# Compartilhando Ambientes Virtuais pelo Interfaceamento das Linguagens VRML e Java*


Artur Ziviani — `artur@gta.ufrj.br`
Otto Carlos M. B. Duarte — `otto@gta.ufrj.br`

Grupo de Teleinformática e Automação – GTA
COPPE/EE – Programa de Engenharia Elétrica
Universidade Federal do Rio de Janeiro
Caixa Postal 68504 - CEP 21945-970 - Rio de Janeiro - RJ - Brasil



**Resumo**

Ambientes virtuais estão surgindo como uma alternativa de interface homem-máquina aos sistemas multimídias atualmente empregados. Este artigo discute o uso combinado das linguagens VRML (*Virtual Reality Modeling Language*) e Java para a criação de ambientes virtuais que possam ser explorados simultaneamente por mais de um usuário. A aplicação desenvolvida possibilita a um número limitado de usuários compartilhar um mesmo cenário virtual, mantendo a consistência de sua visualização conforme movimentam-se ou trocam mensagens. São aplicadas técnicas de filtragem espacial de dados com o objetivo de minimizar o tráfego de informações de atualização desnecessárias entre os usuários conectados.

**Abstract**

*Distributed environments are raising as an alternative computer-human interface to the multimidia systems nowadays largely employed. This paper discusses the combination of VRML (Virtual Reality Modeling Language) and Java for the creation of virtual environments that could be explored by more than one user simultaneously. The application developed allows that a limited number of users shares the same virtual scene keeping their visualization of others updated as they move or exchange messages. Spacial data filtering tecniques are used to minimize the traffic of irrelevant update information among connected users.*


## 1 Introdução

Nos últimos anos, a indústria de computadores vem consolidando os sistemas multimídias como a interface homem-máquina amplamente disponível. Por outro lado, a


*Este trabalho foi realizado com recursos da UFRJ, FUJB, PROTEM-CC, CNPq, CAPES e COFECUB.

tecnologia de realidade virtual (RV) busca alcançar uma interface entre usuários e seus computadores mais simples, direta e natural.

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

A linguagem VRML [1] (*Virtual Reality Modeling Language*) vem firmando-se como padrão para a descrição de realidade virtual na Internet. VRML contorna suas limitações inerentes ao fato de ser uma linguagem simplesmente descritiva, com a possibilidade de criação de uma interface com outras linguagens de programação. A linguagem Java [2] é amplamente utilizada neste caso, devido ao seu caráter portável, compatível com a proposta de portabilidade de VRML.

Este trabalho, desenvolvido no Grupo de Teleinformática e Automação (GTA) da UFRJ, visa criar um ambiente virtual que possa ser compartilhado de forma a tornar claro o potencial deste tipo de aplicação em áreas como trabalho cooperativo, treinamento, simulação, entretenimento, entre outras. O seu desenvolvimento baseou-se na utilização combinada de VRML e Java, onde VRML ficou responsável pela descrição dos objetos tridimensionais que compõe o ambiente e Java pela interface gráfica com o usuário, comunicação dos clientes com o servidor e implementação do próprio programa servidor.

Este artigo está estruturado da seguinte forma. Na próxima seção serão discutidos os aspectos principais de ambientes virtuais distribuídos. Na Seção 3 será apresentada a linguagem VRML e como esta pode interligar-se com a linguagem Java. Na Seção 4, a aplicação desenvolvida será mostrada. Finalmente, na Seção 5 serão feitas algumas considerações acerca do conteúdo deste artigo.

## 2 Ambientes Virtuais Distribuídos

Vistos como um meio de comunicação, ambientes virtuais [3] apresentam um grande potencial de aplicação em diversas áreas. O principal objetivo de um ambiente virtual é convencer seu usuário que ele está imerso no cenário sintético apresentado. Para alcançar esta ilusão são utilizadas técnicas de realidade virtual.

Um ambiente virtual mostra a representação tridimensional sintética de um espaço real, imaginário ou de uma combinação de ambos, no qual o usuário é capaz tanto de visualizar quanto manipular componentes do ambiente, criando um grau de interatividade com o cenário apresentado. Caso o ambiente suporte o acesso simultâneo por mais de um usuário, permita a interação destes entre si e também com o ambiente apresentado, ele é chamado de ambiente virtual distribuído (*Distributed Virtual Environment* - DVE) [4, 5].

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

Ambientes virtuais distribuídos possuem objetivos de desempenho extremamente ambiciosos, dentre eles o projeto de interfaces naturais em tempo real para ambientes fisicamente corretos [6]. A fidelidade na representação do ambiente virtual não necessita de um realismo fotográfico, porém deve acuradamente exprimir os objetos do mundo real retratados.

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

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

A banda passante limitada para o potencial grande volume de dados propicia outro problema: a latência. Latência pode ser definida como o período de tempo entre a mundança de estado de uma entidade em uma estação da rede e a efetiva alteração de seu estado em uma outra estação. Caso este tempo total seja excessivamente grande, haverá um longo período de espera entre, por exemplo, um usuário instruir seu avatar para mover-se e este realmente deslocar-se à vista dos demais participantes do mundo virtual.

Uma alta latência é prejudicial à qualidade de um sistema DVE, assim como uma latência altamente variável. Atualizações de estado em usuários diferentes com intervalos variáveis de tempo podem gerar problemas de consistência ao mundo virtual.

Os problemas de banda passante limitada e latência são obstáculos para o desenvolvimento de ambientes virtuais distribuídos, principalmente aqueles com exigência de escalabilidade. Logo, um ambiente virtual deve considerar as limitações impostas pela rede de comunicação de forma a ser escalável, possibilitando um grande número de participantes [7, 8].

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

Várias abordagens são propostas para contornar o problema de banda passante limitada e latência alta ou variável [9]. Técnicas como *dead-reckoning* [10] para minimizar o volume de dados ou de filtragem de dados espacial [11] por distância ou oclusão para evitar o envio de dados irrelevantes são exemplos das abordagens comumente utilizadas.

# 3 VRML

Para a implementação do ambiente virtual distribuído proposto neste trabalho utilizou-se a linguagem VRML 2.0 [12] para a descrição do cenário virtual tridimensional de forma combinada com a linguagem Java, a qual é responsável pela interface gráfica com o usuário e pela comunicação de cada cliente com o servidor, também escrito em Java.

## 3.1 Conceitos Básicos de VRML

Toda a estrutura do arquivo VRML é baseada em **nós** (*nodes*). Os objetos descritos em um arquivo VRML formam uma hierarquia de objetos conhecida como **grafo de cena**.

Qualquer ação realizada sobre um determinado nó afeta todos os nós localizados abaixo deste na hierarquia do grafo de cena.

O nó é definido pelo seu tipo e pelos parâmetros que o distinguem de outros nós do mesmo tipo. Estes parâmetros são chamados **campos** (*fields*) e **eventos** (*events*). A especificação de cada nó define o seu tipo, nome e o valor inicial para cada um dos campos. Os campos contêm o estado atual do nó e podem ser de dois tipos: *field* ou *exposedField*. Ambos os tipos definem o valor inicial do estado do nó. No entanto, os campos do tipo *field* não podem ser alterados, sendo considerados privados, enquanto os campos do tipo *exposedField* são considerados públicos e podem ser alterados por outros nós. Os eventos podem ser de dois tipos: de entrada (*eventIn*) ou de saída (*eventOut*). Eventos recebidos são mensagens de outros nós com o intuito de alterar algum estado do nó receptor. Os eventos de entrada e de saída são as interfaces que os nós possuem para comunicar-se com os outros nós descritores do mundo virtual. A conexão entre o nó gerador do evento e o nó receptor do evento é conhecida por **rota** (*route*). Os tipos dos eventos de entrada e de saída devem ser os mesmos. Rotas devem ser estabelecidas de eventos de saída para um ou mais eventos de entrada ou campo exposto (*exposedField*). Uma sucessão de conexões de nós através de rotas passando eventos é conhecida como cascata de eventos.

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

Os principais tipos de nós são descritos a seguir.

- Geométricos - estes devem estar contidos em um nó *Shape* para serem visíveis ao usuário. Neles são definidos, entre outros, os objetos volumétricos primitivos, como esferas, cilindros, cubos ou cones, que manipulados e combinados podem formar objetos mais complexos. Pode-se definir a geometria de um objeto em diversos níveis de detalhamento através do nó LOD (de *level of detail*), para diferentes faixas de distância entre o usuário e o objeto. Tipicamente, cria-se um nível de detalhe alto, médio ou baixo para distâncias baixas, médias ou altas, respectivamente;

- Sensores - detectam as ações do usuário ou a passagem de tempo gerando eventos para outros nós. São fundamentais para o estabelecimento de interatividade com o usuário;

- Interpoladores - estes definem uma função linear discreta $f(t)$, onde $t$ pode assumir valores de $-\infty$ a $+\infty$. Existe um campo chamado *key* que define os $n$ valores de $t$ e outro denominado *keyValue*, com os $n$ valores correspondentes a $f(t)$. Logo, deve haver o mesmo número de elementos em ambos os campos. Considerando-se os extremos no infinito, existem $n + 1$ sub-intervalos. Os valores de $f(t)$ nestes sub-intervalos são obtidos fazendo-se uma interpolação entre os valores definidos. O tipo dos elementos no campo *keyValue* são determinados pelo tipo de interpolador. Estes podem interpolar cores, posições, orientação e escalas, entre outros. Com isto pode-se animar objetos, mudando suas cores, dimensão ou posição;

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

Na Figura 1 está ilustrada uma cascata de eventos que faz um objeto chamado Exemplo girar ao ser tocado pelo usuário. Quando o interpretador percebe o toque do usuário no objeto, ele aciona o sensor de toque associado ao objeto. Este passa um evento, através de uma rota, iniciando a ação de um sensor de tempo. Por sua vez, o sensor de tempo irá acionar um nó interpolador de orientação que fará o objeto Exemplo girar pelas posições definidas neste durante o tempo definido no nó sensor de tempo.

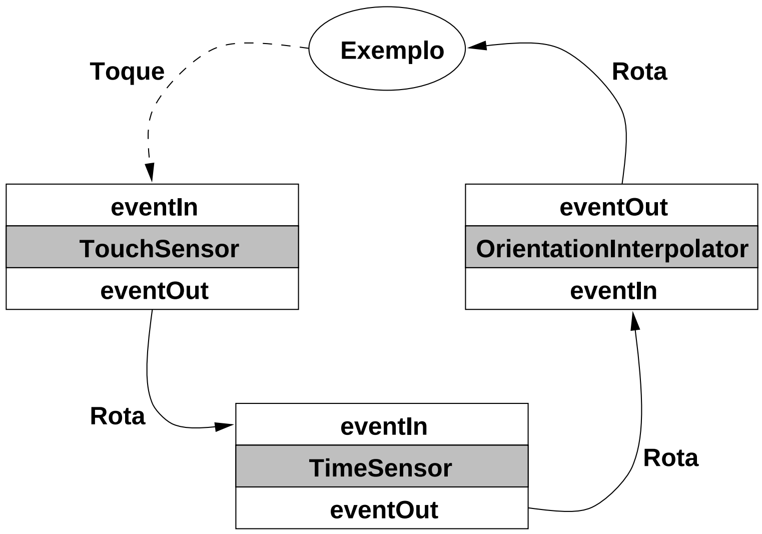


Figura 1: Exemplo de Roteamento de Eventos.

## 3.2 Interface entre VRML e Java

Por ser uma linguagem descritiva, VRML não atende a necessidade de interações mais complexas [13, 14], como as que envolvam cálculos em tempo de execução, decisões lógicas com mais de uma condição a ser verificada ou suporte a comunicação de dados entre usuários visando a construção de ambientes virtuais compartilhados. Pode-se integrar um ambiente virtual descrito em VRML com programas escritos em outra linguagem, sendo Java a mais utilizada, através do estabelecimento de uma interface entre as linguagens pelo uso do nó *Script*.

Quando é criado um nó deste tipo, na realidade, está sendo criada uma interface entre o mundo VRML e o programa *script* associado, usualmente desenvolvido em Java. A sintaxe básica deste nó fornece dois campos que controlam o comportamento do nó *Script* em relação ao mundo virtual: *mustEvaluate* e *directOutput*.

### 3.2.1 O campo *mustEvaluate*

Conceitualmente, o programa *script* associado será executado toda vez que um evento de entrada for recebido pela interface do nó *Script* e então algum processamento será realizado e um evento de saída será fornecido ao mundo virtual descrito pelo arquivo

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

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

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

### 3.2.2 O campo *directOutput*

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

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

## 4 Aplicação

O trabalho desenvolvido visou a implementação de uma interface gráfica com o usuário que permita que este interaja com outros usuários em um ambiente virtual compartilhado. Serão descritos os passos realizados para a descrição do mundo virtual, para a criação da interface gráfica, para o gerenciamento dos diferentes usuários e para a intermediação da troca de informações entre os participantes.

## 4.1 O Mundo Virtual

Foi criado um mundo virtual descrito em VRML, para ser o cenário de atuação dos usuários. Uma visão panorâmica do mundo virtual desenvolvido está apresentada na Figura 2. Há um grande espaço aberto, com um monumento e diversas árvores, e um espaço confinado, uma galeria de arte com representações de quadros clássicos. A Figura 3 ilustra um ângulo de visão interno da galeria de arte.

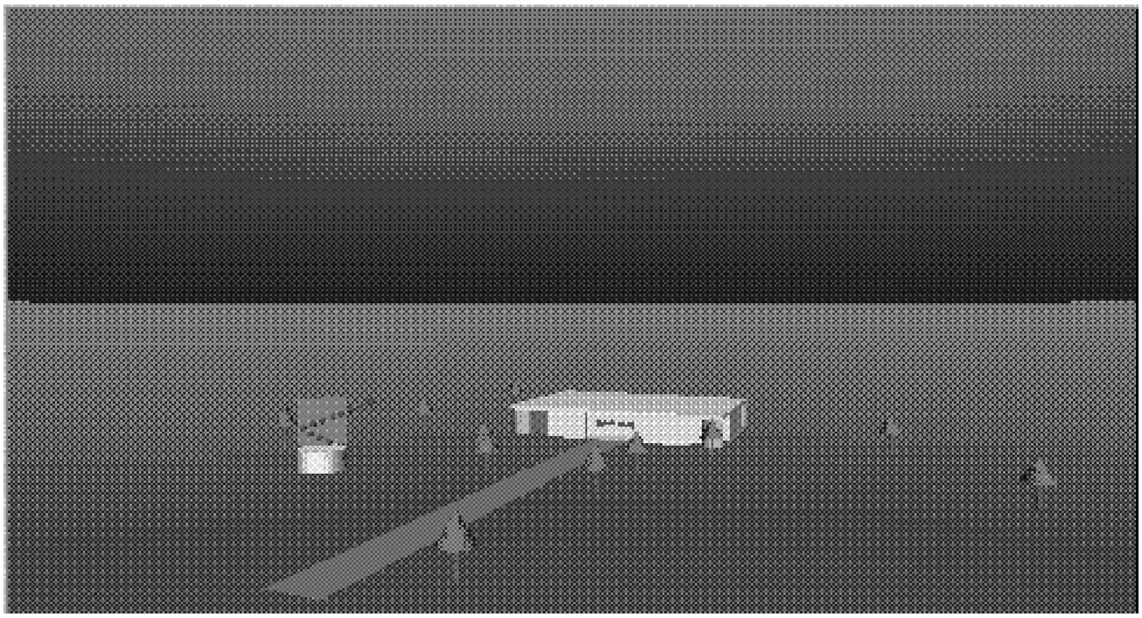

Figura 2: Visão Panorâmica do Mundo Virtual.

A geometria do avatar é definida utilizando-se um nó LOD. Pode-se com o uso deste nó definir níveis de detalhe diferentes a serem utilizados para intervalos de distância diferentes. O avatar é definido em 3 níveis de detalhe. Para o intervalo de 0 a 40 metros, é definido uma versão mais detalhada do avatar, com o corpo inteiro. Para o intervalo de 40 a 80 metros, há uma versão simplificada, sem braços e pernas, e após 80 metros o avatar torna-se invisível, saindo do alcance de visão. Na Figura 4 pode-se ver o avatar em sua versão inteira e simplificada. Assim, quando avatares afastam-se ou aproximam-se uns dos outros, seus usuários vêem-se menos ou mais detalhadamente, respectivamente.

Há um nó sensor de proximidade (*ProximitySensor*) que por sua definição engloba todo a área de interesse do mundo virtual. Um nó *Script* recebe as alterações de posição e orientação do usuário detectadas pelo nó sensor de proximidade e armazena estas alterações em dois campos de sua definição. O campo *mustEvaluate* é mantido em verdadeiro, para que a atualização de movimentação seja imediatamente efetuada. Posteriormente, estes campos serão lidos para o envio das atualizações aos demais usuários.

## 4.2 A Interface Gráfica com o Usuário

A interface gráfica com o usuário é feita por meio de um *applet* Java [15]. A Figura 5 mostra a interface com o usuário. A visão para o mundo virtual é obtida associando-se a um objeto *Panel* o interpretador *Liquid Reality* [16]. Existem os comandos para

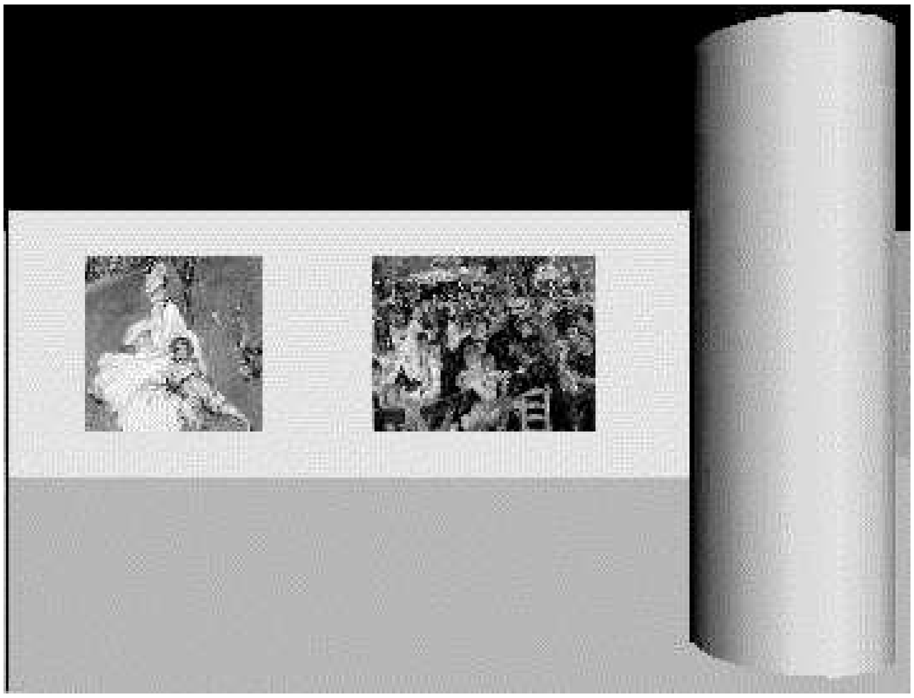

Figura 3: Visão Interna da Galeria de Arte Virtual.

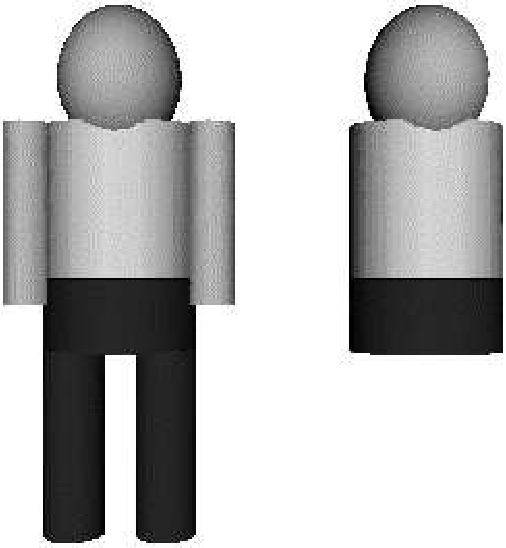

Figura 4: Geometria do Avatar em Dois Níveis de Detalhe.

estabelecimento de conexão com o servidor, desconexão, informção de quais usuários estão conectados e envio de mensagens aos demais usuários.

## 4.3 Filtragens de Dados Utilizadas

Para o espaço aberto existente no mundo virtual é utilizada uma filtragem por distância. Quando um usuário envia uma atualização de sua posição ou orientação ao servidor, este verifica a distância atual entre o usuário que moveu-se e os demais. Caso esta distância seja superior a um determinado valor, o servidor não repassa a informação de atualização ao usuário distante. Considera-se que eles estão fora do alcance de visão um do outro. Na implementação, este limiar ficou estabelecido em 100 metros.

No mundo virtual implementado há um espaço confinado, a galeria de arte. Quando um usuário está no interior da galeria, ele não precisa receber atualizações sobre usuários que estejam no espaço aberto e estes não precisam das informações do usuário atualmente no espaço confinado. Pode-se então implementar uma filtragem por oclusão agindo sobre este espaço confinado.

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

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

## 4.4 Comunicação entre Usuários

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

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

Quando conecta-se ao servidor, o usuário primeiro envia seus dados (nome, posição e orientação) a seu intermediário para que este informe aos outros participantes a sua presença e características. Por sua vez, os usuários que já estavam conectados respondem ao aviso de novo usuário enviando seus dados exclusivamente ao novo usuário. Ao final deste processo todos os usuários, tanto o novo quanto os já conectados, possuem uma visão consistente do mundo virtual. Para cada aviso de novo cliente ou de usuário pré-

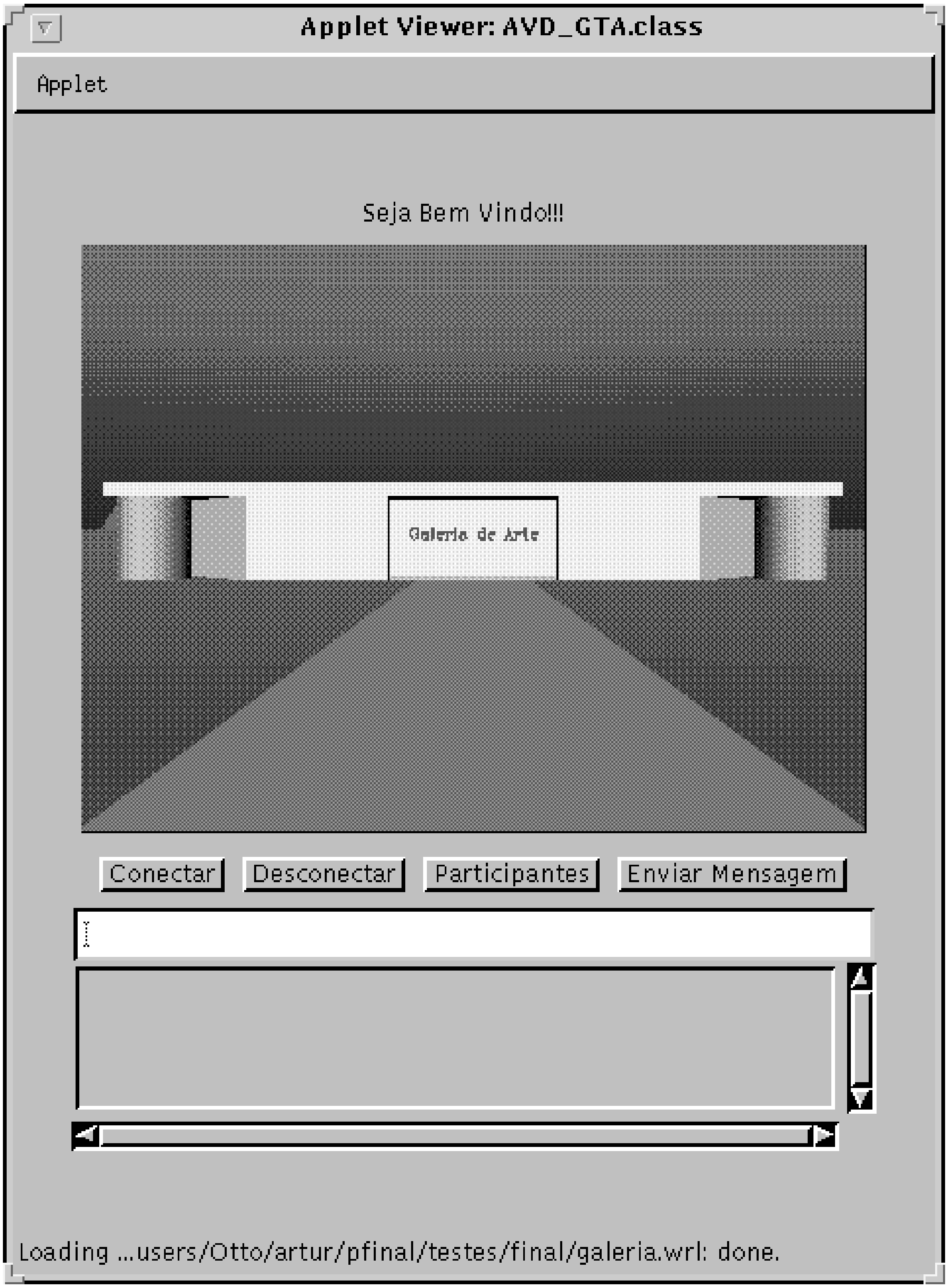


Figura 5: Exemplo da Aplicação.

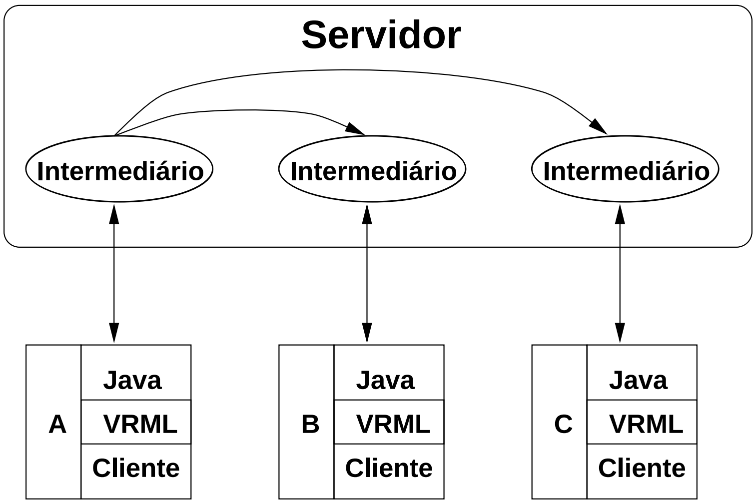


Figura 6: Estrutura de Comunicação.

existente o aplicativo replica a geometria do avatar e a renomeia com o nome do novo usuário. Assim, todas as ações realizadas por um usuário serão refletidas pelo avatar nomeado com seu nome.

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

# 5 Considerações Finais

VRML permite a construção de ambientes virtuais com interatividade alcançada através de nós sensores, interpoladores e roteamento de eventos entre os nós. A combinação de VRML e Java através do nó *Script* permite estender a linguagem VRML à aplicações mais complexas. Possibilita inclusive a criação de novos nós com comportamento determinado pelo programa *script* associado ao nó *Script* definido.

Neste artigo foi apresentada uma aplicação que utiliza esta abordagem para possibilitar a um número limitado de usuários compartilhar um mesmo ambiente virtual simultaneamente. Todos os usuários percebem os movimentos dos outros participantes enquanto estes estiverem em sua área de interesse. Pode-se, portanto, através da aplicação desenvolvida, acessar um ambiente virtual expansível com uma interface amigável que permita a interação entre os diversos usuários conectados.

A aplicação desenvolvida mostrou a realidade virtual como uma técnica de interface homem-máquina. Esta abordagem possui grande potencial em áreas como trabalho cooperativo, em que vários usuários podem compartilhar a visão de um mesmo ambiente modelado em realidade virtual, trocando comentários e idéias sobre o espaço no qual todos estão inseridos.

# Referências


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

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

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

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

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

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

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

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

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

[10] K. Pimentel e B. Blau, "Teaching your system to share", *IEEE Computer Graphics and Applications*, vol. 14, no. 1, pp. 60–65, janeiro 1994.

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

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

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

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

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

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