O ambiente virtual gerado através da biblioteca será uma representação gráfica de um espaço tridimensional finito. Dentro deste espaço, dois ou mais usuários poderão interagir entre si e manipular outros objetos. Espera-se que o ambiente seja capaz de suportar aplicações de pequeno e médio porte, com um limite estimado em 100 usuários simultâneos. Este limite depende, evidentemente, da largura de banda disponível.

Dentro do ambiente virtual, cada usuário será representado por um avatar. Este avatar consistirá em um modelo geométrico (possivelmente com texturas), visível aos outros usuários. Sua função principal será indicar a posição e orientação

dos demais usuários. Adicionalmente, cada avatar poderá ser capaz de realizar animações para indicar a execução de certas ações (por exemplo, pegar um objeto, acenar etc.)

A interação entre usuários poderá ser feita por texto ou voz. A princípio esta comunicação poderá ser tanto ponto-a-ponto (entre dois usuários) como "broadcast" (de um usuário para vários).

Além dos usuários, o ambiente virtual também apresentará outros objetos, que serão classificados em objetos ativos e passivos. Objetos passivos possuem apenas uma representação geométrica mas não permitem qualquer forma de interação, exceto a detecção de colisão.

Objetos ativos permitem um conjunto de operações que podem ser realizadas sobre eles por um avatar, tais como seleção e movimentação (posição e orientação). Adicionalmente, um objeto ativo poderá ter comportamentos mais complexos. De qualquer forma, cada objeto ativo será simulado em um computador ligado ao ambiente virtual. Um mesmo computador poderá simular dois ou mais objetos ativos.

Poderá ser feita a detecção de colisão entre avatares e objetos (ativos e passivos) e entre objetos ativos e avatares ou objetos passivos.

Deve ser possível definir "portais" dentro de um ambiente virtual. Um usuário poderá "atravessar" um destes portais e ser transportado para um outro ambiente virtual. Desta forma, um ambiente virtual de grandes proporções poderá ser dividido em áreas menores ligadas por portais. Exceto pelo tráfego de usuários, não haverá qualquer comunicação entre os dois ambientes ligados por um portal. Isto significa que um participante não poderá "enxergar" através de um portal, nem se

comunicar com usuários do outro lado. Além disso, não está prevista, inicialmente, a possibilidade de um usuário levar consigo objetos de um ambiente para outro.

4.1.1. Restrições

O projeto não pretende abordar técnicas de geração e simulação de fontes de som tridimensionais. Esta é uma área de pesquisa distinta, que envolve a modelagem do ambiente sob o ponto de vista das suas propriedades acústicas, e a simulação de efeitos como reverberação e atenuação do som.

No desenvolvimento do projeto, pretende-se estabelecer uma arquitetura capaz de fornecer flexibilidade no desenvolvimento de interfaces de usuário. Entretanto, não faz parte do projeto efetuar estudos de usabilidade das mesmas.

4.2. Especificação

O objetivo do projeto é definir uma biblioteca de classes para a construção de ambientes virtuais multiusuários. Devem existir classes correspondentes à infra-estrutura do ambiente, e outras classes que podem ser estendidas para satisfazer as necessidades de uma aplicação específica.

A arquitetura de dados adotada seguirá o modelo distribuído replicado, conforme discutido no capítulo 3. Neste modelo, cada participante tem uma cópia completa do ambiente virtual e alterações feitas por ele devem ser propagadas para os demais participantes.

Será feita a distinção entre objetos ativos e passivos dentro do ambiente virtual, representados por classes diferentes. Objetos passivos serão usados essencialmente como cenário, enquanto objetos ativos podem ter comportamentos dinâmicos. Os avatares dos participantes serão objetos ativos.

A partir das escolhas acima, cada participante do ambiente virtual poderá ter em sua base de dados objetos ativos que são simulados localmente, bem como réplicas de objetos ativos remotos. O controle de consistência entre um objeto ativo e suas réplicas é de responsabilidade do próprio objeto.

A comunicação entre objetos ativos do ambiente será feita pela troca de mensagens entre os mesmos, representadas por classes. Cada objeto é identificado por um número único que deve ser suprido pela aplicação. A partir deste mecanismo de mensagens poderão ser construídos sistemas de interação mais complexos, inclusive a comunicação entre usuários.

A biblioteca deverá assumir uma arquitetura de comunicação ponto-a-ponto internamente, ou seja, como se não houvesse um servidor centralizado. Por outro lado, outras classes da biblioteca poderão implementar diferentes arquiteturas de rede. Dessa forma, a biblioteca de classes poderá operar, por exemplo, tanto em TCP (unicast) como UDP (multicast). No protótipo da biblioteca, estão previstas classes para comunicação através destes dois protocolos.

A implementação de portais entre ambientes virtuais poderá ser realizada através do mecanismo de troca de mensagens, juntamente dos componentes de comunicação em rede. Essencialmente, um portal pode ser visto como um objeto ativo que altera as configurações de conexão de rede do participante. Entretanto, não está prevista a implementação de portais no protótipo da biblioteca.

A biblioteca de classes também deve definir interfaces genéricas para permitir aplicações com diferentes tipos de interface de usuário, dispositivos de entrada e dispositivos de exibição. Para o protótipo da biblioteca, está previsto apenas a implementação de classes para dispositivos convencionais: *mouse*, teclado e monitor.

4.3. Arquitetura

A partir da descrição da biblioteca e da sua especificação, foi possível estabelecer uma arquitetura para a biblioteca de classes, que será descrita nesta seção.

Na figura 4-1 pode ser visto um diagrama que representa a arquitetura dos ambientes virtuais construídos com a biblioteca. Como se pode observar, a aplicação é responsável por definir a interface de usuário e o comportamento dos objetos do ambiente, assim como os modelos geométricos destes objetos. A biblioteca de classes DAVI3D (identificada na figura pelo termo "sistema") pode fornecer algumas classes para auxiliar nestas tarefas.

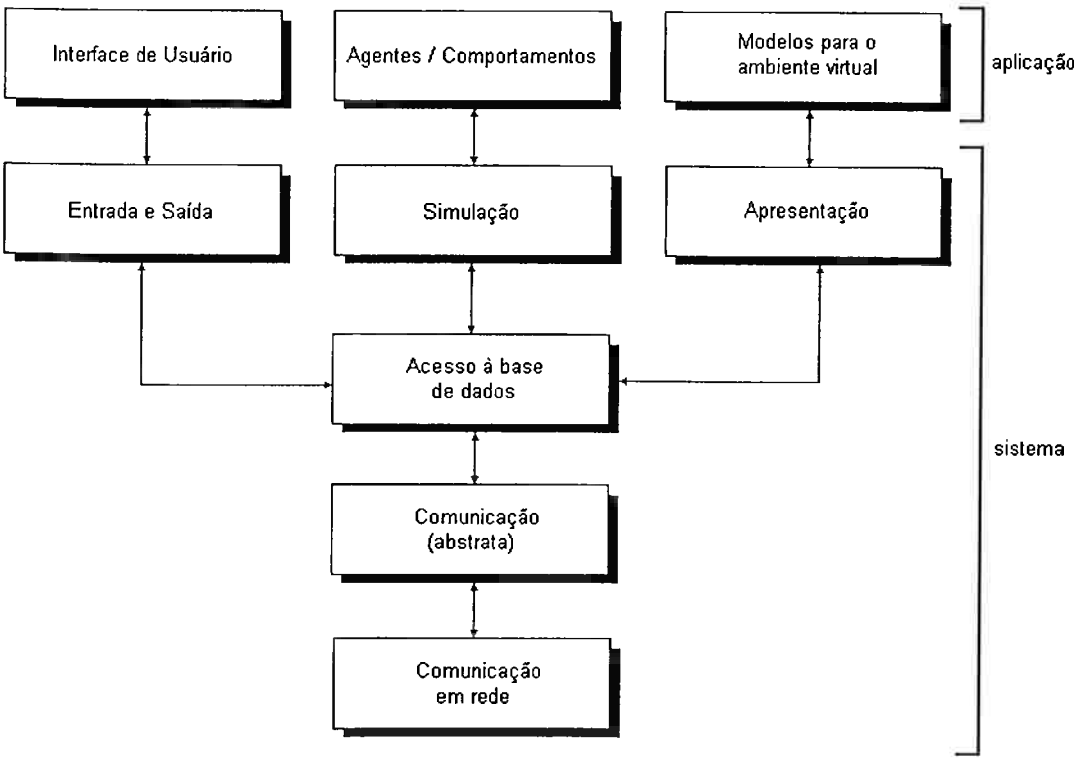


Figura 4-1: Arquitetura da biblioteca DAVI3D

As classes da biblioteca podem ser divididas em seis subsistemas ou componentes, conforme a figura 9: Acesso à Base de Dados, Simulação, Apresentação, Entrada e Saída, Comunicação (abstrata) e Comunicação em Rede.

Como pode ser observado, o subsistema de Acesso à Base de Dados deve determinar as interfaces pelas quais os outros componentes do ambiente virtual têm acesso aos objetos do AVM.

Outro ponto importante é que os dois subsistemas de comunicação definem uma estrutura de duas camadas. Desta forma, a camada de Comunicação (abstrata) define uma interface que simula uma arquitetura ponto-a-ponto, enquanto a camada de Comunicação em Rede implementa os detalhes de comunicação para uma arquitetura ou protocolo específicos.

As seções seguintes discutem em mais detalhes cada um dos subsistemas definidos na arquitetura da biblioteca de classes, incluindo comentários sobre os *design patterns* (Gamma, 1994) adotados.

4.3.1. Acesso à Base de Dados

O componente de acesso à base de dados corresponde à parte central do ambiente virtual, armazenando os objetos que o compõem e controlando o acesso aos mesmos. As classes deste subsistema modelam os objetos ativos e passivos presentes no ambiente virtual e fornecem os serviços básicos para troca de mensagens e detecção de colisão.

A definição de um objeto do ambiente virtual consiste de duas partes: um componente lógico, contendo a posição, orientação e informações de estado do objeto, e um componente visual, contendo sua representação geométrica e eventuais informações sobre animação.

O componente de acesso à base de dados define também uma interface para criação dos objetos do ambiente virtual, baseada no *design pattern* "*Abstract Factory*". Desta forma, aplicações usuárias da biblioteca podem criar "fábricas de objetos" adequadas às suas necessidades.

4.3.2. Comunicação

Os componentes de comunicação são responsáveis pela transmissão dos dados através da rede. Foi feita uma divisão em duas camadas, procurando facilitar a utilização de diferentes tecnologias de rede. Assim, o componente de Comunicação (abstrata) traz uma interface genérica de comunicação, contendo apenas os métodos fundamentais para conexão, desconexão, envio e recepção de mensagens. O componente de Comunicação em Rede é responsável por implementar os métodos reais de comunicação para determinados protocolos e arquiteturas de rede, incluindo métodos para configuração dos parâmetros de conexão. Pode-se comparar esta estrutura ao *design pattern* "*Mediator*" onde a camada de comunicação abstrata reduz o acoplamento entre a base de dados e os protocolos de comunicação.

A comunicação entre objetos do ambiente virtual é feita através da troca de mensagens, de forma similar ao *design pattern* "*Command*". Uma hierarquia de classes descrevendo os diferentes tipos de mensagens é definida, de forma que objetos destas classes possam ser trocados entre os objetos. Aplicações podem estender esta hierarquia construindo mensagens adequadas às suas necessidades.

4.3.3. Simulação

O componente de simulação é responsável pela atualização dos estados dos objetos do ambiente virtual com o passar do tempo. Este componente inclui classes

para controlar a atualização dos objetos, bem como uma interface nos objetos ativos que é chamada para notificá-los da necessidade de atualização.

Desta forma, o componente de simulação opera de forma similar ao *design pattern* "*Observer*". Neste caso, os objetos ativos são os *observers* que se registram junto ao componente de simulação para serem notificados da necessidade de atualizarem seus estados com o passar do tempo.

4.3.4. Entrada e Saída

O componente de entrada e saída permite que outras classes da aplicação, externas à biblioteca, se comuniquem com objetos do ambiente virtual através de um avatar. Novamente utiliza-se o *design pattern* "*Observer*", no qual uma classe da aplicação se registra junto ao avatar para receber as mensagens de interação que chegam até ele.

4.3.5. Apresentação

O componente de apresentação é responsável por configurar o ambiente virtual para um tipo específico de dispositivo de apresentação. Desta forma, este componente procura seguir o *design pattern* "*Bridge*", fazendo com que subclasses do componente básicos implemente os detalhes dependentes de tecnologia específica.

A próxima seção apresenta detalhes da implementação, incluindo uma discussão mais aprofundada de cada um dos subsistemas citados acima.

4.4. Implementação

Para a implementação da biblioteca foi adotada a linguagem Java juntamente à API Java 3D. A escolha da linguagem se deve ao fato de ela disponibilizar, de

maneira simples, os recursos de comunicação em rede e *threads* de controle que são necessários para a construção de ambientes virtuais.

A escolha do Java 3D se deu por três motivos principais: primeiramente, a integração simples com a linguagem, incluindo o fato de ser originalmente uma API que segue a orientação a objetos, ao contrário por exemplo, de APIs que adaptam o acesso do OpenGL para o Java. Em segundo lugar, devido ao modelo de alto nível do grafo de cena, que permite uma abstração natural para os ambientes virtuais, já que a cena pode ser interpretada como o ambiente virtual. Em terceiro lugar, devido aos recursos de *behaviors* (Java 3D, 2002) que permitem a inclusão relativamente simples de comportamentos nos objetos. A biblioteca foi chamada de DAVI3D (Desenvolvimento de Ambientes Virtuais 3D). A implementação da biblioteca foi dividida em dois *packages* separados:

br.usp.pcs.interlab.davi3d: *package* contendo as classes básicas da arquitetura

br.usp.pcs.interlab.davi3d.util: *package* contendo implementações específicas de objetos ativos, objetos passivos, entrada e saída e comunicação.

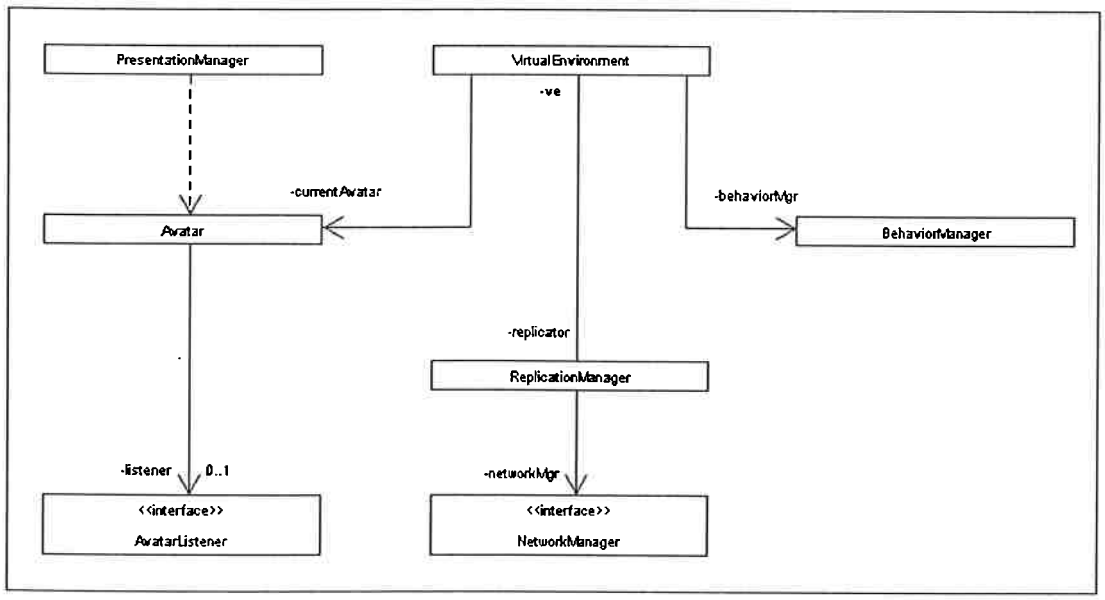


Figura 4-2: Implementação da Arquitetura

A figura 4-2 mostra as principais classes que implementam a arquitetura discutida na seção anterior. Comparando-se este diagrama com a figura 4-1:

- a classe ***VirtualEnvironment*** implementa o componente de Acesso à Base de Dados;
- a classe ***ReplicationManager*** implementa o componente de Comunicação Abstrata;
- a interface ***NetworkManager*** implementa o componente de Comunicação em Rede;
- a classe ***BehaviorManager*** implementa o componente de Simulação;
- as classes ***Avatar***, ***PresentationManager*** e ***AvatarListener*** implementam os componentes de Entrada e Saída e Apresentação.

4.4.1. Acesso à Base de Dados

A figura 4-3 traz um diagrama contendo as classes relacionadas ao subsistema de Acesso à Base de Dados e Simulação.

A classe principal, como discutido anteriormente, é ***VirtualEnvironment***. Esta classe contém métodos públicos para acesso aos objetos do ambiente virtual, além de referências para objetos que fazem a interface dos componentes de simulação e rede. Finalmente, o ***VirtualEnvironment*** contém também um objeto responsável por realizar testes de detecção de colisão.

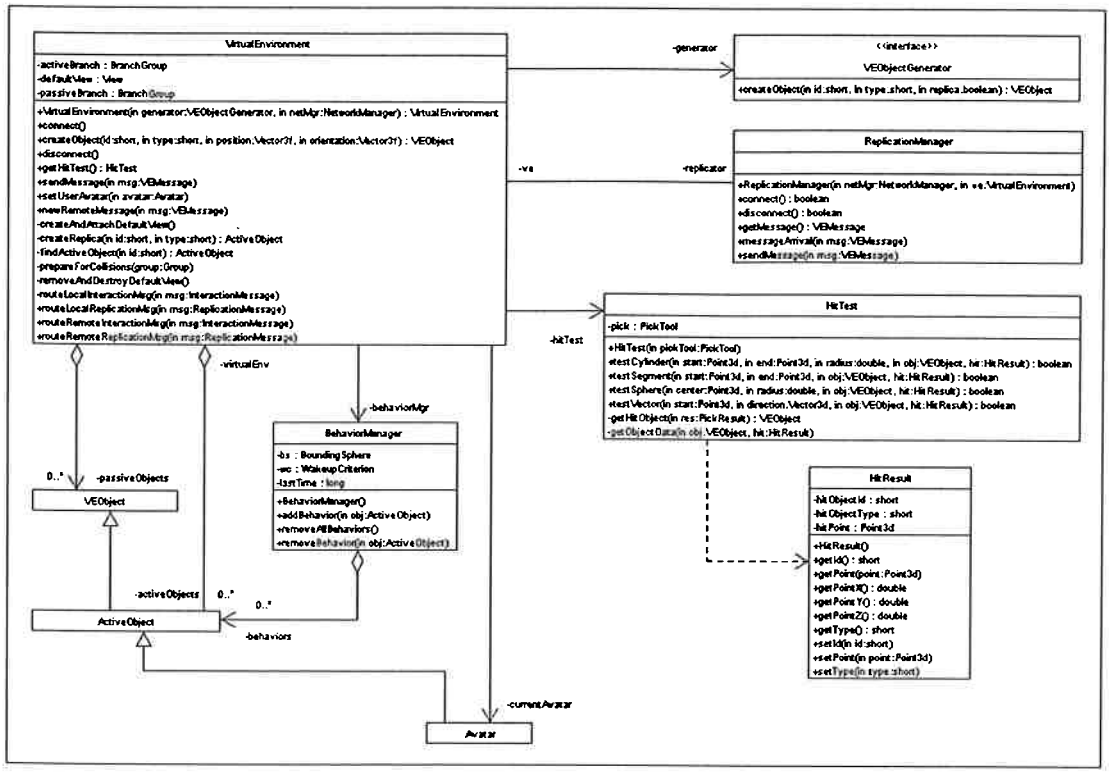


Figura 4-3: Componente de Acesso à Base de Dados

A interface ***VEObjectGenerator*** é responsável pela criação de todos os objetos (ativos e passivos) do AVM (*design pattern* "*Abstract Factory*"). Essencialmente, cada objeto possui um número que identifica o seu "tipo". Esta interface constrói um novo objeto baseado nesta informação de tipo.

A classe ***HitTest*** implementa o sistema de detecção de colisão através do mecanismo de *picking* do Java 3D. São fornecidos métodos que verificam a interseção de um objeto do ambiente virtual com uma forma geométrica que pode ser um vetor, segmento de reta, cilindro ou esfera. Um objeto ativo pode usar os métodos desta classe para determinar se uma mudança de posicionamento irá acarretar uma colisão, e pode ainda obter informações sobre o objeto com o qual iria colidir.

Estes mesmos métodos de detecção de colisão podem ser usados para implementar mecanismos de seleção de objetos.

A classe BehaviorManager é a principal classe do subsistema de Simulação. Esta classe é derivada da classe Behavior do Java 3D e é responsável pela atualização do estado dos objetos ativos do AVM.

As demais classes pertencem a outros subsistemas e estão presentes na figura apenas para ilustrar os relacionamentos com a classe ***VirtualEnvironment***. Elas serão discutidas nas próximas seções.

4.4.2. Objetos do Ambiente Virtual

A figura 4-4 apresenta as classes que são usadas para modelar os objetos do ambiente virtual (objetos passivos e ativos, incluindo avatares).

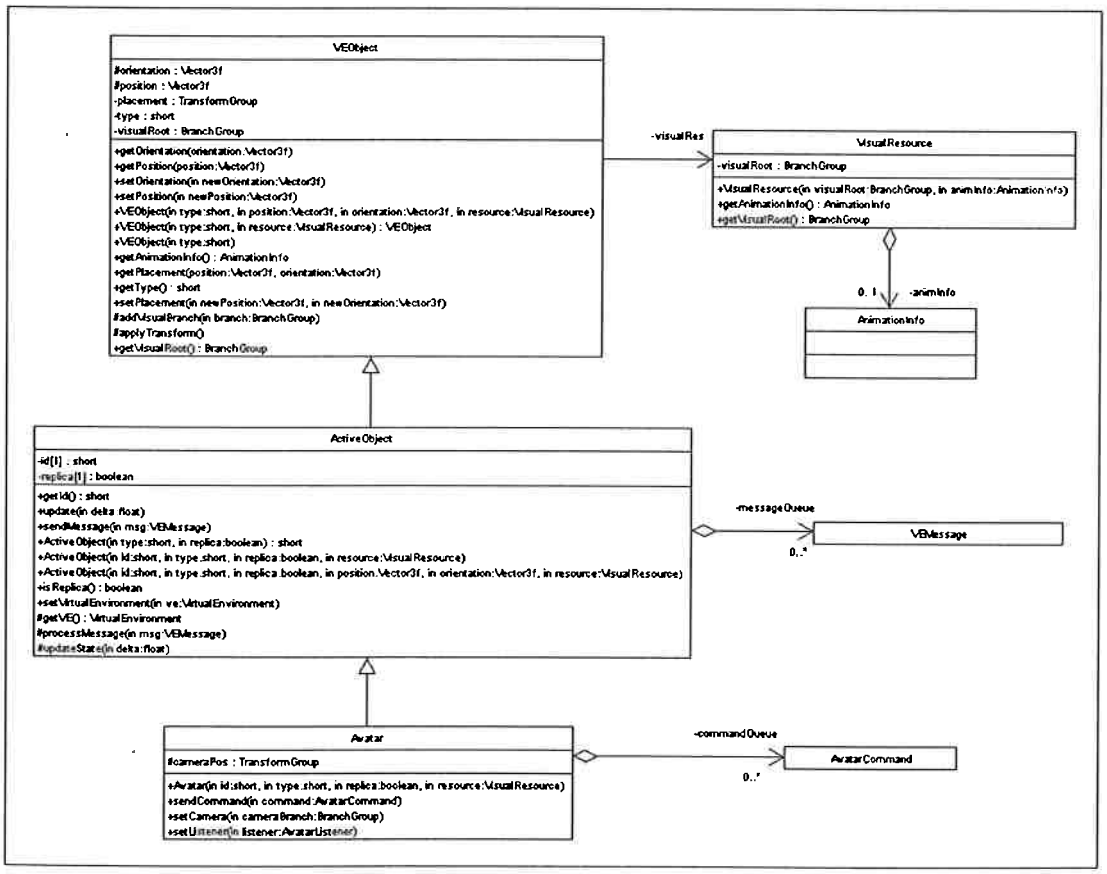


Figura 4-4: Diferentes tipos de Objetos do Ambiente Virtual

Como se pode ver na figura, cada objeto da classe ***VEObject*** possui um objeto da classe ***VisualResource*** associado, que contém informações sobre a sua representação geométrica.

Existe ainda a classe ***AnimationInfo*** que representa um conjunto de informações para realizar animações daquela representação geométrica. Na implementação da arquitetura básica, esta classe não define nenhum tipo de animação padrão. Entretanto, os objetos ativos podem obter uma referência para este objeto. A intenção é que diferentes tipos de animação possam ser implementados, com subclasses de ***AnimationInfo*** adequadas.

Cada ***ActiveObject*** possui ainda uma fila de mensagens correspondente às mensagens recebidas de outros objetos. As classes de mensagens serão discutidas numa outra seção.

Uma última observação importante quanto à classe ***ActiveObject*** se refere à variável de estado "replica". Esta variável indica se o objeto, na base de dados, está sendo simulado localmente, ou é uma réplica de um objeto simulado em outro computador. Através desta variável, é possível implementar diferentes métodos de consistência entre objetos.

4.4.3. Comunicação

Como foi mencionado anteriormente, a comunicação é dividida em uma camada abstrata e outra que implementa uma tecnologia de redes específica. Na figura 4-5, as classes ***ReplicationManager***, ***MessageSender*** e ***MessageReceiver*** correspondem à camada abstrata. A interface ***NetworkManager*** corresponde à camada específica.

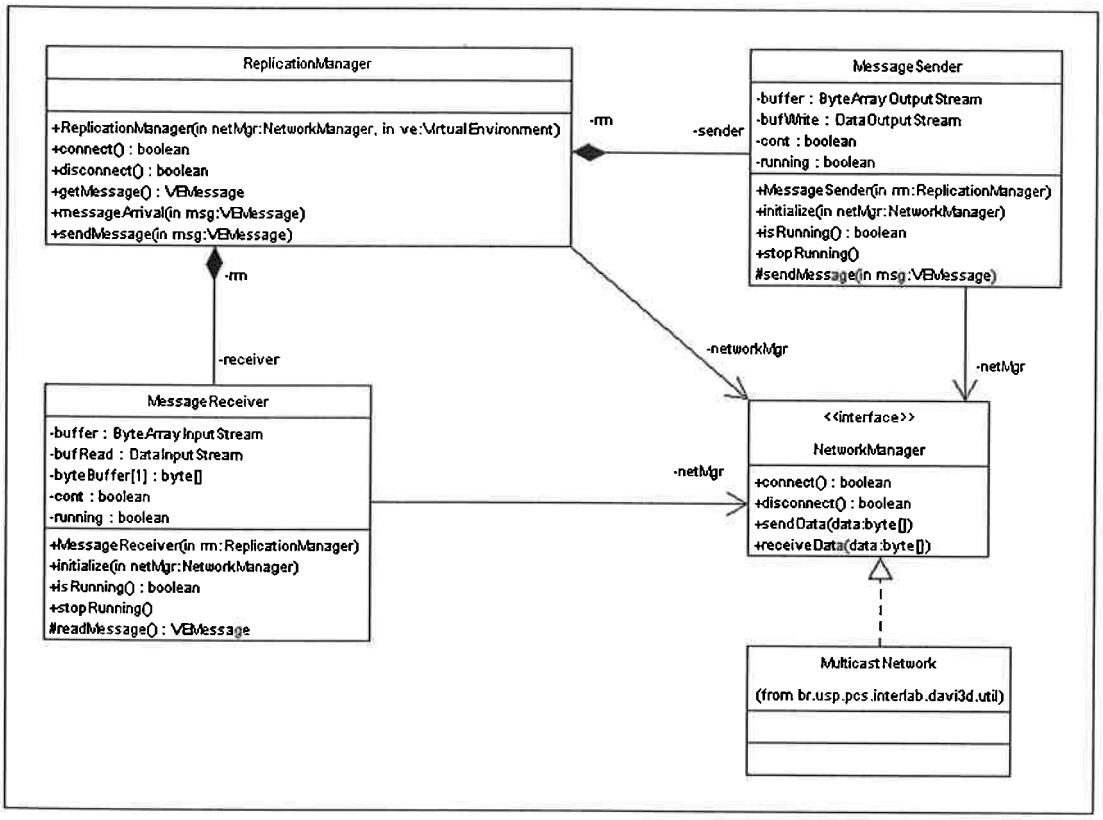


Figura 4-5: Comunicação em Rede

As classes ***MessageSender*** e ***MessageReceiver*** são *threads* que se comunicam com um objeto que implementa a interface ***NetworkManager***. Por exemplo, na figura é mostrada a classe ***MulticastNetwork***, que realiza comunicação através do protocolo IP *Multicast*. Esta classe se encontra no *package* de classes utilitárias da biblioteca.

A interface ***NetworkManager*** define os serviços mínimos necessários para qualquer interface de rede: a capacidade de se conectar e desconectar da rede, e enviar e receber dados. Classes que implementem esta interface são responsáveis por definir métodos específicos para configuração da conexão, de acordo com a tecnologia de rede adotada. Por exemplo, numa implementação cliente-servidor, seria preciso indicar o endereço IP no qual o servidor está localizado, para permitir a conexão.

4.4.4. Entrada e Saída

Na figura 4-6 temos a representação das classes dos componentes de Entrada e Saída e Apresentação. A classe ***PresentationManager*** faz a configuração das classes do Java 3D para um determinado tipo de dispositivo de exibição. Por exemplo, a subclasse ***DesktopPresentation*** (do *package* de classes utilitárias) faz esta configuração para monitores de vídeo.

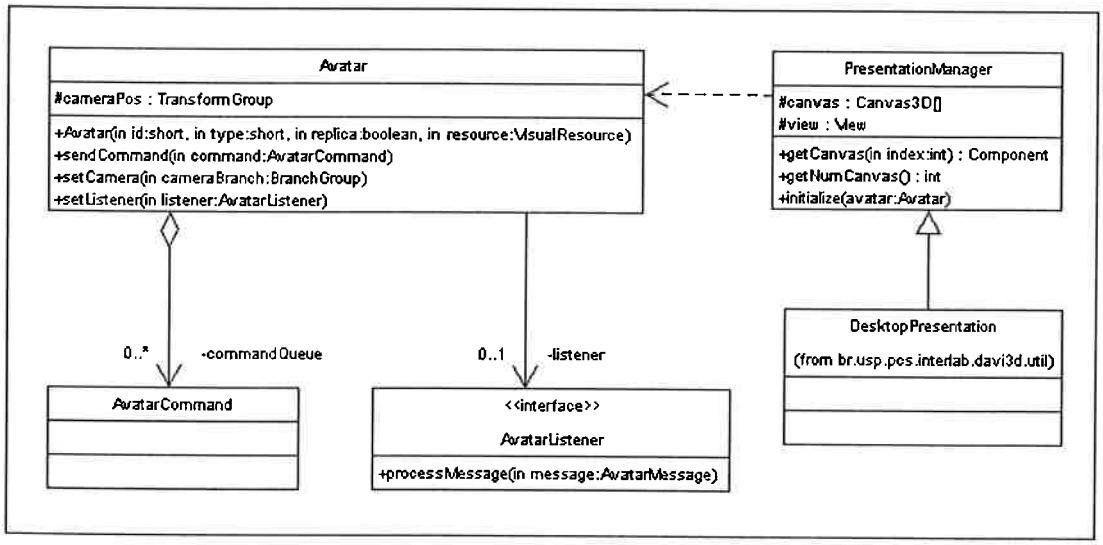


Figura 4-6: Entrada e Saída

O método público ***sendCommand*** da classe ***Avatar***, juntamente da interface ***AvatarListener***, permitem que seja construída uma interface de usuário para interagir com o ambiente virtual.

Essencialmente, uma aplicação define subclasses da classe ***AvatarCommand*** contendo diferentes tipos de comandos que podem ser enviados para o ***Avatar***. A aplicação pode também criar uma subclasse de ***Avatar*** para tratar estes comandos.

A aplicação pode implementar uma classe com a interface ***AvatarListener*** para receber as mensagens que são enviadas para o avatar. Através deste mecanismo, é possível implementar a interação entre usuários, por exemplo.

4.4.5. Tipos de Mensagens

Finalmente, a figura 4-7 apresenta a hierarquia de classes usada para troca de mensagens entre objetos do ambiente virtual.

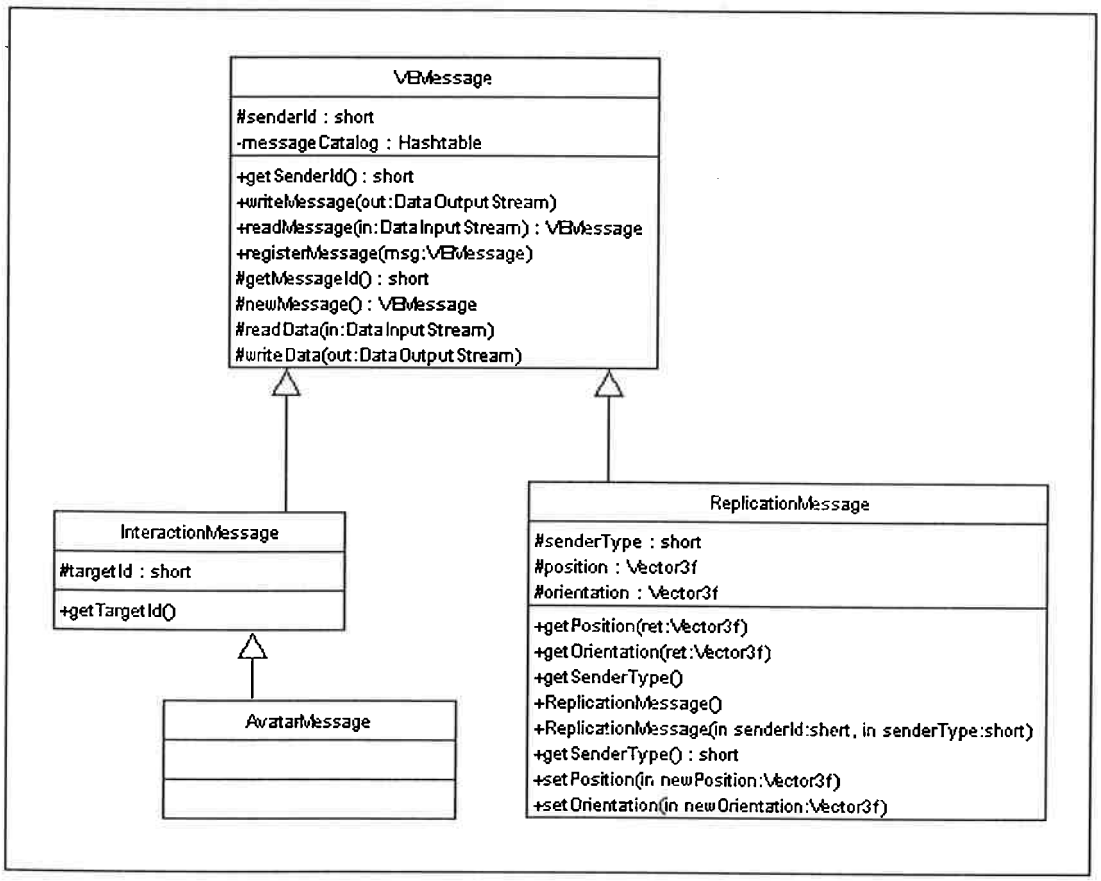


Figura 4-7: Tipos de Mensagens

A classe ***VEMessage*** representa um tipo genérico de mensagem e suas duas subclasses correspondem aos dois tipos básicos de mensagem: interação e replicação. A subclasse ***AvatarMessage*** é um caso particular de mensagem de interação, que é repassado para um ***AvatarListener*** (discutido na seção anterior).

Os dois tipos de mensagem, ***InteractionMessage*** e ***ReplicationMessage*** são usados para duas finalidades específicas. ***InteractionMessage*** corresponde a mensagens de interação definidas pela aplicação. Podem ser utilizadas, por exemplo,

para implementar um *chat* entre usuários, ou para ativar algum serviço de um objeto ativo.

As subclasses de ***ReplicationMessage*** correspondem a dados de atualização de estado dos objetos ativos. Estas mensagens são usadas para implementar um mecanismo de replicação e consistência entre os participantes do ambiente virtual.

4.5. Extensibilidade e Manutenção

Nesta seção serão discutidos os aspectos de extensibilidade e manutenção da biblioteca desenvolvida.

Desde a definição da sua arquitetura, a biblioteca foi projetada para ser extensível. De fato, vários de seus componentes definem apenas uma interface que deve ser implementada pela aplicação usuária da biblioteca.

O fato de que os componentes de apresentação, comunicação em rede e objetos ativos podem ser expandidos e modificados leva à conclusão de que a biblioteca pode ser ampliada conforme as necessidades de cada aplicação sem dificuldades aparentes.

O aspecto da manutenção é um fator importante em todo sistema de software, e especialmente em sistemas orientados a objetos, onde ela pode ser facilitada ou dificultada de acordo com o projeto das classes.

No caso desta biblioteca de classes, procurou-se estabelecer interfaces de métodos públicos muito bem definidos entre as classes que compõem a arquitetura básica, usando o encapsulamento para isolar as partes mais sujeitas a mudanças.

Adicionalmente, as classes foram particionadas de forma que responsabilidades distintas sejam tratadas por classes separadas, como se pode observar nos diagramas de classes das seções anteriores.

A adoção destas medidas buscou fazer com que a manutenção da biblioteca possa ocorrer de forma localizada e sem interferir no restante do sistema.

4.6. Testes da Biblioteca

A partir do protótipo da biblioteca, foi possível construir alguns ambientes virtuais simples para teste do protótipo. Estes testes são apresentados abaixo.

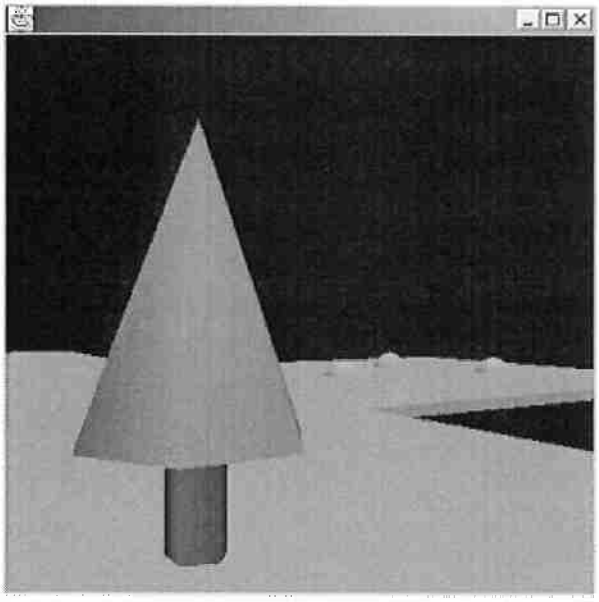


Figura 4-8: Ambiente Mono-usuário

A figura 4-8 apresenta o primeiro teste realizado. Neste ambiente virtual, não foi utilizado nenhum tipo de ***NetworkManager***. O resultado foi um ambiente virtual para um único usuário, pois o subsistema de comunicação em redes não é inicializado. Desta forma, conclui-se que a biblioteca desenvolvida também pode ser usada para ambientes virtuais mono-usuário.

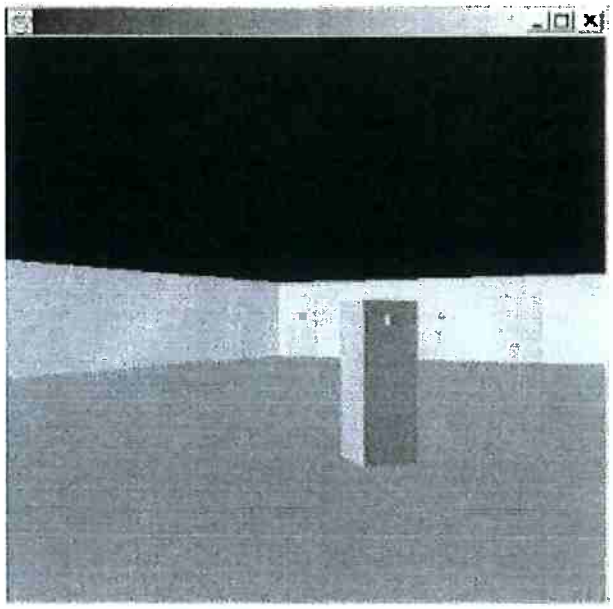


Figura 4-9: Ambiente Multiusuário

A figura 4-9 exibe uma imagem de um ambiente virtual multiusuário desenvolvido com a biblioteca. Neste caso, foi usada a classe ***MulticastNetwork*** para realizar a comunicação em rede através do protocolo IP *Multicast* (UDP). No centro da figura é possível observar o avatar de outro participante do AVM.

Estes testes foram realizados para verificar a utilização da biblioteca de classes no desenvolvimento de ambientes virtuais. Não foram realizados testes de desempenho das aplicações desenvolvidas.

4.7. Conclusão

Este capítulo apresentou os detalhes de desenvolvimento e implementação da biblioteca. O protótipo resultante demonstrou as características de flexibilidade especificadas, que puderam ser observadas na construção de dois ambientes virtuais simples.

Os testes e protótipos poderiam ser expandidos aproveitando-se a arquitetura da biblioteca. Por exemplo, seria possível implementar mecanismos de nível de

detalhamento nos objetos ativos. Da mesma forma, seria possível experimentar outros protocolos de rede, incluindo protocolos de tempo real. Entretanto, nada disso foi feito por divergir do objetivo principal do trabalho que é a definição de uma arquitetura flexível para a biblioteca de classes, e o teste desta arquitetura através de um protótipo básico.

Deve-se observar que a API Java 3D ainda se encontra em desenvolvimento. Durante o desenvolvimento do protótipo ocorreram alguns problemas devido a este fato, tais como erros e instabilidade no código gerado. Entretanto, por se tratar de um protótipo utilizando uma nova tecnologia, acredita-se que os resultados obtidos ainda sejam válidos e importantes.

5. Aplicação: A Exposição Virtual Multiusuário

Neste capítulo será feita uma introdução sobre o projeto Museu Virtual 3D desenvolvido no Laboratório de Tecnologias Interativas (Interlab). Em seguida, será apresentado o relato da integração entre este projeto e o protótipo da biblioteca DAVI3D para a disponibilização de conteúdo gerado com o museu virtual para múltiplos usuários simultâneos.

5.1. Sobre o Museu Virtual 3D

A Exposição Virtual Multiusuário foi desenvolvida com o propósito de testar a biblioteca em uma aplicação real e se baseia em um trabalho desenvolvido no Interlab como tema de uma dissertação de Mestrado. O projeto inclui o desenvolvimento de um editor de exposições virtuais que são gravadas em um formato de arquivo próprio e apresentadas utilizando-se o Java 3D (Tori, 2000).

Para testar o funcionamento da biblioteca DAVI3D, foi desenvolvido um ambiente virtual multiusuário capaz de ler arquivos de configuração gerados pelo editor.

5.2. Desenvolvimento

Para o desenvolvimento do AVM a partir das exposições virtuais, foi decidido construir uma arquitetura cliente-servidor, usando o protocolo TCP/IP.

O servidor utiliza duas portas de conexão. A primeira é utilizada para fornecer informações de configuração aos participantes sobre a exposição virtual que está sendo utilizada no ambiente virtual. A outra porta é utilizada para clientes já configurados se conectarem ao ambiente virtual.

A figura 5-1 ilustra esta arquitetura, apresentando três participantes em situações distintas. O *software* cliente de um participante se conecta inicialmente à porta de configuração. Nesta etapa, o cliente é informado sobre a exposição virtual que está sendo exibida e pode se configurar internamente.

Em seguida, o cliente acessa a porta de conexão, para estabelecer uma conexão para troca de dados com outros participantes do ambiente. A partir deste ponto, o novo participante se encontra efetivamente conectado e visível para os demais participantes.

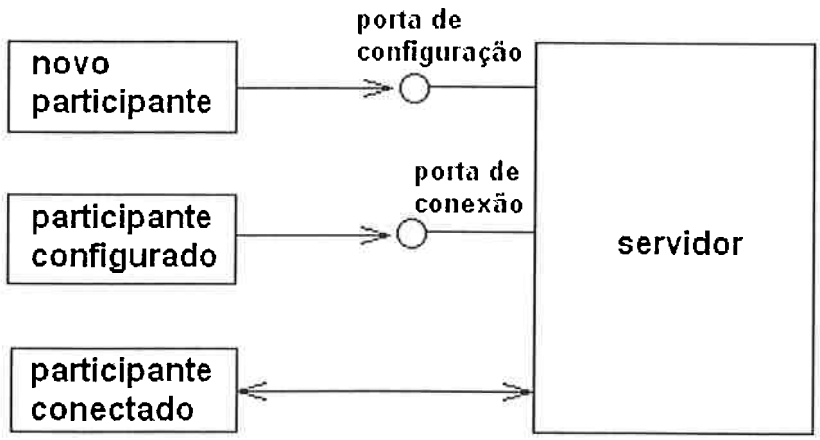


Figura 5-1: Arquitetura Cliente-Servidor do Museu Virtual

Foi desenvolvida uma classe, ***MultiExpoGenerator***, implementando a interface ***VEObjectGenerator*** especificamente para esta aplicação. Esta classe é capaz de construir dois tipos de objetos: um ***VEObject*** contendo toda a estrutura da cena e um ***Avatar***. Inicialmente, foi adotado um modelo geométrico extremamente simples para o avatar.

A classe ***ExpoLoader*** foi criada para ler e interpretar um arquivo gerado pelo editor de exposições virtuais e construir a cena da exposição virtual, usando classes auxiliares do Java 3D para este fim. Esta classe é utilizada internamente pelo ***MultiExpoGenerator***.

Para o ***Avatar*** foi utilizada a classe ***WalkerAvatar*** do *package* de classes auxiliares da biblioteca DAVI3D. Essa classe implementa um avatar capaz de se mover horizontalmente e girar em torno do seu eixo vertical. O avatar também implementa detecção de colisão.

Para a comunicação foram desenvolvidas duas classes, uma para o cliente e outra para o servidor. A classe do cliente, denominada ***MultiExpoClient*** implementa a interface ***NetworkManager***, bem como o protocolo de configuração.

O servidor desenvolvido foi bastante simples. Essencialmente, a porta de configuração recebe conexões dos clientes e retorna informações sobre o arquivo de configuração que contém os dados da exposição virtual.

Ao mesmo tempo, o servidor aguarda clientes que desejam participar do ambiente virtual e cria conexões permanentes (TCP/IP) para cada um deles. Quando uma mensagem chega a partir de um cliente, o servidor apenas envia a mesma mensagem para todos os demais clientes conectados. A figura 5-2 traz um diagrama de classes simplificado contendo a estrutura da aplicação que foi discutida acima.

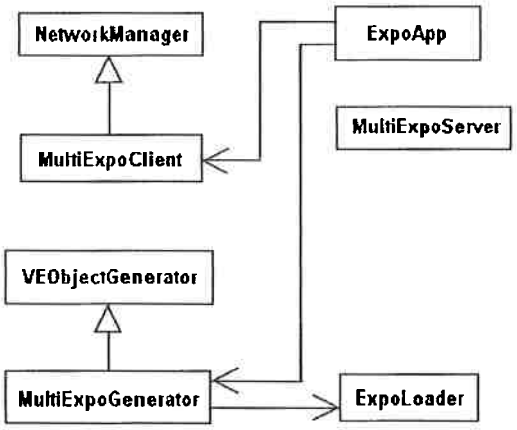


Figura 5-2: Classes desenvolvidas para a Exposição Virtual Multiusuário

5.3. Resultados

A figura 5-3 apresenta uma imagem do resultado do projeto, com a aplicação cliente exibindo uma exposição virtual. O avatar de outro usuário pode ser visto no centro da figura.

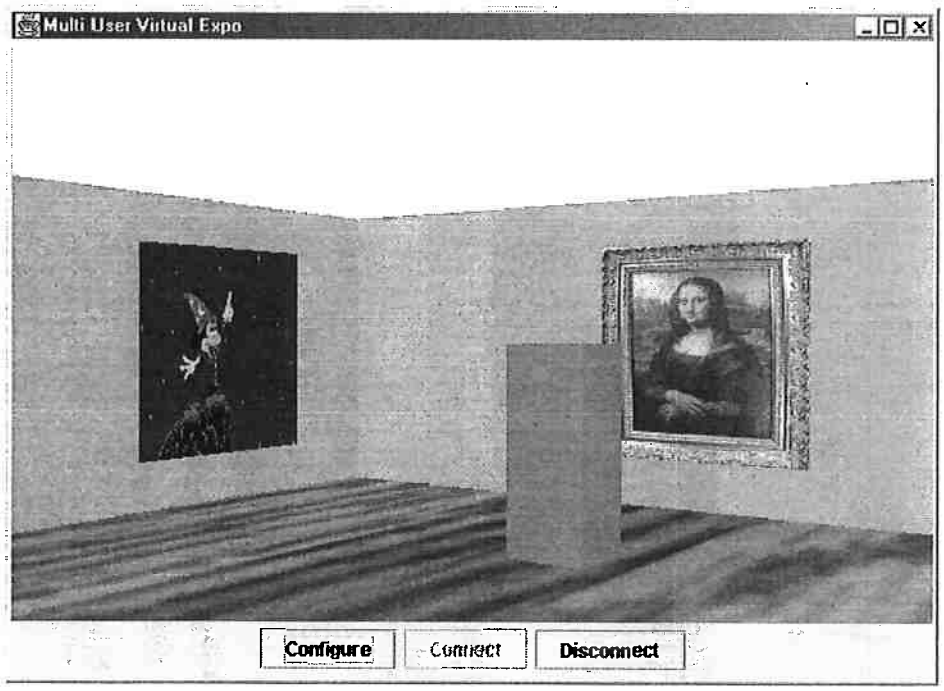


Figura 5-3: Exposição Virtual Multiusuário

Nesta versão inicial da aplicação, não foi implementada nenhuma forma de interação entre os usuários, como por exemplo, a troca de mensagens de texto. Adicionalmente, nesta versão a porta de conexão do servidor informa apenas o nome do arquivo de configuração da exposição virtual; assume-se que tal arquivo, assim como os arquivos de imagem para texturas e outros recursos já estejam presentes na máquina do cliente. Numa versão mais elaborada, seria possível construir um protocolo para transferir os arquivos necessários durante a etapa de configuração.

Conclui-se, a partir desta versão inicial do aplicativo Exposição Virtual Multiusuário que foi possível desenvolver uma aplicação completa envolvendo um ambiente virtual multiusuário, utilizando o protótipo da biblioteca DAVI3D.

6. Conclusão

6.1. Avaliação dos Resultados

Uma biblioteca de classes para o desenvolvimento de ambientes virtuais multiusuários, denominada DAVI3D, foi modelada seguindo a metodologia de projeto de *software* denominada Processo Unificado. Um protótipo desta biblioteca foi desenvolvido utilizando-se a linguagem Java e a API Java 3D.

Ao final do desenvolvimento, foi possível concluir que a metodologia de projeto adotada foi adequada para a tarefa, permitindo o estabelecimento de uma arquitetura bem definida para a biblioteca de classes.

A escolha do Java e Java 3D para a construção do protótipo se mostrou adequada, sendo que os recursos de comunicação em rede disponíveis na linguagem Java e a estrutura de alto nível do grafo de cena do Java 3D simplificaram as tarefas de implementação da biblioteca de classes.

Como foi observado pela construção de alguns protótipos de teste e da aplicação Exposição Virtual Multiusuário, a biblioteca de classes DAVI3D permitiu o desenvolvimento de diferentes ambientes virtuais multiusuários e mono-usuário,

com graus de complexidade diversos. Desta forma, foi atingido o objetivo de definir uma arquitetura flexível e extensível para a biblioteca de classes e construir um protótipo desta biblioteca.

Com a construção da primeira versão da Exposição Virtual Multiusuário, foi cumprido o objetivo de integrar os resultados deste projeto com o projeto do Museu Virtual 3D e editor de exposições virtuais do Interlab.

6.2. Contribuições

Pode-se destacar como principais contribuições deste trabalho a arquitetura da biblioteca de classes desenvolvida, bem como o seu protótipo. Adicionalmente, o desenvolvimento deste protótipo permitiu a obtenção de informações sobre a utilização da API Java 3D em projetos de Realidade Virtual.

O ambiente virtual multiusuário para exposições virtuais foi outra contribuição importante, realizando a integração com outro projeto do laboratório e permitindo a exploração de novas possibilidades a partir deste resultado.

A biblioteca DAVI3D, por ser um *software* aberto e gratuito, poderá ser utilizada em atividades práticas de cursos de Computação Gráfica e Realidade Virtual.

Finalmente, pode-se dizer que os resultados deste trabalho, principalmente o modelo da biblioteca de classes, poderão servir como ponto de partida para novas pesquisas e projetos ligados a ambientes virtuais multiusuários.

6.3. Trabalhos Futuros

O protótipo da biblioteca de classes desenvolvido ao longo deste trabalho permitiu verificar a viabilidade da sua arquitetura no desenvolvimento de ambientes

virtuais multiusuários. Entretanto, alguns elementos não foram implementados nesse protótipo e podem ser desenvolvidos futuramente, destacando-se:

- classes para realizar a animação de avatares e objetos ativos;
- portais para conectar dois ou mais ambientes virtuais.

Adicionalmente, é possível expandir a biblioteca acrescentando classes para interação com outros dispositivos de entrada e saída, e dispositivos de exibição tais como CAVE.

Pretende-se disponibilizar a biblioteca de classes para utilização em projetos dos cursos de Computação Gráfica do curso de graduação em Engenharia de Computação e possivelmente, para os projetos de conclusão de curso.

Referências Bibliográficas

ANDERSON, D. B. et al. Building Multi-user Interactive Multimedia Environments at MERL. **IEEE MultiMedia**, v. 2, n. 4, p. 77-82, 1995.

AQUINO, P. T., KIRNER, T. G. Desenvolvimento de um Sistema de Gerenciamento de Dados e Arquivos para um Ambiente Virtual Colaborativo. SBC SYMPOSIUM ON VIRTUAL REALITY, 4., 2001, Gramado. **Anais...** São Paulo: SBC, 2001. p. 326-337.

BARAFF, D. Interactive Simulation of Solid Rigid Bodies. **IEEE Computer Graphics and Applications**, v. 15, n. 5, p. 63-75. Maio 1995.

BARRUS, J. W. et al. Locales And Beacons: Efficient And Precise Support For Large Multi-User Virtual Environments. In: IEEE VIRTUAL REALITY ANNUAL INTERNATIONAL SYMPOSIUM, Santa Clara, 1996. **Anais...** IEEE Computer Society Press, 1996. p. 204-213.

BOWMAN, D., et al. Travel in immersive virtual environments: an evaluation of viewpoint motion control techniques. In: IEEE VIRTUAL REALITY ANNUAL

INTERNATIONAL SYMPOSIUM, 1997. **Anais...** IEEE Computer Society Press, 1997. p. 45-52.

BOWMAN, D., HODGES, L. An evaluation of techniques for grabbing and manipulating remote objects in immersive virtual environments. In: ACM SYMPOSIUM ON INTERACTIVE 3D GRAPHICS, 1997. **Anais...** ACM Press, 1997. p. 35-38.

BOWMAN, D., HODGES, L. User Interface Constraints for Immersive Virtual Environment Applications. Georgia: Georgia Institute of Technology, 1995. (Relatório Técnico GITGVU -95-26).

BROLL, W. DWTP - An Internet Protocol For Shared Virtual Environments. In: INTERNATIONAL SYMPOSIUM ON THE VIRTUAL REALITY MODELING LANGUAGE, 1998. **Anais...** ACM Press, 1998. p. 49-56.

BROLL, W. Bringing People Together - An Infrastructure For Shared Virtual Worlds On The Internet. In: IEEE WORKSHOP ON ENABLING TECHNOLOGIES: INFRASTRUCTURE FOR COLLABORATIVE ENTERPRISES, 6., 1997, Cambridge. **Anais...** Los Alamitos: IEEE Computer Society Press, 1997. p. 199-204.

BRUTZMAN, D. P. et al. Internetwork Infrastructure Requirements for Virtual Environments. In: VRML SYMPOSIUM, 1995, San Diego. **Anais...** San Diego: ACM Press, 1995. p. 95-104.

BURDEA, G., COIFFET, P. **Virtual Reality Technology**. Nova York: John Wiley & Sons, 1994. 400p.

DIRECTX Home Page. *Site* sobre a biblioteca para desenvolvimento de aplicações multimídia DirectX. Disponível em <http://www.microsoft.com/directx>. Acessado em: 19 mar 2002.

DUCHAINEAU, M. et al. ROAMing terrain: real-time optimally adapting meshes. In: IEEE VISUALIZATION, 1997. **Anais...** Los Alamitos: IEEE Computer Society Press, 1997. p. 81-88.

ELLIS, S. What Are Virtual Environments? **IEEE Computer Graphics and Applications**, v. 14, n. 1, p. 17-22, Janeiro 1994.

FOLEY, J. D. et al. **Computer Graphics:** principles and practice. 2. ed. Reading: Addison-Wesley, 1996. 1175p.

FRASER, M. et al. Supporting Awareness and Interaction through Collaborative Virtual Interfaces. **CHI Letters**. v. 1, n. 1, p. 27-36, 1999.

FRÉCON, E., NÖU, A. A. Building Distributed Virtual Environments to Support Cooperative Work. In: ACM SYMPOSIUM ON VIRTUAL REALITY SOFTWARE AND TECHNOLOGY, 1998 Taipei. **Anais...** ACM Press, 1998. p. 105-119.

FUNKHOUSER, T. A. RING: A Client-Server System For Multi-User Virtual Environments. In: SYMPOSIUM ON INTERACTIVE 3D GRAPHICS, 1995, Monterrey. **Anais...** ACM Press, 1995. p. 85-92.

FUNKOUSER, T. A., SEQUIN, C. H. Adptive display algorithm for interactive frame rates during visualization of complex virtual environments. In: SIGGRAPH, 1993. **Anais...** ACM Press, 1993. p. 247-254.

GALLI, R.; LUO, Y. Mu3d: A Causal Consistency Protocol For A Collaborative Vrml Editor. In: VRML SYMPOSIUM, 2000, Monterey. **Anais...** ACM Press, 2000.

GAMMA, E. et al. **Design Patterns:** Elements of Reusable Object-Oriented Software. Boston: Addison-Wesley, 1994. 416p.

HARADA, C. Y. **Arquitetura para Suporte a Aplicações Colaborativas**. 2001. 217p. Dissertação de Mestrado – Escola Politécnica, Universidade de São Paulo, São Paulo.

HARMON, R. et. al. The Virtual Annotation System. In: IEEE VIRTUAL REALITY ANNUAL INTERNATIONAL SYMPOSIUM, 1996, Santa Clara. **Anais...** IEEE Computer Society Press, 1996. p. 239-245.

HEARN, D., BAKER, M. P. **Computer Graphics.** 2. ed. London: Prentice-Hall International, 1994. 652p.

HINDMARSH, J. et al. Fragmented Interaction: Establishing Mutual Orientation in Virtual Environments. In: ACM CONFERENCE ON COMPUTER SUPPORTED CO-OPERATIVE WORK, Seattle, 1998. **Anais...** ACM Press, 1998. p. 217-226.

JACOB, L. J. et al. Avaliação de Técnicas de Interação em Ambientes Imersivos: Uma Proposta de Aplicação no Tratamento de Fobia de Altura. In: SBC SYMPOSIUM ON VIRTUAL REALITY, 4., 2001, Florianópolis. **Anais...** São Paulo: SBC, 2001. p. 12-22.

JACOBSON, I. et al. **The Unified Software Development Process**. Addison-Wesley. 1998. 463p.

JAVA 3D API Homepage. *Site* com informações sobre a biblioteca gráfica Java 3D. Disponível em: <http://java.sun.com/products/java-media/3D/index.html>. Acesso em: 19 mar. 2002.

JOHNSON, A. et al. The NICE Project: Learning Together in a Virtual World. In: IEEE VIRTUAL REALITY ANNUAL INTERNATIONAL SYMPOSIUM, 1998, Atlanta. **Anais...** IEEE Computer Society Press, 1998. p. 176-183.

KAZMAN, R. Making WAVES: On the Design of Architectures for Low-end Distributed Virtual Environments. In: IEEE VIRTUAL REALITY ANNUAL INTERNATIONAL SYMPOSIUM, 1993. **Anais...** IEEE Computer Society Press, 1993. p. 443-449.

KIM, G. J. et al. Software engineering of virtual worlds. ACM SYMPOSIUM ON VIRTUAL REALITY SOFTWARE AND TECHNOLOGY, 1998 Taipei. **Anais...** ACM Press, 1998. p. 131-138.

KIRNER, T. G., et. al. Development Of A Collaborative Virtual Environment For Educational Applications. In: ACM ANNUAL WEB3D SYMPOSIUM. 6., 2001, Paderborn. **Anais...** ACM Press, 2001. p. 61-68.

KOIFMAN, A., ZABELE, S. RAMP: A Reliable Adaptive Multicast Protocol. In: IEEE INFOCOM, 1996, San Francisco. **Anais...** San Francisco: IEEE, 1996. p. 1442-1451.

KUBO, M. M. **Suporte para Comunicação para Sistemas Colaborativos Interoperáveis de Realidade Virtual**. 2000. 210p. Dissertação de Mestrado – Centro de Ciências Exatas e Tecnologia, Universidade Federal de São Carlos, São Carlos.

KUBO, M. M. et. al. VIRTUAL VIDEOCONFERENCE ROOM (VVR) - Uma Aplicação de Trabalho Cooperativo Suportado por Computador Baseada nas Técnicas de Realidade Virtual. In: WORKSHOP BRASILEIRO DE REALIDADE VIRTUAL, 2., 1999, Marília. **Anais...** São Paulo: SBC, 1999. p. 1-22.

LIN, M., GOTTSCHALK, S. Collision Detection Between Geometric Models: A Survey. In: IMA Conference on Mathematics of Surfaces, 1998, San Diego. **Anais...** 1998. p. 37-56.

LU, T. et al. Control Mechanism For Large-Scale Virtual Environments. **Journal of Visual Languages and Computing**. n. 10, p. 69-85, 1999

MACEDONIA, M. R., ZYDA, M. A. Taxonomy For Networked Virtual Environments. **IEEE Multimedia**, v. 1, n. 4, p. 48-56, Janeiro 1997.

MACEDONIA, M. R. et al. NPSNET: A Multi-player 3D Virtual Environment Over the Internet. In: SYMPOSIUM ON INTERACTIVE 3D GRAPHICS, 1995, Monterrey. **Anais...** ACM Press, 1995. p. 93-94.

MÄNTYLÄ, M. **An Introduction to Solid Modeling**. Rockville: Computer Science Press, 1988. 401p.

MARTINS, V. F. et al. Análise, Projeto E Implementação De Uma Aplicação De Realidade Virtual Cooperativa. In: WORKSHOP BRASILEIRO DE REALIDADE VIRTUAL, 2., 1999. **Anais...** São Paulo: SBC, 1999. p. 43-55.

MORSE, K. L. Interest Management In Distributed Large Scale Simulations. **IEEE Computer**, v. 28, n. 7, p. 49-56, Julho 1996.

OLIVEIRA, M. et. al. Components For Distributed Virtual Environments. In: VIRTUAL REALITY SOFTWARE AND TECHNOLOGY CONFERENCE, 1999, Londres. **Anais...** ACM Press, 1999. p. 176-177.

OPEN Scene Graph. *Site* com informações e acesso à biblioteca gráfica Open Scene Graph. Disponível em: <http://www.openscenegraph.org>. Acesso em: 19 mar. 2002.

OPENGL: The Industry Foundation for High Performance Graphics. *Site* com informações sobre a biblioteca gráfica OpenGL. Disponível em: <http://www.opengl.org>. Acesso em: 19 mar. 2002.

PIMENTEL, K., TEIXEIRA, K. **Virtual Reality:** Through the New Looking Glass. 2. ed. Nova York: Intel/McGraw-Hill, 1995. 438p.

PINTO, A. et al. Modelagem e Animação de Objetos e Cenários para Ambientes Virtuais Colaborativos. In: SYMPOSIUM ON VIRTUAL REALITY, 4., 2001, Florianópolis. **Anais...** São Paulo: SBC, 2001. p. 190-201.

ROBERTSON, G. et al. Immersion in Desktop Virtual Reality. In: ACM Symposium on User Interface Software and Technology, 1997, Alberta. **Anais...** ACM Press, 1997. p. 11-19.

SEMENTILLE, A. C. et. al. Ambientes Virtuais Distribuídos Usando CORBA: Um Estudo De Caso. Workshop Brasileiro de Realidade Virtual, 3., 2000, Gramado. **Anais...** São Paulo: SBC, 2000. p. 145-156.

SENSE8: Virtual Reality 3D Software. *Site* com informações sobre a ferramenta World ToolKit. Disponível em <http://www.sense8.com/>. Acesso em: 19 mar 2002.

SINGHAL, S., ZYDA, M. **Networked Virtual Environments:** Design and Implementation. Nova York: ACM Press, 1999. 331 p.

TANENBAUM, A. S. **Computer Networks.** Upper Saddle River: Prentice Hall PTR, 1996. 813p.

TORI, R., ZOTOVICI, A. Museu Virtual 3D. In: Seminário Internacional NUTAU 2000 Tecnologia & Desenvolvimento, 2000, São Paulo. **CD-ROM NUTAU 2000.** São Paulo, 2000. v. CD-ROM. p. 264-269.

VINCE, J. **Virtual Reality Systems.** Cambridge: ACM Press, 1995. 380p.

WAZLAWICK, R. S. et al. MuseuVirtual: An Authoring Tool for the Collaborative Building of Virtual Museums. In: WORKSHOP BRASILEIRO DE REALIDADE VIRTUAL, 3., 2000, Gramado. **Anais...** São Paulo: SBC, 2000. p. 281-282.

Apêndice A

Neste apêndice encontram-se os documentos gerados durante o projeto da biblioteca.

Casos de Uso

A seguir são apresentados os casos de uso que foram analisados durante o projeto. Por se tratar de uma biblioteca para ambientes virtuais multiusuários, os casos de uso foram desenvolvidos sob o ponto de vista de um ambiente construído com a biblioteca, explicitando-se quando necessário as responsabilidades da biblioteca de classes.

Atores

Desenvolvedor do AV

Este ator representa uma pessoa que está usando a biblioteca para construir um AV específico. Assume-se que o desenvolvedor do AV tenha conhecimentos de programação e da linguagem Java.

Participante

Este ator representa um participante do ambiente virtual, seja, um usuário humano ou um agente autônomo. No caso de participantes humanos, este ator também inclui o software cliente que implementa a interface entre o usuário e o sistema.

Objeto

Este ator representa os demais objetos existentes no AV, incluindo objetos que compõem o cenário e objetos simulados por computador. Este ator é uma generalização dos dois tipos de objetos que podem existir no AV.

Objeto passivo

Este ator representa um objeto do AV que é essencialmente estático. A única interação possível com um objeto passivo é a verificação de colisão com este objeto. Objetos passivos em geral representam elementos de "cenário" do AV.

Objeto ativo

Este ator é definido como uma especialização do ator "Objeto", representando os objetos que possuem algum tipo de comportamento dinâmico. Objetos ativos são passíveis de interação com os participantes do AV e podem também apresentar comportamento autônomo.

Casos de Uso

Criação de um avatar

Atores: Desenvolvedor do AV

Descrição: este caso de uso se inicia quando o desenvolvedor decide construir um avatar para uso em um AV. Um avatar contém uma representação visual, composta

de um ou mais modelos geométricos relacionados em uma estrutura hierárquica. Assim, a primeira coisa que o desenvolvedor deve fazer é criar os modelos geométricos utilizando as ferramentas adequadas, e fornecer ao sistema uma descrição da construção do avatar a partir destes modelos.

Um avatar também pode conter animações que são executadas para representar ações do participante que é representado por ele. Cada animação consiste em uma descrição dos movimentos dos componentes do avatar e de um identificador único. O segundo passo que o desenvolvedor deve realizar é especificar estas animações, se houverem.

Observações: o desenvolvedor pode expandir a funcionalidade fornecida pela biblioteca, alterando a forma de representação do avatar e suas animações.

Criação de um objeto ativo

Atores: Desenvolvedor do AV

Descrição: este caso de uso se inicia quando o desenvolvedor decide construir um objeto ativo para uso em um AV. Um objeto ativo consiste em um modelo hierárquico composto de um ou mais componentes. O desenvolvedor deve construir estes componentes com as ferramentas adequadas, e fornecer ao sistema uma descrição da estrutura hierárquica. O objeto ativo também pode conter animações que são expressas na forma de movimentos dos componentes. Adicionalmente, cada animação deve ter um identificador único.

Por último, um objeto ativo pode ter comportamentos autônomos. O desenvolvedor é responsável por especificar o comportamento desejado, que pode ser um comportamento pré-definido pelo sistema ou uma extensão criada pelo desenvolvedor.

Criação de um objeto passivo

Atores: Desenvolvedor do AV

Descrição: este caso de uso se inicia quando o desenvolvedor decide criar um objeto passivo para uso em um AV. O desenvolvedor do AV constrói um modelo sólido do objeto com uso de software apropriado. Objetos passivos são estáticos, portanto são construídos com apenas um componente. Eles não podem ter animações nem comportamento autônomo. Assim, o desenvolvedor precisa apenas vincular o objeto passivo à sua representação visual.

Criação de um portal

Atores: Desenvolvedor do AV

Descrição: este caso de uso se inicia quando o desenvolvedor decide construir um portal para uso em um AV. Um portal é um objeto que transporta participantes entre AVs diferentes. Um portal é composto basicamente por uma representação visual, tal qual um objeto passivo (isto é, estático e sem animações) e informações de configuração que indicam o destino para onde o usuário é transportado. Assim, o desenvolvedor deve modelar a representação visual do portal com ferramentas apropriadas e fornecer ao sistema as informações de configuração do mesmo.

Criação de um cenário

Atores: Desenvolvedor do AV

Descrição: este caso de uso se inicia quando o desenvolvedor do AV decide construir o cenário que será usado como base para o AV. O desenvolvedor do AV constrói um arquivo de definição de cenário, que descreve aquele cenário a partir da composição de um ou mais objetos passivos e portais. O arquivo de definição pode conter

referências a objetos ativos presentes no cenário, que serão carregados conforme a necessidade.

Construção da interface de usuário

Atores: Desenvolvedor do AV

Descrição: este caso de uso se inicia quando o desenvolvedor do AV constrói a interface de usuário utilizando ferramentas adequadas. Em seguida, esta interface é vinculada ao AV através de classes fornecidas pelo sistema.

Observações: a princípio o sistema não permite a inclusão de elementos da interface dentro do AV, isto é, objetos da interface no espaço virtual tridimensional.

Movimentação do participante

Atores: participante

Descrição: este caso de uso se inicia quando um participante requisita ao sistema movimentação em uma direção. O sistema verifica se o participante irá colidir com algum objeto ou outro participante e em caso negativo, realiza o movimento requisitado.

Observações: o sistema fornece uma escolha de tipos de movimento para o participante (andar, voar), e o tipo de invólucro de colisão a ser usado (cilíndrico, esférico). Estes parâmetros são usados para determinar o comportamento da movimentação.

Animação do participante

Atores: participante

Descrição: este caso de uso se inicia quando o participante ativa uma animação do seu avatar (que foi definida na criação do avatar). O sistema propaga a ativação desta

animação para os demais participantes. A partir deste ponto, a animação é executada independentemente em cada participante.

Comunicação entre dois participantes

Atores: participante

Descrição: este caso de uso se inicia quando um participante envia uma mensagem de comunicação. O sistema pode ter diferentes meios de tratar mensagens de tipos diferentes, por exemplo texto e voz. A mensagem é propagada utilizando protocolos apropriados para o participante desejado.

Observações: assume-se que um participante já foi selecionado como destinatário da mensagem. Não existe garantia de sincronização da mensagem com animações do avatar, para simular fala, por exemplo.

Comunicação entre vários participantes

Atores: participante

Descrição: este caso de uso se inicia quando um participante envia uma mensagem de comunicação. O sistema pode ter diferentes meios de tratar mensagens de tipos diferentes, por exemplo texto e voz. A mensagem é propagada utilizando protocolos apropriados para os demais participantes.

Observações: assume-se que um participante já foi selecionado como destinatário da mensagem. Não existe garantia de sincronização da mensagem com animações do avatar, para simular fala, por exemplo. O sistema pode apresentar um mecanismo de seleção baseado na distância entre os participantes, para não propagar a mensagem a todos os participantes do AV.

Seleção de participante

Atores: participante

Descrição: este caso de uso se inicia quando um participante requisita a seleção de um outro participante do AV. A seleção pode ser feita apontando um participante no campo visual, ou através de um identificador único do participante.

Observações: o sistema oferecerá alguns métodos de seleção que podem ser utilizados.

Seleção de objeto ativo

Atores: participante, objeto ativo

Descrição: este caso de uso se inicia quando o sistema recebe um pedido de seleção de objeto. A seleção do objeto é feita apontando um objeto ativo no campo visual. Se o objeto ativo não está selecionado por nenhum outro participante, ele é selecionado por este participante.

Observações: o sistema oferecerá alguns métodos de seleção que podem ser utilizados. Quando um

Manipulação de objeto ativo

Atores: participante, objeto ativo

Descrição: este caso de uso se inicia quando o sistema recebe pedidos para manipulação (alteração de posição ou orientação) de um objeto ativo. O sistema transmite estes pedidos para o objeto ativo, que deve processá-los.

Observações: o objeto ativo já deve ter sido previamente selecionado.

Acesso a objeto ativo

Atores: participante, objeto ativo

Descrição: através do sistema, o participante requisita acesso a métodos especiais do objeto ativo. Estes métodos são publicados através de uma interface determinada pelo sistema.

Observações: o objeto ativo já deve ter sido previamente selecionado.

Entrada de um participante no AV

Atores: participante

Descrição: este caso de uso se inicia quando o participante entra no AV. O sistema deve fornecer a ele a descrição do cenário e todos os objetos ativos, passivos e avatares que o compõem. Ele deve pedir ao participante uma representação do seu avatar, que será repassada para os demais participantes, juntamente com uma notificação da sua chegada.

Saída de um participante do AV

Atores: participante

Descrição: este caso de uso se inicia quando o participante notifica o sistema de que vai sair da simulação. O sistema propaga esta informação para os demais participantes, que devem deixar de apresentar o avatar daquele participante.

Observações: objetos selecionados pelo participante são liberados quando ele sai.

Passagem de um participante por um portal

Atores: participante

Descrição: este caso de uso se inicia quando o participante ativa um portal, sendo transportado para outro AV. Na prática, este caso de uso corresponde à Saída do Participante do seu AV atual, e sua Entrada em outro AV.

Observações: objetos selecionados pelo participante são liberados quando ele passa pelo portal. O participante não leva consigo nenhum objeto ativo.

Movimentação de objeto ativo

Atores: objeto ativo, participante

Descrição: este caso de uso se inicia quando o sistema requisita a execução das rotinas de comportamento de um objeto ativo. Este objeto deve executar suas rotinas de atualização. Mudanças no seu estado são propagadas para os participantes.

Animação de objeto ativo

Atores: objeto ativo, participante

Descrição: este caso de uso se inicia quando o sistema requisita a execução das rotinas de comportamento de um objeto ativo. Este objeto deve executar suas rotinas de atualização. Mudanças no seu estado são propagadas para os participantes.

Geração de representação visual do AV para um participante

Atores: participante

Descrição: este caso de uso se inicia quando o sistema, periodicamente, gera uma representação visual do estado atual do AV para um participante específico. Esta representação deve ser construída sob o ponto de vista daquele participante e deve ser uma imagem em três dimensões, em perspectiva, do AV.

Observações: participantes que são agentes autônomos podem não necessitar da geração de uma imagem.

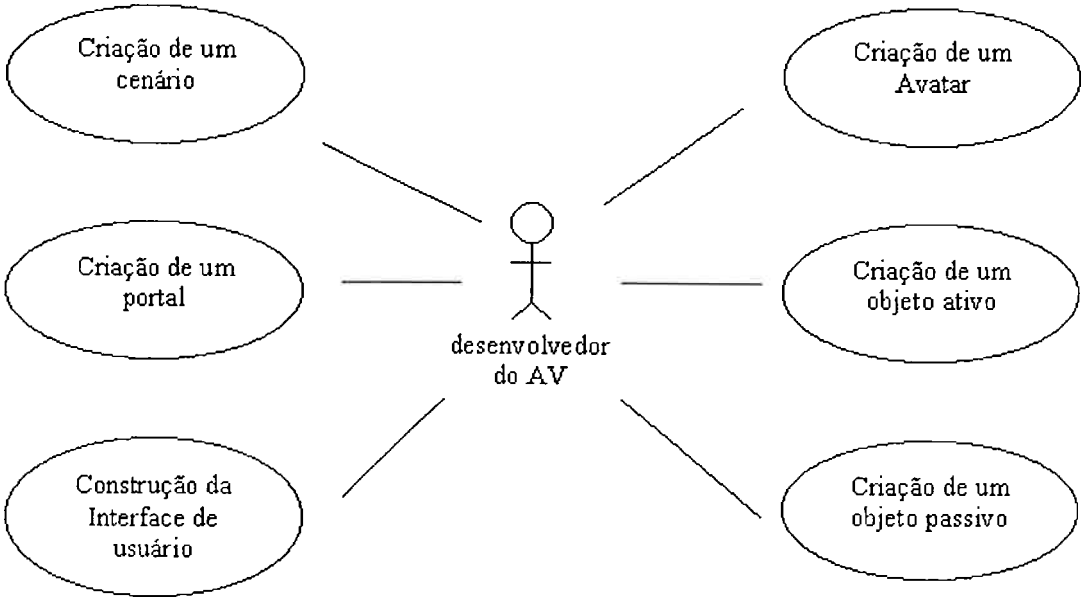
Propagação de alterações na base de dados

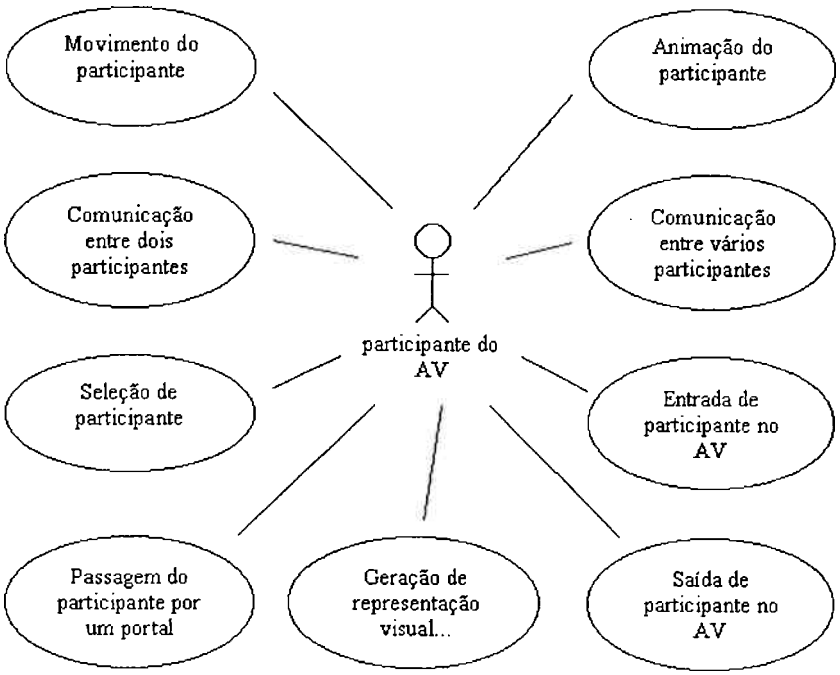
Atores: participante, objeto ativo

Descrição: este caso de uso se inicia quando o sistema, periodicamente, verifica se houveram alterações na base de dados de um participante. Em caso positivo, estas alterações são propagadas para os demais participantes.

Observações: só serão propagadas as alterações relativas a participantes e objetos ativos que são simulados no computador daquele participante.

Diagramas de Casos de Uso




Movimento do participante
Animação do participante
Comunicação entre dois participantes
Comunicação entre vários participantes
participante do AV
Seleção de participante
Entrada de participante no AV
Passagem do participante por um portal
Geração de representação visual...
Saída de participante no AV

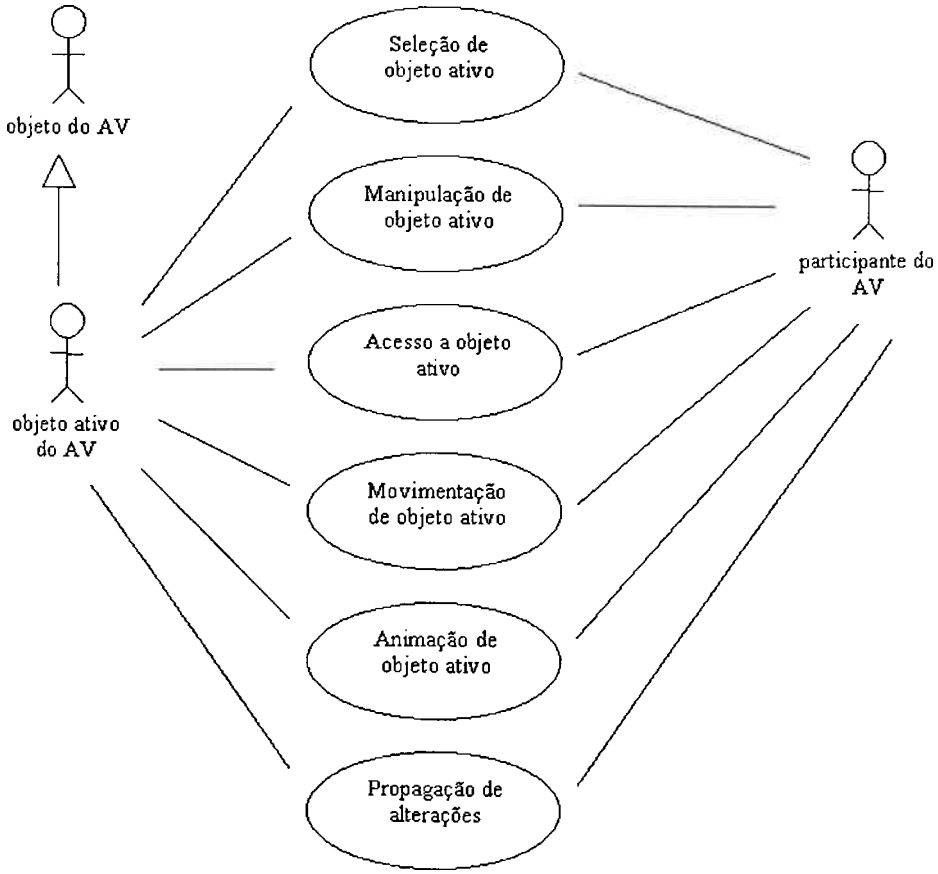
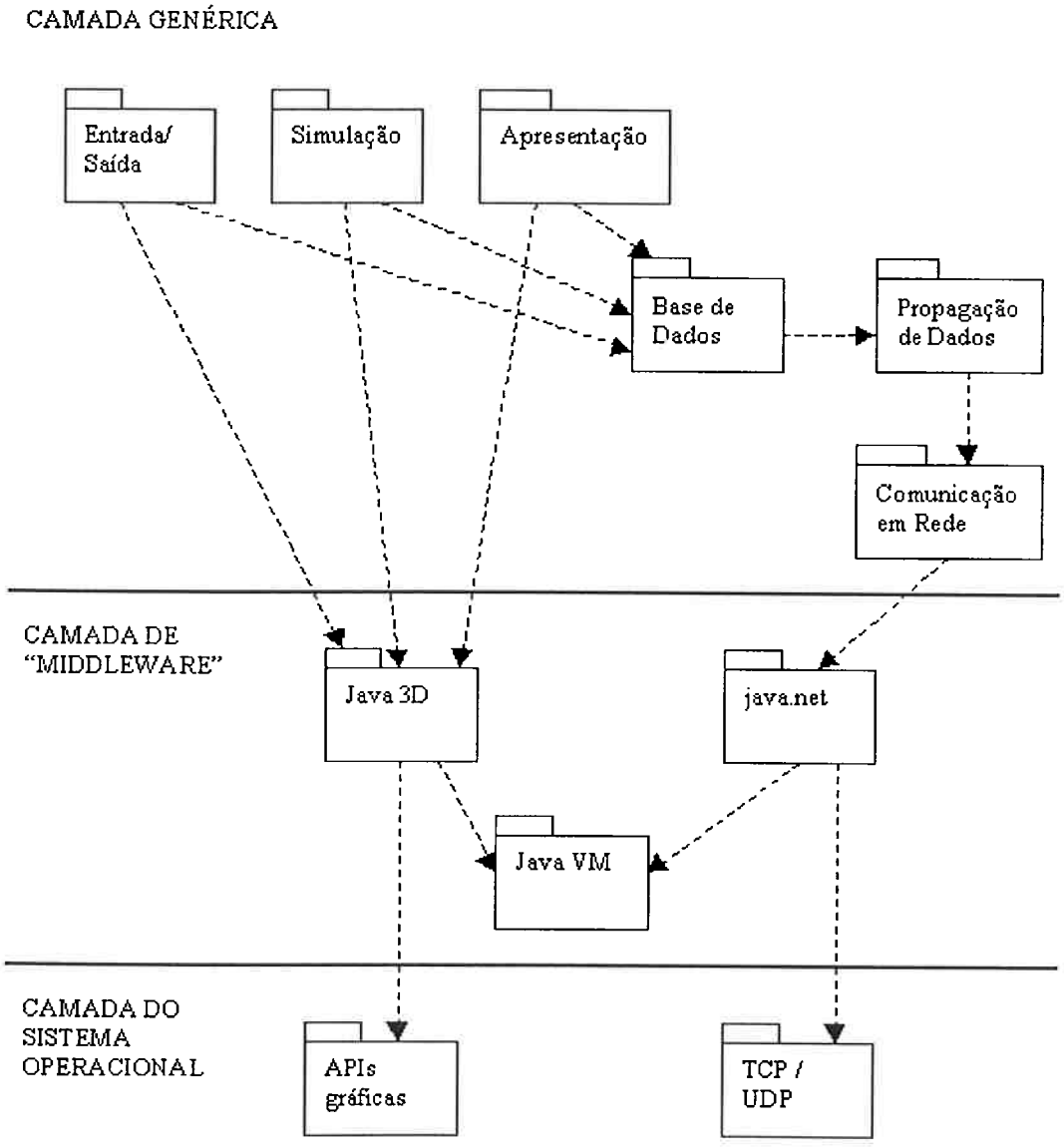

objeto do AV
Seleção de objeto ativo
Manipulação de objeto ativo
participante do AV
Acesso a objeto ativo
objeto ativo do AV
Movimentação de objeto ativo
Animação de objeto ativo
Propagação de alterações

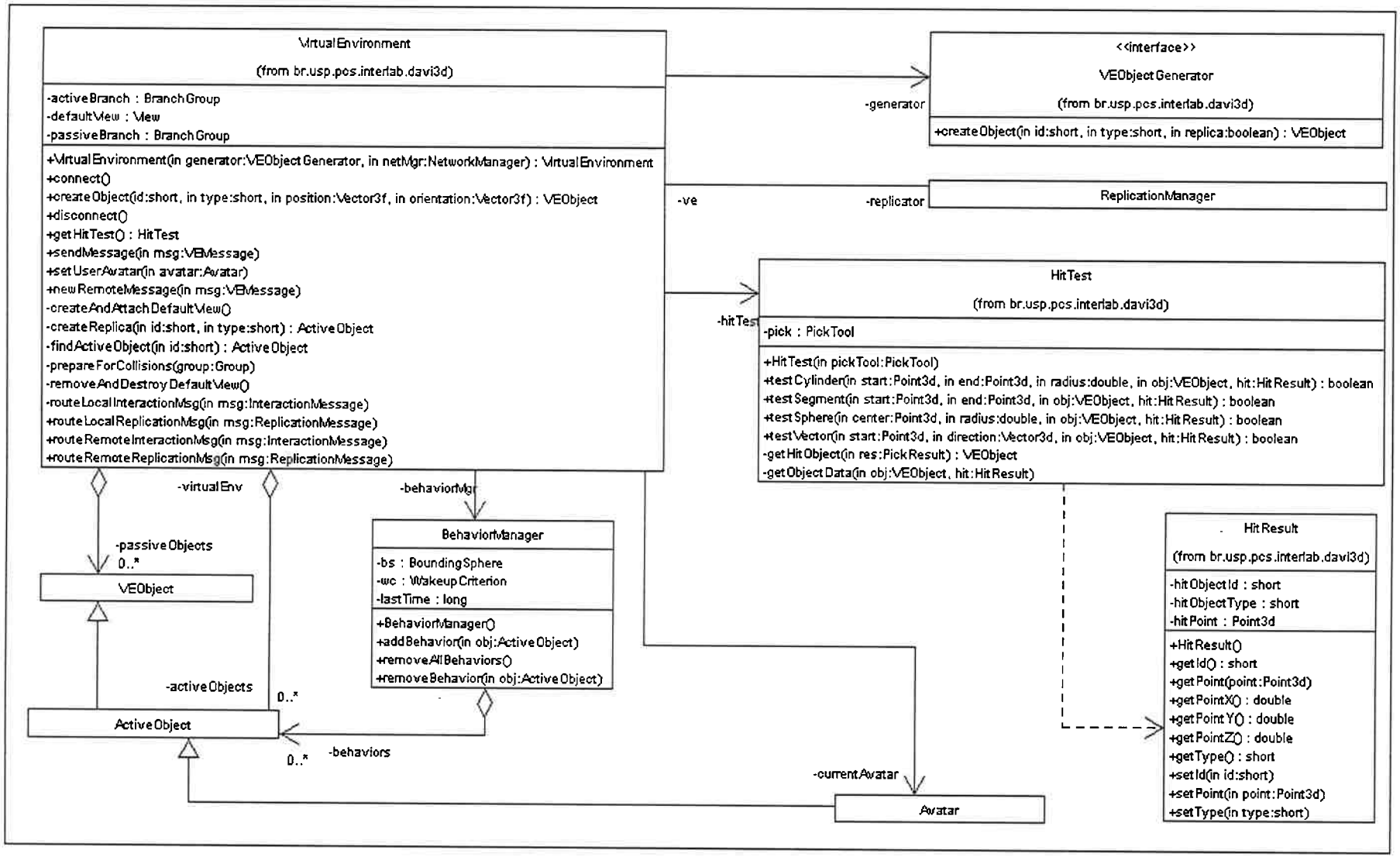
Diagrama de Subsistemas

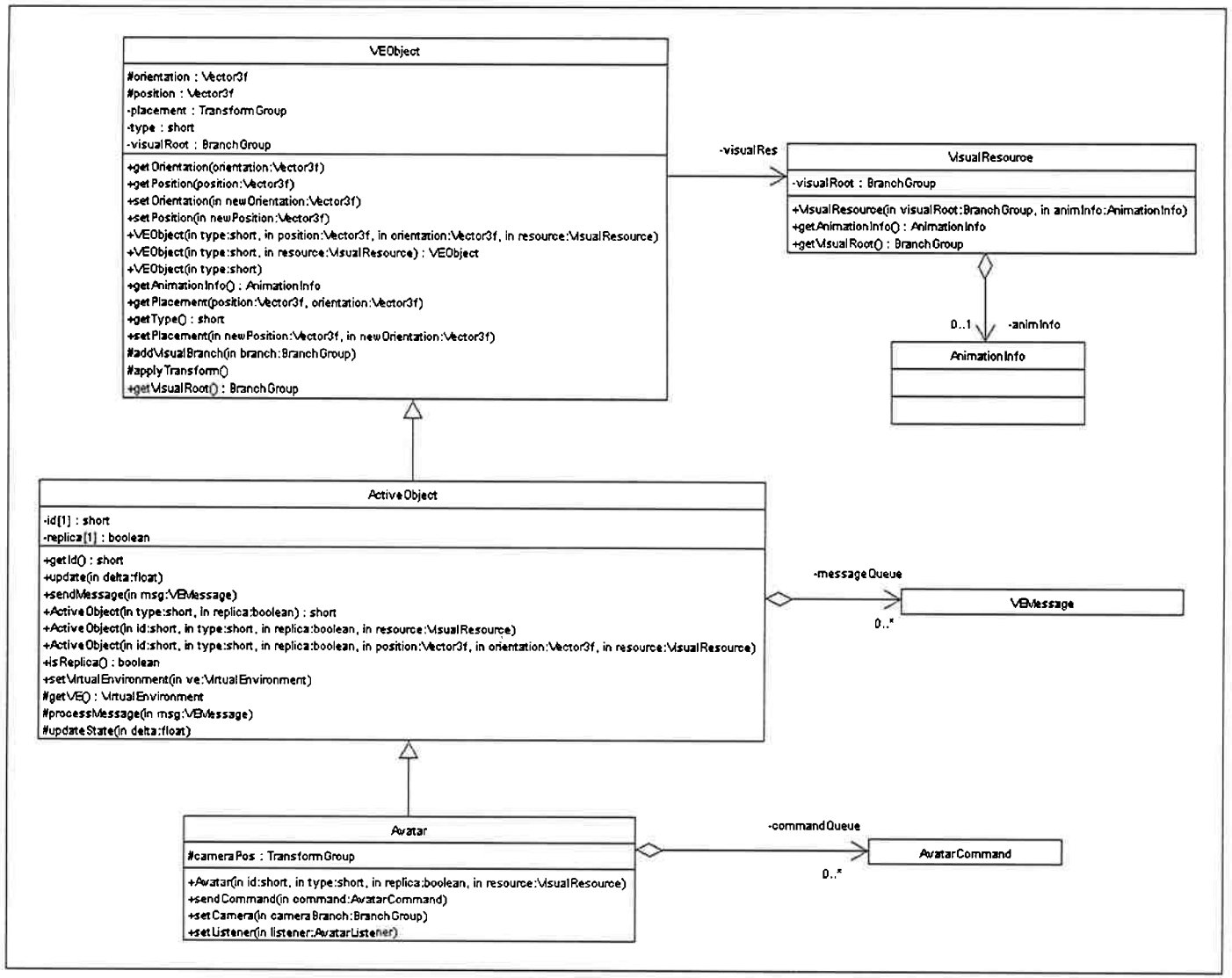
O diagrama abaixo mostra os subsistemas planejados para a biblioteca e o seu relacionamento com as APIs do Java e o sistema operacional.

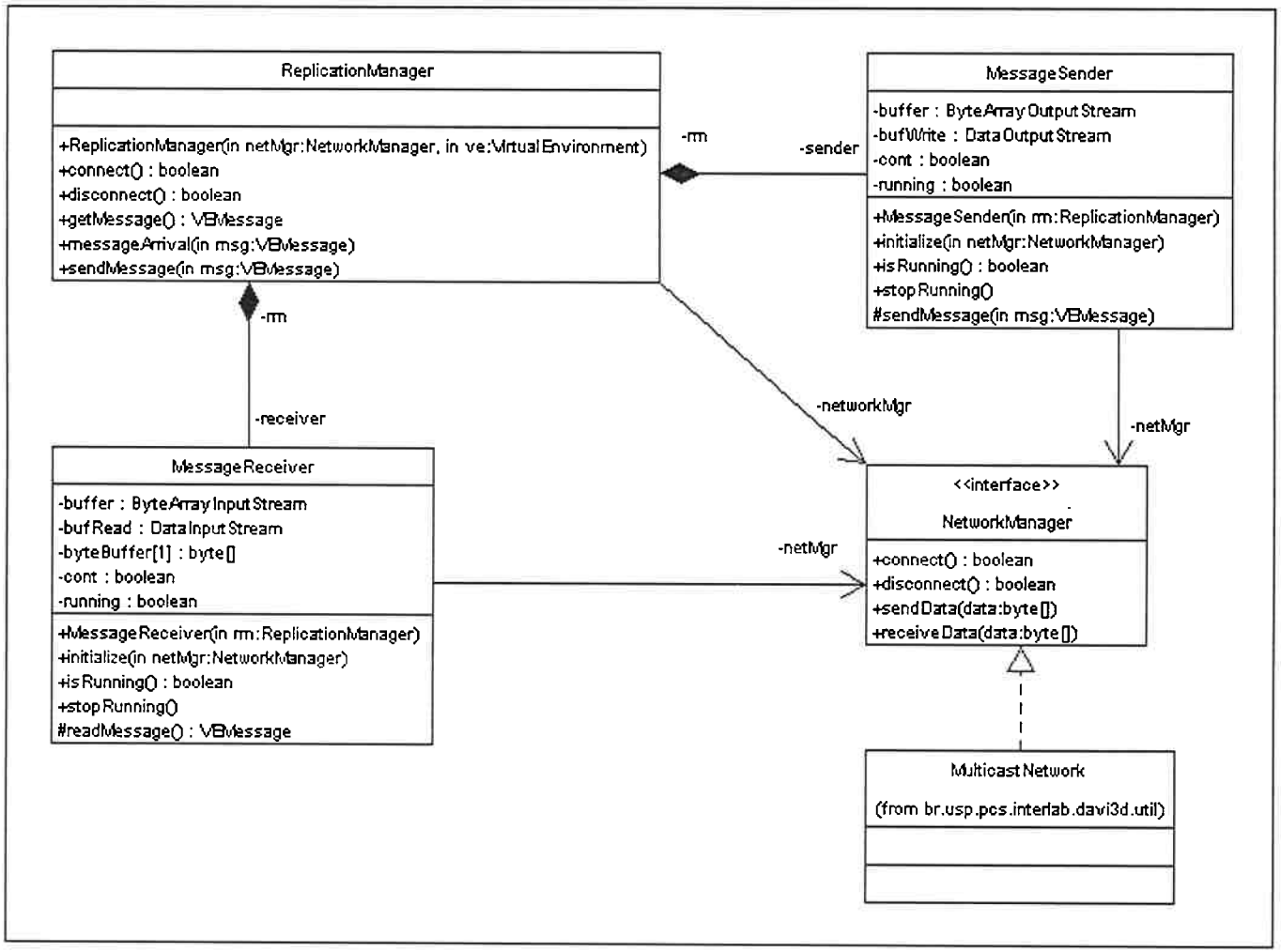


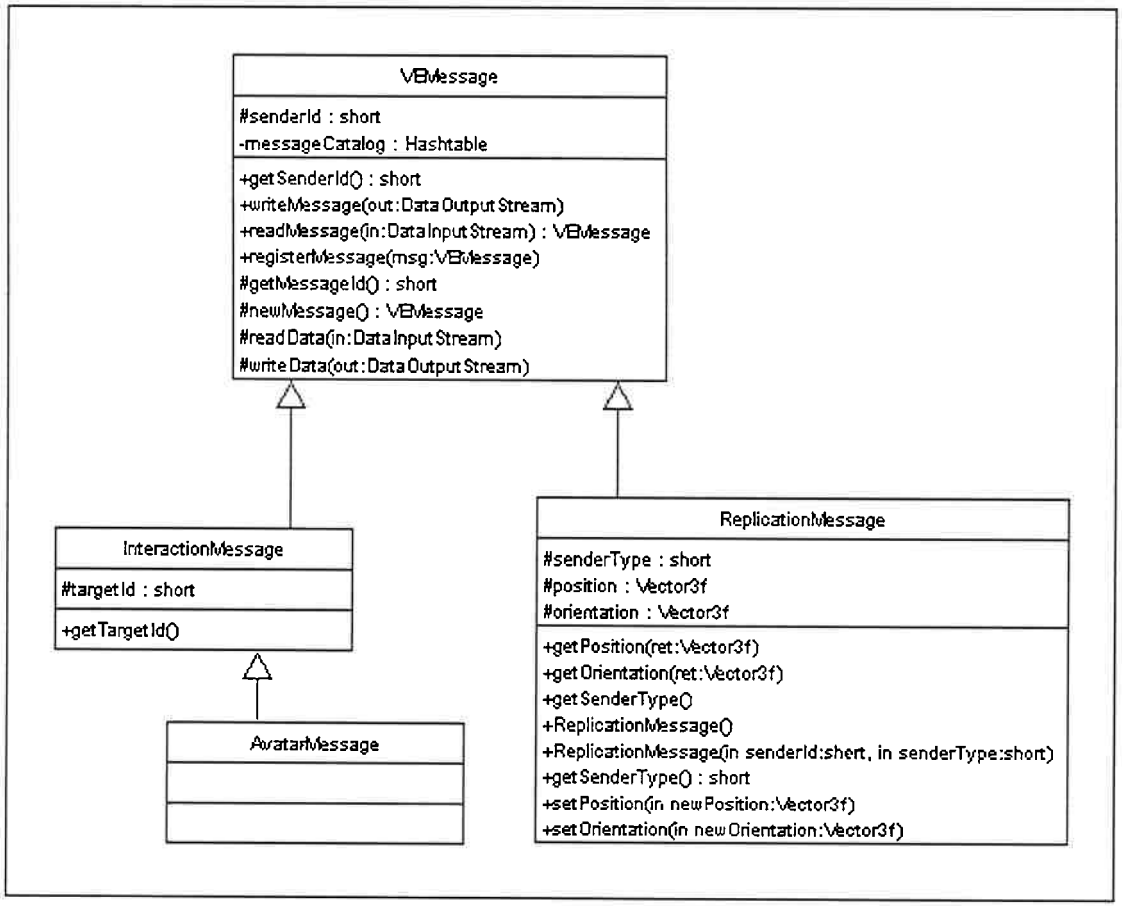
Diagramas de Classes

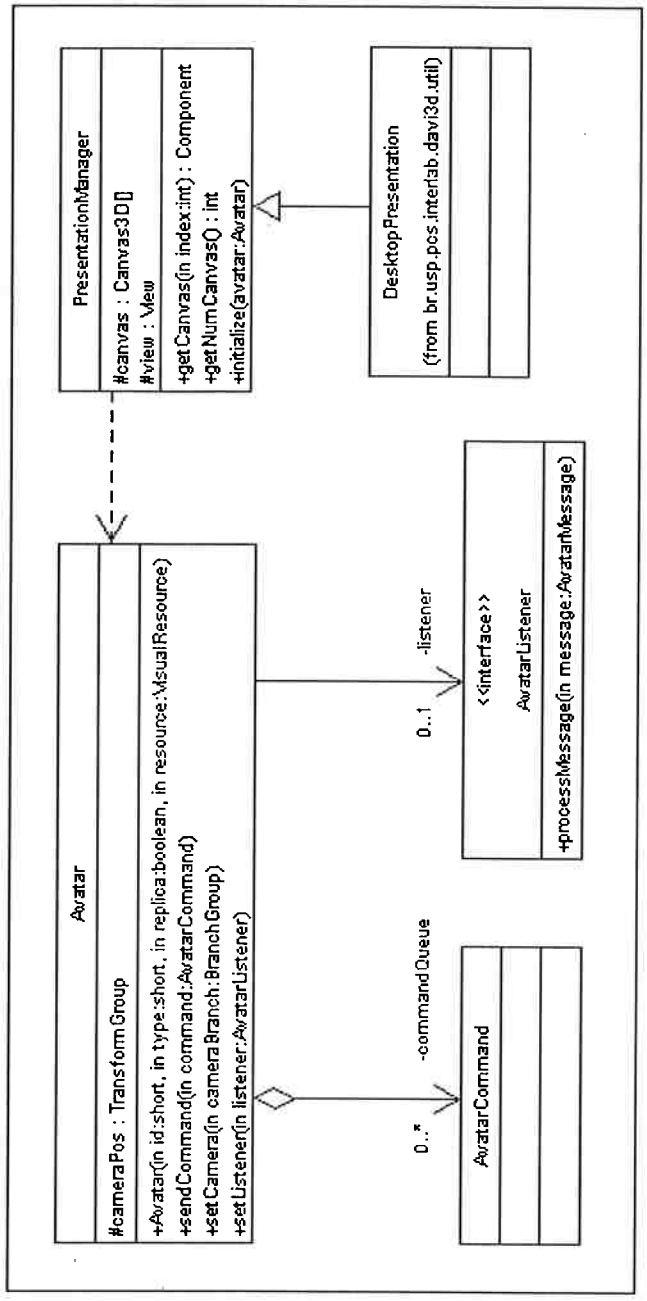
Os diagramas de classes foram apresentados no capítulo 4. Entretanto, eles estão reproduzidos aqui em tamanho ampliado.


VirtualEnvironment
(from br.usp.pos.interlab.davi3d)
-activeBranch : BranchGroup
-defaultView : View
-passiveBranch : BranchGroup
+VirtualEnvironment(in generator:VEObjectGenerator, in netMgr:NetworkManager) : VirtualEnvironment
+connect()
+createObject(id:short, in type:short, in position:Vector3f, in orientation:Vector3f) : VEObject
+disconnect()
+getHitTest() : HitTest
+sendMessage(in msg:VEMessage)
+setUserAvatar(in avatar:Avatar)
+newRemoteMessage(in msg:VEMessage)
-createAndAttachDefaultView()
-createReplica(in id:short, in type:short) : ActiveObject
-findActiveObject(in id:short) : ActiveObject
-prepareForCollisions(group:Group)
-removeAndDestroyDefaultView()
-routeLocalInteractionMsg(in msg:InteractionMessage)
+routeLocalReplicationMsg(in msg:ReplicationMessage)
+routeRemoteInteractionMsg(in msg:InteractionMessage)
+routeRemoteReplicationMsg(in msg:ReplicationMessage)
-generator
<<interface>>
VEObjectGenerator
(from br.usp.pos.interlab.davi3d)
+createObject(in id:short, in type:short, in replica:boolean) : VEObject
-ve
-replicator
ReplicationManager
-hitTest
HitTest
(from br.usp.pos.interlab.davi3d)
-pick : PickTool
+HitTest(in pickTool:PickTool)
+testCylinder(in start:Point3d, in end:Point3d, in radius:double, in obj:VEObject, hit:HitResult) : boolean
+testSegment(in start:Point3d, in end:Point3d, in obj:VEObject, hit:HitResult) : boolean
+testSphere(in center:Point3d, in radius:double, in obj:VEObject, hit:HitResult) : boolean
+testVector(in start:Point3d, in direction:Vector3d, in obj:VEObject, hit:HitResult) : boolean
-getHitObject(in res:PickResult) : VEObject
-getObjectData(in obj:VEObject, hit:HitResult)
-virtualEnv
-behaviorMgr
-passiveObjects
0..*
VEObject
BehaviorManager
-bs : BoundingSphere
-wc : WakeupCriterion
-lastTime : long
+BehaviorManager()
+addBehavior(in obj:ActiveObject)
+removeAllBehaviors()
+removeBehavior(in obj:ActiveObject)
HitResult
(from br.usp.pos.interlab.davi3d)
-hitObjectId : short
-hitObjectType : short
-hitPoint : Point3d
+HitResult()
+getId() : short
+getPoint(point:Point3d)
+getPointX() : double
+getPointY() : double
+getPointZ() : double
+getType() : short
+setId(in id:short)
+setPoint(in point:Point3d)
+setType(in type:short)
-activeObjects
0..*
ActiveObject
0..*
-behaviors
-currentAvatar
Avatar


VEObject
#orientation : Vector3f
#position : Vector3f
-placement : TransformGroup
-type : short
-visualRoot : BranchGroup
+getOrientation(orientation:Vector3f)
+getPosition(position:Vector3f)
+setOrientation(in newOrientation:Vector3f)
+setPosition(in newPosition:Vector3f)
+VEObject(in type:short, in position:Vector3f, in orientation:Vector3f, in resource:VisualResource)
+VEObject(in type:short, in resource:VisualResource) : VEObject
+VEObject(in type:short)
+getAnimationInfo() : AnimationInfo
+getPlacement(position:Vector3f, orientation:Vector3f)
+getType() : short
+setPlacement(in newPosition:Vector3f, in newOrientation:Vector3f)
#addVisualBranch(in branch:BranchGroup)
#applyTransform()
+getVisualRoot() : BranchGroup
-visualRes
VisualResource
-visualRoot : BranchGroup
+VisualResource(in visualRoot:BranchGroup, in animInfo:AnimationInfo)
+getAnimationInfo() : AnimationInfo
+getVisualRoot() : BranchGroup
0..1
-animInfo
AnimationInfo
ActiveObject
-id[1] : short
-replica[1] : boolean
+getId() : short
+update(in delta:float)
+sendMessage(in msg:VEMessage)
+ActiveObject(in type:short, in replica:boolean) : short
+ActiveObject(in id:short, in type:short, in replica:boolean, in resource:VisualResource)
+ActiveObject(in id:short, in type:short, in replica:boolean, in position:Vector3f, in orientation:Vector3f, in resource:VisualResource)
+isReplica() : boolean
+setVirtualEnvironment(in ve:VirtualEnvironment)
#getVE() : VirtualEnvironment
#processMessage(in msg:VEMessage)
#updateState(in delta:float)
-messageQueue
VEMessage
0..*
Avatar
#cameraPos : TransformGroup
+Avatar(in id:short, in type:short, in replica:boolean, in resource:VisualResource)
+sendCommand(in command:AvatarCommand)
+setCamera(in cameraBranch:BranchGroup)
+setListener(in listener:AvatarListener)
-commandQueue
AvatarCommand
0..*


ReplicationManager
+ReplicationManager(in netMgr:NetworkManager, in ve:VirtualEnvironment)
+connect() : boolean
+disconnect() : boolean
+getMessage() : VBMessage
+messageArrival(in msg:VBMessage)
+sendMessage(in msg:VBMessage)
-rm
-sender
MessageSender
-buffer : ByteArrayOutputStream
-bufWrite : DataOutputStream
-cont : boolean
-running : boolean
+MessageSender(in rm:ReplicationManager)
+initialize(in netMgr:NetworkManager)
+isRunning() : boolean
+stopRunning()
#sendMessage(in msg:VBMessage)
-rm
-receiver
-networkMgr
-netMgr
MessageReceiver
-buffer : ByteArrayInputStream
-bufRead : DataInputStream
-byteBuffer[1] : byte[]
-cont : boolean
-running : boolean
+MessageReceiver(in rm:ReplicationManager)
+initialize(in netMgr:NetworkManager)
+isRunning() : boolean
+stopRunning()
#readMessage() : VBMessage
-netMgr
<<interface>>
NetworkManager
+connect() : boolean
+disconnect() : boolean
+sendData(data:byte[])
+receiveData(data:byte[])
MulticastNetwork
(from br.usp.pcs.interlab.davi3d.util)

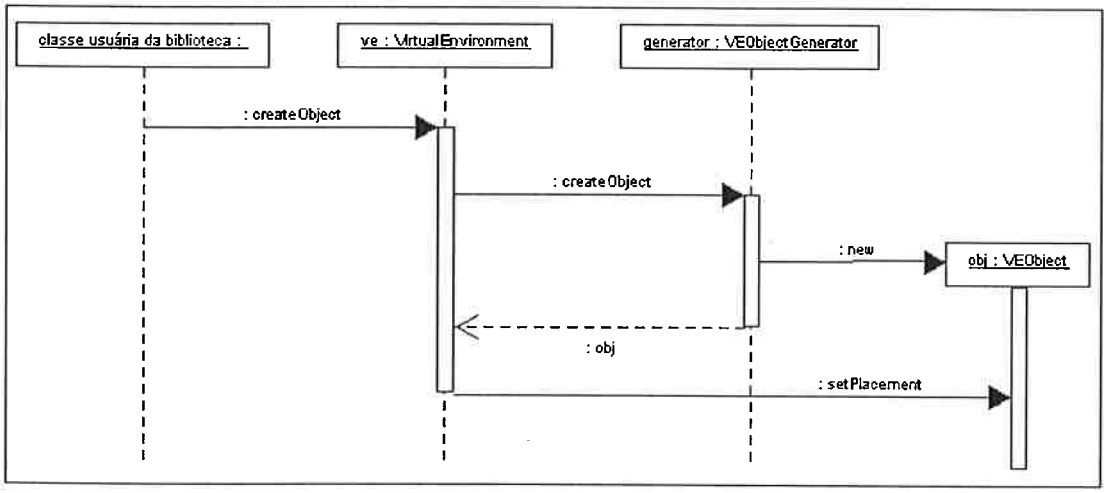

VEMessage
#senderId : short
-messageCatalog : Hashtable
+getSenderId() : short
+writeMessage(out:DataOutputStream)
+readMessage(in:DataInputStream) : VEMessage
+registerMessage(msg:VEMessage)
#getMessageId() : short
#newMessage() : VEMessage
#readData(in:DataInputStream)
#writeData(out:DataOutputStream)
InteractionMessage
#targetId : short
+getTargetId()
AvatarMessage
ReplicationMessage
#senderType : short
#position : Vector3f
#orientation : Vector3f
+getPosition(ret:Vector3f)
+getOrientation(ret:Vector3f)
+getSenderType()
+ReplicationMessage()
+ReplicationMessage(in senderId:short, in senderType:short)
+getSenderType() : short
+setPosition(in newPosition:Vector3f)
+setOrientation(in newOrientation:Vector3f)



Diagramas de Seqüência

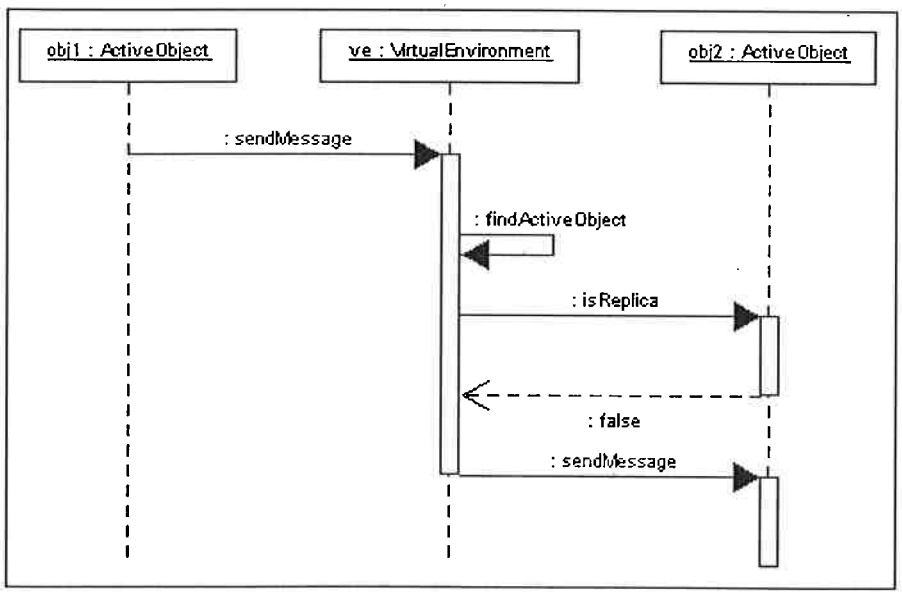
A seguir são apresentados os diagramas de seqüência para as operações mais complexas e importantes envolvendo classes da biblioteca.

Criação de objeto do ambiente virtual



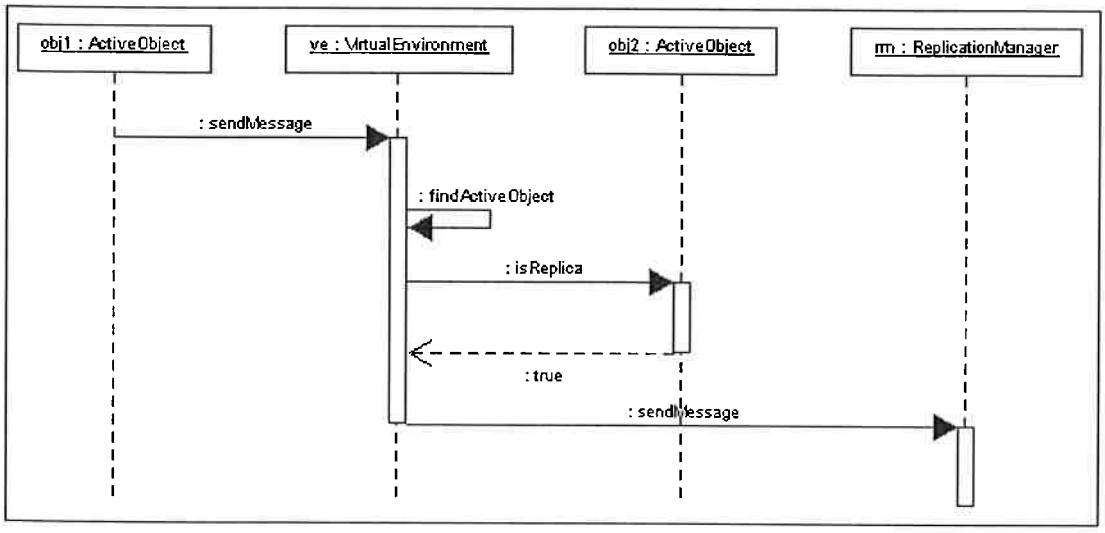
Envio de mensagem – caso 1

Este diagrama representa a troca de mensagens de interação entre dois objetos ativos simulados localmente.



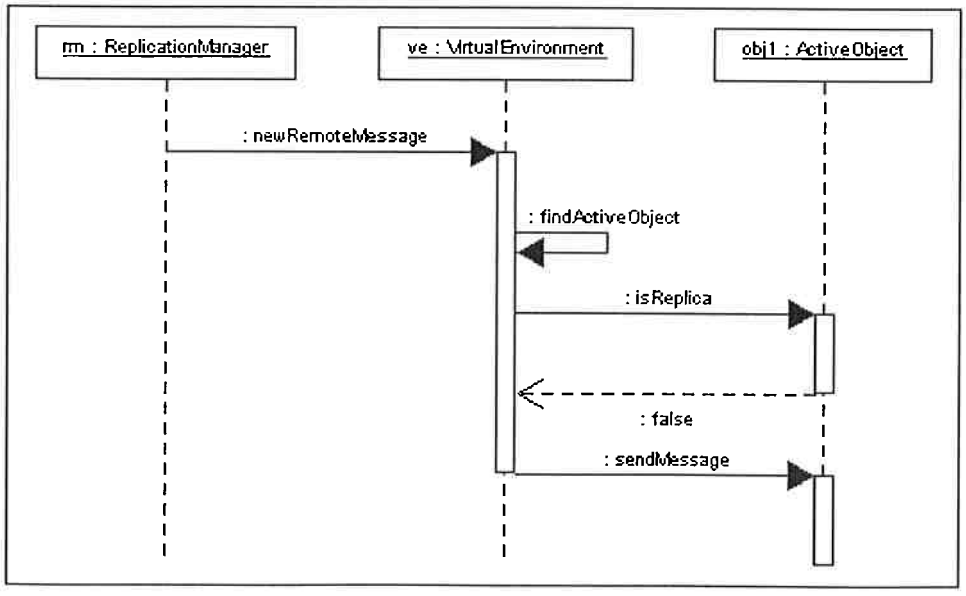
Envio de mensagem – caso 2

Este objeto representa o envio de uma mensagem de interação para um objeto ativo simulado remotamente. O envio de mensagens de replicação ocorre de forma semelhante, mas não é feita a busca pelo objeto localmente.



Recepção de mensagem de interação

Este diagrama apresenta a recepção de uma mensagem de interação no sistema em que o objeto ativo está sendo simulado. O diagrama também é válido para mensagens de replicação em que já existe a réplica local do objeto. Neste caso, o método isReplica retorna *true*.

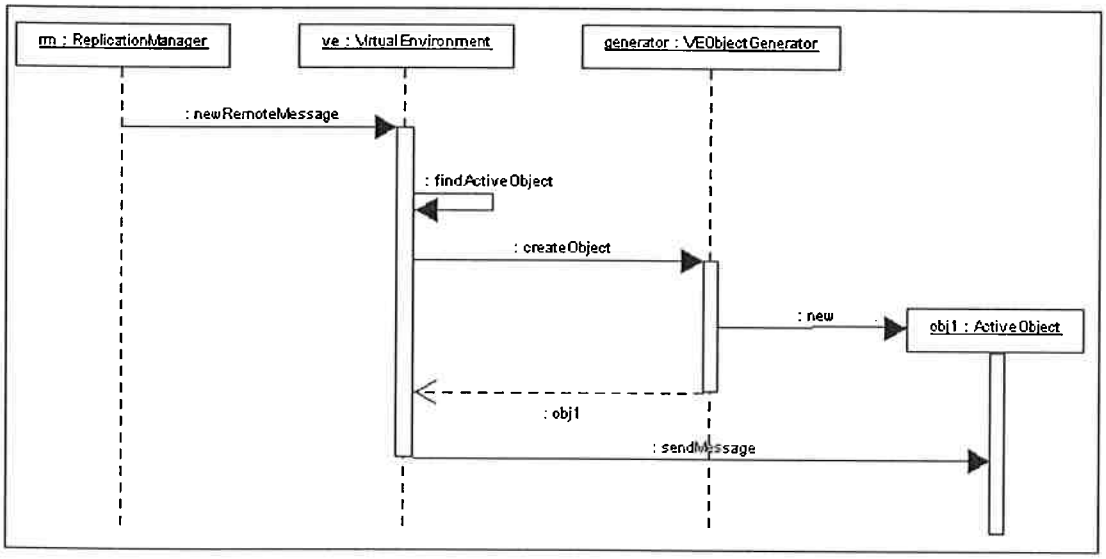


Recepção de mensagem de replicação para um novo objeto

Este diagrama representa o caso em que uma mensagem de replicação chega para um objeto sem réplica local. Este é o mecanismo pelo qual efetivamente os

participantes são notificados da criação de novos objetos ativos no ambiente, incluindo novos avatares.

Neste diagrama, o método findActiveObject retorna *null*, indicando que não existe nenhum objeto com o número de identificação especificado na mensagem de replicação. Neste caso, o VirtualEnvironment cria um novo objeto com as informações da mensagem e a envia imediatamente para que o objeto possa se atualizar com os dados de replicação.



Glossário

Ambiente Virtual (AV): um sistema de *software* construído usando tecnologias de Realidade Virtual, capaz de gerar um espaço tridimensional para uma determinada aplicação.

Ambiente Virtual Compartilhado (AVC): um ambiente virtual que pode ser utilizado por dois ou mais usuários simultaneamente. Normalmente, os usuários não precisam estar fisicamente próximos. Cada usuário tem uma indicação da presença dos demais através do sistema, e pode interagir com eles.

Area of Interest Management (AOIM): Area of Interest Management, ou gerenciamento de área de interesse, são técnicas que buscam reduzir o volume de dados que cada usuário do ambiente virtual precisa receber, com o objetivo de reduzir os requisitos de largura de banda e processamento. Através destas técnicas, cada participante do ambiente virtual determina que informações precisa receber baseado na sua posição e em outros atributos específicos de cada aplicação.

Avatar: Um avatar é uma representação, humanóide ou não, de um usuário dentro do ambiente virtual (Kirner, 2001). Em geral, o avatar é utilizado para permitir que os

usuários possam observar a posição e orientação uns dos outros, aumentando a sensação de presença no ambiente virtual.

Grafo de Cena: o grafo de cena é um modelo para a descrição de "cenas". Uma cena pode ser entendida como um conjunto de objetos geométricos e outros componentes tais como fontes de luz, que formam uma base de dados utilizada para gerar imagens tridimensionais através de técnicas de computação gráfica. O grafo de cena organiza a cena sob a forma de um grafo orientado acíclico, estabelecendo relações hierárquicas entre componentes.

A vantagem do modelo de grafo de cena é que ele abstrai os detalhes de implementação relativos à representação de cada componente da cena, permitindo que o desenvolvedor trabalhe com conceitos mais familiares.

Java3D: O Java3D é uma API criada pela Sun Microsystems em 1998, com o objetivo de permitir a criação de aplicações Java com gráficos tridimensionais. De acordo com Nadeau (1999), o Java3D possui funcionalidade semelhante ao VRML, com o acréscimo de algumas capacidades não encontradas no VRML. O modelo de visualização do Java3D também permite a inclusão de dispositivos de interface especiais com facilidade.

Largura de Banda: Este termo, quando aplicado a redes de computadores, se refere ao máximo volume de dados que podem ser transmitidos por unidade de tempo, através de um canal de comunicação.

Latência: O termo latência se refere ao atraso na resposta do sistema a uma entrada do usuário. Acima de um certo limite, a latência começa a interferir na sensação de imersão e na usabilidade do sistema. A latência pode ser causada por diversas fontes.

A primeira delas é o próprio limite da velocidade da luz; além disso, o tempo de processamento necessário para transmitir dados através da rede (computadores, roteadores etc.) também contribuem para aumentar a latência.

Participante: ao longo do texto, este termo é usado para indicar um usuário humano ou uma simulação autônoma, conectado e interagindo em um AVM.

Realidade Virtual: este termo é usado para designar o campo de pesquisa e desenvolvimento, envolvendo diferentes áreas de conhecimento como a computação gráfica, simulação e sistemas de tempo real com o objetivo de criar ambientes sintéticos tridimensionais. Estes ambientes podem ser reproduções de espaços físicos existentes ou construções baseadas em conjuntos de dados abstratos.

VRML: O VRML 1.0, criado em 1995, começou como uma linguagem para descrição de formas. Posteriormente, o VRML 2.0 redefiniu a linguagem adicionando suporte para som, animação e interatividade (Nadeau, 1999). A versão mais atual é o VRML97, especificado pela norma ISO/IEC 14772-1:1997, que o define da seguinte forma:

> *"The Virtual Reality Modeling Language (VRML) is a file format for describing interactive 3D objects and worlds. VRML is designed to be used on the Internet, intranets, and local client systems. VRML is also intended to be a universal interchange format for integrated 3D graphics and multimedia. VRML may be used in a variety of application areas such as engineering and scientific visualization, multimedia presentations, entertainment and educational titles, web pages, and shared virtual worlds."*