MATHEUS DEGIOVANI

SISTEMA PARA SINCRONIZAÇÃO DE AMBIENTES VIRTUAIS USANDO X3D

Trabalho de Conclusão de Curso apresentado como exigência parcial, para a obtenção do grau no curso de Ciência da Computação, da Universidade de Franca.

Orientador: Prof. Fabiano Magrin C. Vieira

FRANCA
2005

MATHEUS DEGIOVANI

SISTEMA PARA SINCRONIZAÇÃO DE AMBIENTES VIRTUAIS USANDO X3D

Orientador: _____
Nome: Prof. Fabiano Magrin C. Vieira
Instituição: Universidade de Franca

Examinador(a): _____
Nome: Prof. Dr. Ângelo César Colombini
Instituição: Universidade de Franca

Examinador(a): _____
Nome: Prof. Murilo Táparo
Instituição: Universidade de Franca

Franca, 22 de novembro de 2005

DEDICO *este trabalho aos meus pais, pelo amor sem medida, pela educação e ensinamento que muito me auxiliaram, dando-me base necessária para vencer esta e todas as etapas que estão por vir; aos meus irmãos e familiares, pela amizade e apoio.*

AGRADEÇO: *Ao meu orientador e amigo, Professor Fabiano, que muito me apoiou e auxiliou através de seu profundo conhecimento;*
A Crosby Fitch, pela inspiração proporcionada através de suas palavras;
A todos que direta ou indiretamente contribuíram para a realização deste trabalho.

O cyberespaço. Uma alucinação consensual vivida diariamente por bilhões de operadores autorizados, em todas as nações, por crianças aprendendo altos conceitos matemáticos... Uma representação gráfica de dados abstraídos dos bancos de dados de todos os computadores do sistema humano. Uma complexidade impensável. Linhas de luz abrangendo o não-espaço da mente; nebulosas e constelações infindáveis de dados.

William Gibson

RESUMO

DEGIOVANI, Matheus. *Sistema para Sincronização de Ambientes Virtuais Usando X3D.* 2005. 115f. Trabalho de Conclusão de Curso (Graduação em Ciência da Computação) – Universidade de Franca, Franca.

Este trabalho trata do desenvolvimento de um sistema que permita a sincronização de eventos através de redes de computadores funcionando sobre o protocolo TCP/IP em ambientes virtuais baseados no padrão internacional de arquivos chamado Extensible 3D. Um novo componente para o padrão X3D que possibilite a criação de ambientes virtuais distribuídos foi criado, sendo denominado Sistema de NetNodes. Foi desenvolvido também um conjunto inicial de protocolos que funcionam sobre esse sistema e um browser X3D experimental, chamado GLX3D. Com um teste de usabilidade efetuado sobre uma aplicação de comunidade virtual construída a partir do Sistema de NetNodes, um possível caminho para melhoria das interfaces de usuário de sistemas DVEs através da inclusão de melhores metáforas de interação foi encontrado.

Palavras-Chave: Ambientes Virtuais Distribuídos; Jogos Online Massivos; Jogos Multiusuário; Extensible 3D.

ABSTRACT

DEGIOVANI, Matheus. *Sistema para Sincronização de Ambientes Virtuais Usando X3D.* 2005. 115f. Trabalho de Conclusão de Curso (Graduação em Ciência da Computação) – Universidade de Franca, Franca.

This work deals the development of a system that allows the sincronization of events accross computer networks that work over the TCP/IP protocol stack in virtual environment applications based on the international standard called Extensible 3D. A new component for the X3D standard was created, being named NetNode System. Also developed were an initial set of protocols for this component and an experimental X3D browser, named GLX3D. An usability test done on a sample virtual community based application built over the NetNode system shows a possible way of improving user interfaces on DVEs, through the inclusion of better interaction metaphores.

Keywords: Distributed Virtual Environments, Networked Virtual Environments, MMORPGs, Extensible 3D, VRML

SUMÁRIO

LISTA DE FIGURAS

LISTA DE ABREVIAÇÕES

CAD – Computer Aided Design

CSCW – Computer Supported Collaborative Work

DOM – Document Object Model

DVE – Distributed Virtual Environment

FTP – File Transfer Protocol

HTTP – Hypertext Transfer Protocol

MMORPG – Massively Multiplayer Online Role Playing Game

MUD – Multi User Dungeon

Net-VE – Networked Virtual Environment

RPG – Role Playing Game

SMTP – Simple Mail Transfer Protocol

URL – Uniform Resource Locator

WWW – World Wide Web

INTRODUÇÃO

Com a publicação de Neuromancer (GIBSON, 2003) em 1984, a imagem de uma grande rede tridimensional ligando todos os computadores da Terra em uma simulação de um ambiente virtual atingiu o grande público, dando início às primeiras concepções de como esse mundo virtual paralelo ao real poderia ser criado.

Ambientes Virtuais e Realidade Virtual têm sido matéria de estudo acadêmico há pelo menos duas décadas. O termo Realidade Virtual (RV) é em geral utilizado para denominar um sistema tecnológico, predominantemente de hardware que possibilita a imersão de um usuário em um mundo virtual utilizando óculos de visão estéreo e luvas para interação homem-máquina (STEUER, 1995, p. 5), enquanto um ambiente virtual é mais genericamente definido como uma simulação de um cenário (possivelmente baseado em exemplos do mundo real) executada sobre um sistema computacional (SMED; KAUKORANTA; HAKONEN, 2005).

Um dos campos nascidos com o surgimento das redes computacionais e o aumento do poder de processamento (especialmente gráfico) dos computadores pessoais foi o dos ambientes virtuais conectados em rede, os Net-VEs (de *Networked Virtual Environments*), também conhecidos como *Distributed Virtual Environments*, ou DVEs.

Inicialmente com suporte apenas para texto, e gradualmente ganhando mais recursos como gráficos 2d, 3d e som, os DVEs vêm ganhando grande atenção

tanto da comunidade acadêmica quanto do grande público (à partir da popularização da internet). Através dela, até centenas de milhares (INTERNATIONAL GAME DEVELOPERS ASSOCIATION, 2005 p. 49) de usuários podem acessar, de maneira concorrente, jogos denominados MMORPGs - ambientes virtuais temáticos, onde os jogadores podem personificar seres das mais variadas formas, de cavaleiros medievais à piratas espaciais.

Além da área de entretenimento, DVEs podem ser aplicados em sistemas de Educação a Distância e também para trabalho cooperativo, especialmente em aplicações de design de objetos tridimensionais.

Este trabalho concentra-se na criação de um modelo de comunicação que possibilite a implantação rápida, fácil e barata de ambientes virtuais com os mais variados temas possíveis, baseado no formato de arquivo X3D, que consiste em um padrão internacional para representação de cenários tridimensionais.

1 APLICAÇÕES

Sistemas DVE têm grande aplicabilidade, especialmente no ramo de entretenimento digital. A massificação das redes de computadores (em especial a internet) popularizou um tipo de jogo conhecido como *Massively Multiplayer Online Role Playing Game* (MMORPG). Neles, um grande número de usuários pode interagir, fazendo o papel de um personagem assim como nos tradicionais RPGs (jogos de interpretação de papéis). Este setor chega a movimentar milhões de dólares (INTERNATIONAL GAME DEVELOPERS ASSOCIATION, 2005) anualmente e é um dos grandes focos da indústria de jogos na atualidade. Dois exemplos de MMORPGs são o World of Warcraft (WOW, 2005) e o Lineage II (LINEAGE, 2005).

Além de agregarem até dezenas de milhares de usuários simultâneos, os mais populares MMORPGs são desenvolvidos especificamente com intuito comercial, e portanto, tendem à incorporar as mais modernas tecnologias gráficas e de interface de usuário. Outro importante ponto é que esse tipo de software demanda um grande investimento inicial, não só para o desenvolvimento do código em si, como para a parte artística, royalties de licenciamento de marcas, suporte ao consumidor e equipamentos para conexão com a internet (INTERNATIONAL GAME DEVELOPERS ASSOCIATION, 2005).

Antes da popularização dos MMORPGs com seus cenários tridimensionais sofisticados, os ambientes virtuais multiusuário eram

predominantemente acessados através de interface texto e eram chamados *Multi-User Dungeons* (MUDs). Ainda em uso atualmente, alguns milhares de MUDs (MUDCONNECT, 2005) estão em funcionamento a qualquer momento. Sua grande massificação (em contraste ao número relativamente reduzido de MMORPGs) se deve ao fato da utilização de código pré-fabricado que contém os principais componentes que qualquer jogo desse tipo deve conter, facilitando a criação de novos ambientes até mesmo por usuários sem grandes conhecimentos de programação.

Entre os MMORPGs e os MUDs, situam-se os *webgames* como Meteorus (2005). Webgames podem ser entendidos como jogos acessados diretamente através da *World Wide Web* (normalmente com a utilização de um browser web). Ao contrário dos MUDs, esse tipo de jogo contém alguns gráficos para a criação de interfaces de usuário mais amigáveis, que no entanto, não são tão avançadas quanto aquelas utilizadas em MMORPGs.

Uma característica dos três sistemas (MMORPGs, MUDs e Webgames) é que a inclusão de conteúdo criado pelos usuários é relativamente pequena e segue estritos parâmetros determinados pelos criadores (ONDREJKA, 2004), já que todas as entidades devem aderir à temática do jogo. Por exemplo, em um MMORPG com temática medieval não podem existir elementos arquitetônicos com características modernas, o que poderia quebrar a sensação de imersão dos usuários.

Uma outra aplicação para DVEs relacionada ao setor de entretenimento são as comunidades virtuais. Ao contrário dos jogos, essas comunidades não têm um fim em si mesmas, servindo apenas como ponto de encontro para amigos com interesses semelhantes. Dois exemplos desse tipo de

software são o Active Worlds (ACTIVEWORLDS, 2005) e o Second Life (SECONDLIFE, 2005).

Essas comunidades privilegiam a criação de conteúdo pelos próprios usuários e, em geral, não tem temática ou objetivo definidos formalmente, o que proporciona um menor ciclo de desenvolvimento do software. Além disso, uma característica do Active Worlds é a de que cada mundo é executado como um servidor (programa) diferente, tornando os requisitos de hardware proporcional ao número de usuários. Essa peculiaridade pode ser encontrada na lateral esquerda da Figura 1, onde cada esfera (verde ou vermelha) indica um “mundo” diferente, implementado por pessoas diferentes e com um cenário (e muitas vezes temática) diferente.

Figura 1 – Active Worlds
Fonte: ActiveWorlds (2005)

A aplicação menos explorada para DVEs são os softwares educacionais. Embora a possibilidade de interação tridimensional pareça estimulante, poucos estudos visam a implementação de um sistema de educação à distância sobre um ambiente virtual (BOURAS; FILOPOULOS, 2005).

Outro uso de sistemas DVEs são os softwares de Computer Aided Design (CAD), também conhecidos como Computer Supported Collaborative Work (CSCW). Nesse tipo de software vários usuários podem trabalhar sobre um projeto ao mesmo tempo, facilitando a integração de equipes em pequenas ou longas distâncias (BILLINGHURST; WEGHORST; FURNESS, 2005).

Ainda outra possível aplicação para ambientes virtuais é a criação de lojas ou shoppings virtuais, como feito por Han (2005), onde o usuário pode interagir com os produtos desejados, conversar com outros usuários ou pedir ajuda para atendentes virtuais.

Claramente existe uma grande aplicabilidade para sistemas DVE, sendo as grandes limitações atuais a falta de ferramentas facilmente disponíveis para implementação desse tipo de aplicação e os custos de manutenção dos próprios ambientes (que, na maioria dos casos, é proporcional ao número de usuários).

2 HISTÓRICO

O objetivo deste capítulo é trazer uma revisão de alguns dos principais DVEs construídos ao longo das últimas décadas separados arbitrariamente em quatro categorias. Nessas seções, certos detalhes sobre as tecnologias que caracterizam cada projeto serão omitidos, sendo mostrados no capítulo três, uma vez que grande parte dessas tecnologias é compartilhada por diversos softwares diferentes.

Em primeiro lugar, serão estudados os DVEs de origem militar, uma vez que mesmo as redes de computadores com grande número de pontos, como a ARPANET, surgiram neste ambiente (WIKIPEDIA, 2005).

Em seguida, os principais DVEs desenvolvidos em meios acadêmicos serão analisados. Essa é a categoria mais bem documentada e que comporta maior quantidade de idéias originais, uma vez que esses softwares não precisam alcançar objetivos comerciais. Por essa mesma razão, grande parte dos sistemas acadêmicos não obteve a mesma popularidade dos comerciais.

O próximo conjunto de DVEs analisados serão os MMORPGs. Embora este tipo de aplicativo contenha um grande número de usuários, detalhes sobre as implementações estão menos disponíveis que nos tipos anteriores, por razões de estratégia mercadológica.

Finalmente, as principais características dos jogos online acessados através de browser, os webgames, serão analisadas.

2.1 DVES MILITARES

Assim como a internet surgiu dentro de um contexto militar, também as primeiras simulações online de grande escala foram patrocinadas pelo Departamento de Defesa dos Estados Unidos.

O projeto SIMNET (MARSHALL et al., 2005) foi iniciado em 1983, finalizado em 1990 e visava a simulação de um ambiente virtual com alta fidelidade e baixo custo, tendo ênfase em cenários militares. Essa simulação utilizava uma arquitetura de objeto-evento, significando que objetos do sistema geravam eventos para outros objetos. Ele também utilizava a técnica de dead-reckoning para suavizar a movimentação das entidades do sistema e reduzir o gasto de largura de banda.

Uma evolução do projeto SIMNET foi o estabelecimento do padrão Distributed Interactive Simulation (DIS). Sua principal contribuição para o estudo de DVEs foi a criação de uma coleção de mensagens padrão, denominadas Protocol Data Units (PDUs) e a especificação das características de desempenho mínimas que os usuários devem obedecer para participar da simulação, como por exemplo a latência máxima permitida ou a largura de banda mínima necessária para comunicação (HARDT; WHITE, 2005).

O padrão DIS utiliza conexões ponto-a-ponto através de grupos multicast para a transmissão de PDUs e com suas características consegue obter

uma média de envio de 250 bytes por segundo por participante. Em uma de suas maiores batalhas simuladas, 5400 entidades participaram de um combate dentro de um mesmo ambiente virtual (HARDT; WHITE, 2005).

Construída a partir da observação da plataforma DIS, um dos trabalhos mais novos dentro da área de DVEs militares é a especificação da High Level Architecture (HLA), criada em 1996 (DAHMANN; FUJIMOTO; WEATHERLY, 2005). Esta arquitetura é mais generalizada que a DIS, sendo seus dois principais objetivos o reaproveitamento e a interoperabilidade entre simulações.

Na arquitetura HLA, cada simulação individual é chamada de federado, e deve oferecer um conjunto de funcionalidades predefinido, de forma que vários federados possam ser combinados em uma federação. Uma federação, portanto, pode combinar diversos ambientes virtuais para um exercício militar altamente complexo.

2.2 DVES ACADÊMICOS

A área acadêmica também teve grandes avanços dentro do campo dos DVEs. Um dos grupos com maior experiência no desenvolvimento desse tipo de aplicação é o Grupo de Pesquisas NPSNET, cujo trabalho mais novo, o NPSNET-V (KAPOLKA; MCGREGOR; CAPPS, 2005; BRUTZMAN et al., 2005) introduz um framework construído com uma arquitetura de microkernel para extensibilidade do sistema, utilizando para isso a linguagem Java.

A principal motivação para a criação da arquitetura NPSNET-V foi prover uma plataforma que permitisse extensibilidade dinâmica do ambiente virtual, sem que ele precisasse ser desligado. Para esse fim, cada funcionalidade de um DVE (como protocolos de rede, algoritmos de oclusão e renderizadores do ambiente) foi implementada seguindo uma arquitetura unificada, altamente modular. Cada funcionalidade existe independentemente de outras, sendo uma aplicação construída como uma federação desses módulos, montada dinamicamente durante a execução da aplicação (KAPOLKA; MCGREGOR; CAPPS, 2005).

A única peça realmente fundamental do sistema NPSNET-V é o microkernel invariável, responsável por descobrir os módulos existentes na máquina do usuário, quais os necessários para a execução de um determinado ambiente virtual, como obtê-los e como combiná-los para execução.

Outro sistema que também foi muito importante no estabelecimento formal do campo acadêmico dos DVEs foi o sistema SPLINE desenvolvido no Mistsubishi Electric Research Laboratories (BARRUS; WATERS; ANDERSON, 2005). A grande contribuição desse sistema foi a definição do conceito de locales como unidades geográficas de ambientes virtuais, que podem ser unidas criando a sensação de um mundo contínuo.

O sistema SPLINE funciona como uma Application Programming Interface (API) a partir da qual outros programadores podem fazer uso para a construção de seus próprios ambientes virtuais. Ele utiliza comunicação multicast quando possível ou unicast quando necessário (por exemplo, através da internet). Não existem servidores centrais em um ambiente virtual do SPLINE, onde cada criador de um locale ou de um objeto é o responsável por sua manutenção

(BARRUS; WATERS; ANDERSON, 2005). A Figura 2 mostra uma imagem de um ambiente virtual construído sobre o sistema SPLINE, denominado Diamond Park.

Figura 2 – Diamond Park, uma aplicação criada sobre o sistema SPLINE
Fonte: MERL, 2005.

Também classificado como acadêmico é o sistema MASSIVE-3 (SNOWDON; GREENHALGH; PURBRICK, 2005), que utiliza um banco de dados distribuído para armazenamento de informações e transmite seus dados via algoritmos de multicast lógico, utilizando uma implementação cliente-servidor para possibilitar seu uso por usuários domésticos conectados à internet.

O sistema MASSIVE-3 foi fortemente influenciado pelo sistema SPLINE e também utiliza o conceito de locales para divisão de um mundo virtual. No entanto, esse conceito é estendido com a definição de aspectos (aspects) diferentes para cada locale individual.

Uma outra contribuição desse sistema foi a generalização do conceito de políticas de seleção, que são definidas pelo usuário ou pelo próprio ambiente virtual e que são utilizadas para determinar quantos e quais locales exibir ao usuário em um dado momento (SNOWDON; GREENHALGH; PURBRICK, 2005).

Já os sistemas de Knutsson (2005) e Chapman (2005) utilizam uma rede de overlay ponto-a-ponto chamada Pastry sobre a internet, visando criar um ambiente virtual altamente escalável. Nesses sistemas, cada participante tem um

endereço ou identificador específico dentro da rede P2P lógica, chamado NodeID. Cada entidade do mundo virtual é armazenada no nó que apresenta o NodeID mais próximo de um hash calculado sobre a área (ou no sistema de Chapman, no “Voxel”) onde esta entidade está localizada.

Uma vantagem dessas arquiteturas é que a complexidade máxima do mundo virtual é diretamente proporcional à quantidade de usuários e de seu poder de processamento. Além disso, embora a rede seja teoricamente ponto-a-ponto é possível simular uma arquitetura cliente-servidor fazendo com que as áreas tenham exatamente o tamanho de todo o ambiente virtual, porém com a vantagem de replicação do conteúdo nos nós vizinhos ao servidor (CHAPMAN, 2005, p. 21).

2.3 MMORPGS

Os antecessores dos atuais MMORPGs foram os MUDs que contém apenas interface de texto. Nos MUDs a interação é feita exclusivamente através de comandos enviados pelo jogador ou por acontecimentos gerenciados pelo servidor (CHURCHILL; BLY, 2005).

Por ser uma interface texto e estar limitado, basicamente, pela velocidade de interação do usuário, seu gasto de largura de banda é relativamente baixo. Além disso, MUDs simulam ambientes geográficos (como continentes ou países) ou arquitetônicos (como cidades ou casas) e portanto utilizam a separação entre seções desse ambiente (por exemplo, os cômodos de uma casa) para diminuir

o gasto de largura de banda nas comunicações feitas entre jogadores, em essência utilizando essa característica como um algoritmo de oclusão.

A grande maioria dos MUDs existentes atualmente deriva seu código fonte de bases de código (codebases) pré-existentes como o CircleMud (2005). Isso permite um ciclo de desenvolvimento e implantação de novos ambientes virtuais de maneira rápida por pessoas que não tenham muita familiaridade com programação de sistemas de rede.

Embora seu uso seja majoritariamente em aplicações de entretenimento, Churchill e Bly (2005) mostram resultados sobre possíveis usos de MUDs em um ambiente de trabalho. O primeiro sistema MUD foi criado por Richard Bartle e Roy Trubshaw na Universidade de Essex, no Reino Unido (HISTORYOFMUDS, 2005) em 1979.

Na maior parte dos MUDs, apenas os coordenadores (muitas vezes conhecidos como “deuses”) podem modificar o ambiente de maneira arbitrária, embora jogadores com mais experiência sejam muitas vezes chamados para integrar essa equipe, e desenvolverem eles mesmos certas áreas do jogo.

O primeiro jogo que pode ser classificado como online massivo foi o Meridian 59, desenvolvido em 1996 (MERIDIAN59, 2005). Embora não tenha sido tão bem sucedido quanto seus sucessores, chegando à “meros” 12.000 assinantes (KENT, 2005), demonstrou a viabilidade de DVEs que comportam um grande número de participantes ao mesmo tempo.

Este jogo continha gráficos popularizados pela série Doom da ID Software, também conhecidos como gráficos 2.5D por não serem realmente tridimensionais. Nas palavras de Rich Vogel, um de seus criadores: “Nós fizemos muitas coisas que ninguém havia feito antes. [...] Nós lideramos o caminho e muita

gente nos seguiu" (KENT, 2005, p. 2).Com um grande esquema de marketing e uma forte licença dentro dos círculos dos RPGs de computador, o Ultima Online (UO, 2005) foi lançado em 1997 através da distribuidora Electronic Arts, sendo um absoluto sucesso e atraindo em seu apogeu cerca de 200.000 assinantes. A Figura 3 mostra dois jogadores combatendo um dragão, dentro do cenário deste jogo.

Figura 3 – Ultima Online
Fonte: UO, 2005

Este foi realmente o primeiro grande sucesso dos RPGs massivos, abrindo caminho para toda uma nova geração de jogos online. Com gráficos isométricos ao invés de tridimensionais, foi um dos primeiros ambientes virtuais a sofrer extensivamente com problemas de vandalismo e desempenho do sistema (KENT, 2005, p. 3).

Em 1998 houve o lançamento do Lineage (LINEAGE, 2005), um MMORPG que valorizava o aspecto de interações sociais de grandes grupos (também conhecidos como tribos), alcançando a marca de quatro milhões de assinantes, com a maior parte de sua base de usuários localizada na Ásia (KENT, 2005, p. 3).

O Lineage também é um jogo com perspectiva isométrica, porém com grandes diferenças em relação aos MMORPGs ocidentais, que valorizam a evolução individual de personagens.

Em 1999 houve o lançamento de Everquest (EVERQUEST, 2005), o primeiro MMORPG completamente tridimensional a alcançar grande popularidade, atingindo cerca de 500.000 assinantes.

Ao contrário de Ultima Online, Everquest concentrava a ação dos personagens em batalhas e desenvolvimento pessoal e utilizava um sistema econômico e social menos ambicioso (KENT, 2005, p. 4).

A partir de 2003, uma nova safra de MMORPGs começou a atingir o mercado (ASHER, 2005).

Utilizando uma das mais reconhecidas marcas internacionalmente, o Star Wars: Galaxies (SWGALAXIES, 2005) conseguiu cerca de 275.000 assinantes em seus primeiros meses após o lançamento. Uma grande novidade foi o estabelecimento de classes não combatentes dentro do jogo, que possibilita sua utilização por jogadores que não estão interessados em simples simulações de combate (ASHER, 2005).

Já Lineage II (LINEAGE2, 2005) seguiu os passos de seu sucessor, porém com novos gráficos tridimensionais, chegando a marca de 1.5 milhão de assinantes ao fim de 2004 (WIKIPEDIAc, 2005).

Os jogos mais recentemente lançados dentro da categoria de MMORPGs são Everquest II em 2004 (EVERQUEST2, 2005) e World of Warcraft (WOW, 2005) também lançado em 2004, sendo um dos jogos mais vendidos nas primeiras 24 horas de seu lançamento, com 240.000 unidades (WIKIPEDIAb, 2005).

Os MMORPGs citados até aqui foram todos desenvolvidos com código fechado, e portanto detalhes sobre sua implementação são, no mínimo, escassos. Uma exceção dentro dessa categoria é o Planeshift (PLANESHIFT, 2005) que é desenvolvido como um projeto de código aberto por uma grande comunidade e não têm objetivos comerciais.

Uma outra característica desses grandes MMORPGs é a quantidade de material criado pelos seus usuários. Ainda mais restritos que os MUDs, esses jogos contém mínimas capacidades de inclusão de modelos pelos jogadores, tornando a criação de novo material completamente dependente dos próprios desenvolvedores.

2.4 WEBGAMES

Webgames podem ser classificados como jogos que são executados dentro de um browser da *World Wide Web*. No entanto, apenas aqueles que permitem interação entre os usuários (ou seja, são jogos multiplayer) estão dentro do escopo deste trabalho.

Jogos como Meteorus (2005) podem ter um ciclo de desenvolvimento mais reduzido, uma vez que seus gráficos são simples e eles utilizam tecnologias amplamente disseminadas, como páginas estáticas HTML ou tecnologias de páginas dinâmicas como PHP.

Uma característica marcante de sua implantação é que rodam sobre servidores *web* comuns e, portanto podem ser gerenciados por qualquer pessoa que

tenha conhecimento desse tipo de servidor. Além disso, esse tipo de serviço de hospedagem tem custos reduzidos e alta disponibilidade. Por exemplo, o servidor de presença BlueHosting (2005) oferece serviços de hospedagem de páginas web à partir de R$7,00.

3 ESTADO DA ARTE

O conhecimento acumulado de duas décadas de estudo militar, acadêmico e comercial dos DVEs permite a delineação das principais tecnologias que caracterizam esse tipo de software. Este capítulo revisa os conceitos mais importantes dentro do contexto do *framework* desenvolvido neste trabalho.

A seção 3.1 irá apresentar uma pequena introdução sobre as redes de computadores (em particular aquelas ligadas pela pilha de protocolos TCP/IP) e os assuntos primordiais em sua programação. Tanenbaum (1997), que é tido como leitura obrigatória em cursos de Ciência da Computação, é a principal fonte de informações dessa seção.

Na seção 3.2 serão discutidos as possíveis arquiteturas de comunicação para aplicações que funcionam em redes, com especial atenção aos DVEs.

A seção 3.3 traz uma análise dos métodos de divisão espacial encontrados em sistemas de ambiente virtual.

A seção 3.4 discute problemas e soluções relacionados a escalabilidade de sistemas DVE.

Finalmente, a seção 3.5 apresenta algumas questões relacionadas a segurança dos ambientes virtuais executados em rede.

3.1 INTRODUÇÃO À PROGRAMAÇÃO DE REDES

Até o início da década de 80, redes de computadores não eram muito mais do que uma curiosidade acadêmica ainda insipiente. Duas décadas mais tarde, a computação de rede tornou-se não só popular como indispensável para uma parcela crescente da população humana, seja para atividades profissionais ou pessoais. Com o surgimento da internet, um novo meio de comunicação global modificou completamente a maneira como as interações humanas são realizadas.

Pode-se definir uma rede de computadores como um conjunto de unidades de entrada, saída ou processamento de dados, comumente referidos como “nós” ou “*hosts*”, que podem trocar informações sem um intermediador humano. Dessa maneira, dois computadores ligados através de um cabo coaxial podem ser considerados uma rede, porém vinte computadores trocando informações através de disquetes manuseados por um operador humano, não.

Toda troca de informações dentro de uma rede de computadores, esteja ela ligada por fios de cobre, microondas ou satélites, deve ser feita seguindo um conjunto pré-estabelecido de comandos e estruturas de dados chamado de protocolo. Um protocolo de comunicação pode definir, por exemplo, qual a quantidade máxima de informações que dois nós da rede podem transmitir em um determinado momento ou como eles são identificados.

Para reduzir a complexidade de implementação dos sistemas de rede, atualmente utiliza-se uma arquitetura denominada pilha de protocolos, onde cada item da pilha corresponde a um protocolo (ou “camada”) com responsabilidades diferentes. Em cada nó da rede, as camadas se comunicam logicamente enviando

informações para a camada inferior, até a última camada que é chamada de física, responsável pela efetiva transmissão de dados pelo meio de suporte físico da rede.

Existem vários modelos de referência que implementam pilhas de protocolos, sendo os mais conhecidos o ISO/OSI e o TCP/IP. O modelo OSI foi desenvolvido como um padrão internacional e, portanto é muito bem conhecido e estudado. Já o modelo TCP/IP surgiu da necessidade de interconexão de redes díspares, e se tornou o padrão dentro da internet, sendo amplamente utilizado no mundo da informática.

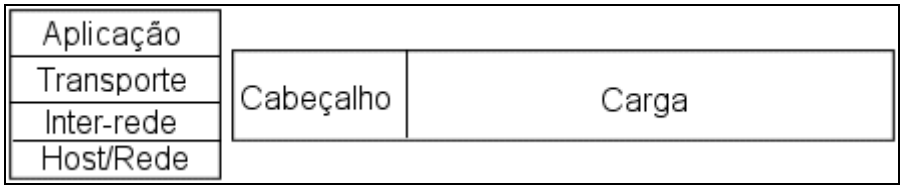

Figura 4 – Camadas do protocolo TCP/IP e modelo abstrato de um pacote

Ambos modelos funcionam através do envio de pequenas mensagens, também conhecidas como pacotes, que contém um determinado número de *bytes*. Cada pacote tem (basicamente) um cabeçalho onde estão presentes informações sobre o remetente e o destinatário da mensagem e tamanho dos dados, e uma área de carga (*payload*) que realmente carrega as informações desejadas. A Figura 4 mostra as camadas da pilha de protocolos TCP/IP e o modelo básico dos pacotes de informação trocados entre *hosts*.

A pilha de protocolos TCP/IP é composta de quatro camadas, denominadas Camada de Aplicação, de Transporte, de Inter-redes e Host/Rede.

A Camada de Aplicação é a mais alta do modelo TCP/IP e corresponde aos programas que utilizam a rede para transferência de informações. Alguns dos protocolos mais populares dessa camada são HTTP (para navegação na *World Wide Web*), FTP (para troca de arquivos) e SMTP (para envio de correio eletrônico).

A Camada de Transporte oferece dois protocolos de comunicação fim a fim para que aplicações de rede possam ser desenvolvidas. O protocolo *Transmission Control Protocol* (TCP) é orientado à conexão e confiável, enquanto o *User Datagram Protocol* (UDP) utiliza datagramas e não é confiável. Mais informações sobre esses protocolos serão dadas na seção 3.2, onde seu impacto em sistemas DVE será melhor analisado.

A Camada de Inter-redes define o protocolo IP e é a responsável pela comutação e roteamento dos pacotes através da rede de computadores subjacente. Esse protocolo trabalha de forma que um pacote chegue do remetente ao destinatário através de pequenos saltos (*hops*) entre roteadores individuais.

Finalmente, no modelo TCP/IP a camada Host/Rede é a responsável por enviar informações entre dois *hosts* ligados diretamente através de um meio físico. O protocolo usado aqui depende da configuração do nó e da rede a qual ele pertence e varia de máquina a máquina. Um dos protocolos mais utilizados (dentro do escopo de redes locais) para esse fim é o protocolo Ethernet (especificação IEEE 802.3).

A programação de aplicativos de rede através da pilha TCP/IP envolve a criação de um novo protocolo da Camada de Aplicação, que utilize algum dos (ou todos os) protocolos da Camada de Transporte (TCP e UDP).

As estruturas lógicas mais utilizadas para a programação desse tipo de aplicativo são os *sockets*. Um *socket* é um ponto final de comunicação entre dois *hosts* e pode ser do tipo *stream* (sendo, portanto um *socket* TCP) ou *datagram* (sendo um *socket* UDP).

Um *socket* pode ser do tipo cliente (onde ele é usado para enviar informações e pode apenas iniciar comunicações) ou do tipo servidor (quando aceitar conexões de outros nós da rede).

Para que uma comunicação possa concretizar-se, um aplicativo precisa especificar dois parâmetros relacionados à localização de um nó dentro da inter-rede: endereço e porta.

O endereço (quando discutindo o modelo TCP/IP utiliza-se um endereço IP) de um nó é um número que o identifica unicamente dentro da rede. Esse endereço pode ser obtido através de entidades oficiais reguladoras da internet ou, no caso de usuários domésticos, através do provedor de acesso que estabelece um endereço para o usuário no momento de sua conexão. Em redes locais também é possível utilizar uma faixa de endereços comum que por definição não é visível dentro da internet.

Endereços IP podem ter 32 bits de comprimento (IPv4) ou mais recentemente 128 bits (IPv6). Endereços IPv4 normalmente são expressos na forma *Dot-Decimal Notation* (por exemplo, 200.160.42.55), enquanto endereços IPv6 têm uma forma mais complexa, expressa em hexadecimal (por exemplo, 2001:0db8:85a3:08d3:1319:8a2e:0370:7334).

Utilizar apenas o endereço para localização de um *host* pode provocar problemas no momento em que uma mensagem chega a esse nó, pois a pilha de protocolos precisa decidir para qual aplicação enviar os dados. Para suportar multiplexação de aplicações em um mesmo endereço, utiliza-se o conceito de portas como um número que identifica um canal de comunicação dentro de um nó. Toda mensagem TCP ou UDP carrega um endereço de porta (de 16 bits). Quando essa

mensagem chega ao seu destino, ela é passada para o *socket* ligado à porta especificada, completando a transmissão.

3.2 ARQUITETURAS DE COMUNICAÇÃO

A arquitetura de comunicação entre os usuários de um ambiente virtual é um dos principais fatores que influenciam em seu desempenho. Os dois modelos mais conhecidos para troca de mensagens são o Cliente-Servidor (C/S) e o Ponto-a-Ponto (P2P).

A arquitetura cliente-servidor se baseia em uma interação bidirecional entre cada usuário do sistema (chamado cliente) e um único outro nó da rede lógica (o servidor). Nessa arquitetura, toda mensagem de um cliente é enviada para o servidor que processa a mensagem e envia seu resultado para todos os outros clientes. A Figura 5 exemplifica esse modelo de comunicação.

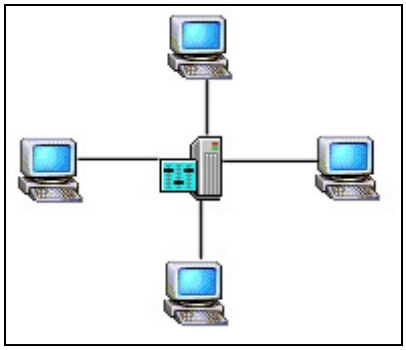

Figura 5 – Arquitetura de comunicação Cliente-Servidor

Outra característica de interações C/S é o servidor ter controle absoluto e final sobre a utilização do ambiente virtual e poder, em princípio, controlar, validar

e sincronizar todas as ações de todos os usuários do sistema. Esse controle é fundamental em sistemas comerciais e, em grande parte, justifica a escolha da arquitetura cliente-servidor para sistemas dessa categoria.

Uma desvantagem desse modelo de comunicação, no entanto, é que o servidor é um possível gargalo no sistema, em termos de processamento e gasto de largura de banda. Em grandes MMORPGs isso é um problema real, sendo uma das fontes de seu alto custo de implantação quando comparado com outros tipos de jogos (INTERNATIONAL GAME DEVELOPERS ASSOCIATION, 2005).

Alguns exemplos de sistemas que utilizam a arquitetura cliente-servidor são Barrus, Waters e Anderson (2005), Brutzman et al. (2005) e Oliveira (2001).

A outra arquitetura proposta para utilização em DVEs é a ponto-a-ponto. Nesse modelo não existe um servidor central ou este está reduzido ao mínimo de tarefas possíveis (em geral, sua única responsabilidade é permitir o contato inicial de clientes que não têm conhecimento prévio um do outro). As mensagens são trocadas entre os diversos usuários do sistema em uma rede lógica altamente conectada, onde cada nó da rede está conectado a todos os outros nós. Cada mensagem, portanto, é enviada diretamente a todos os usuários da rede sem a ajuda de intermediários.

Essa característica permite a diminuição da latência das mensagens, porém cria outras dificuldades na implementação de um DVE, sendo a escalabilidade do sistema em relação ao gasto total de largura de banda e a segurança da simulação dois exemplos. A Figura 6 ilustra a topologia de uma rede ponto-a-ponto.

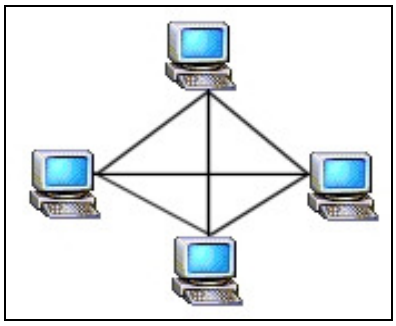

Figura 6 – Arquitetura de comunicação Ponto-a-Ponto

Alguns exemplos de DVEs projetados para a utilização de arquitetura P2P são: Chapman (2005), Hu e Liao (2005) e Knutsson et al. (2005).

Uma modalidade menos estudada é a dos ambientes virtuais que incorporam elementos de ambas arquiteturas, criando um modelo híbrido de comunicação (ilustrado na Figura 7). Como exemplo dessa arquitetura, destaca-se o trabalho de Feijó e Kozovitz (2005), onde servidores *web* são usados para um "nível" do jogo (estratégico) contatado através de comunicação estilo cliente-servidor, enquanto comunicações ponto-a-ponto são utilizadas em cada "sala" (ou partida individual) entre os participantes para a execução de um jogo de ação.

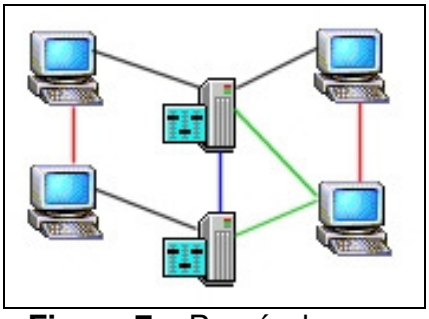

Figura 7 – Possível topologia para uma arquitetura híbrida

Outro trabalho nesse sentido é o de Pellegrino (2005) que determina a utilização de uma arquitetura híbrida chamada de Ponto-a-Ponto com Árbitro Central

(P2P-CA), onde informações são trocadas entre usuários em uma arquitetura P2P sem checagens de consistência, sendo qualquer conflito resolvido através de um árbitro central.

Já os trabalhos de Knutsson et al. (2004) e Chapman (2005) utilizam uma rede *overlay* chamada Pastry para simular comunicação ponto-a-ponto. Uma rede *overlay* funciona como uma rede lógica de roteamento de mensagens sobre a internet. Cada participante dessa rede tem um endereço e toda mensagem enviada por um *host* é mandada para um nó mais próximo do destino da mensagem. Experimentalmente, pode-se determinar que em média as mensagens utilizam um número de saltos (*hops*) logarítmico para alcançar seu destino, porém como cada transmissão individual é uma mensagem TCP ou UDP completa, a latência final pode ser inaceitável para aplicações que tenham requerimentos muito estritos.

Com relação aos protocolos utilizados para a transmissão de dados entre usuários, os dois mais populares protocolos de transporte são o *Transmission Control Protocol* (TCP) e o *User Datagram Protocol* (UDP), ambos utilizados como base para protocolos de aplicação mais específicos.

O protocolo TCP utiliza um modelo de conexão virtual, criando um canal de fluxo de dados lógico que permanece ativo enquanto os usuários assim o desejarem (ou as condições da sub-rede permitirem). Por isso ele é classificado como orientado à conexão e confiável, uma vez que a chegada e ordenação dos dados é garantida pela especificação do protocolo (novamente, caso as condições da sub-rede não forcem um desligamento da conexão).

O protocolo UDP, ao contrário, é classificado como sem conexão e não confiável, já que ele transporta apenas datagramas (mensagens completas) e não dá garantias sobre a ordem ou sequer a chegada das mensagens em seu destino.

Uma razão dada para a utilização de UDP ao invés de TCP em aplicações que requeiram atualização em tempo real é a de que o atraso (latência ou *latency*) ou perda de pacotes (*packetloss*) no protocolo TCP acarreta uma interrupção do fluxo de dados até que a transmissão seja restabelecida, o que não acontece no UDP. Como entre as exigências para aplicações de tempo real, em geral encontra-se um fluxo de dados à taxa constante ao invés de transmissão perfeita, esse atraso pode acarretar perda de sincronia e conseqüente perda de desempenho do software (LARZON; DEGERMARK; PINK, 2005).

Considerando o modo de troca de mensagens entre os usuários de um ambiente virtual, um sistema DVE pode utilizar comunicações *unicast*, *multicast* ou *broadcast*.

Comunicação *unicast* é realizada unicamente entre dois nós da rede: entre um cliente e um servidor ou entre dois pontos. Esse tipo de comunicação é a mais usada em sistemas comerciais por ser a mais disponível aos usuários domésticos e tanto o protocolo TCP quanto o UDP implementam essa funcionalidade.

A tecnologia *multicast* consiste em utilizar grupos de comunicação onde apenas uma mensagem é enviada por um cliente e todos os outros participantes recebem essa mensagem. Isso é conseguido através de uma rede de roteadores subjacente que permita selecionar as sub-redes e eventualmente *hosts* para a qual a mensagem deve ser enviada. Isso significa que o remetente envia uma informação uma única vez e ela atinge um grande número de pessoas, porém a um custo: poucos nós da Internet suportam o envio de comunicação *multicast*, o que a torna inadequada para sistemas comerciais ou que visam utilização por usuários domésticos (ZHANG; JAMIN; ZHANG, 2005). Um método para solucionar esse

problema pode ser encontrado no sistema proposto por Purbrick e Greenhalgh (2005) que utiliza *multicast* quando possível, e *unicast* por protocolo UDP quando necessário.

Finalmente, a tecnologia *broadcast* está disponível apenas dentro de redes locais e distribui uma determinada mensagem a todos os nós da rede, estejam eles interessados na mensagem ou não. Isso provoca uma perda de desempenho não só nos usuários individuais, mas também na rede como um todo e, em geral é evitado. Apenas o protocolo UDP tem capacidade para enviar mensagens de *broadcast*.

3.3 DIVISÃO ESPACIAL

Um assunto de grande importância no projeto de sistemas DVE é a forma de particionamento geográfico do ambiente virtual.

O trabalho de Barrus, Waters e Anderson (2005) fornece a formalização do conceito de *locale* como uma unidade geográfica que guarda os elementos visuais, sonoros e lógicos de um “pedaço” do ambiente virtual. *Locales* podem ser conectados através de suas bordas, de tal forma que, ao atravessar o limite do *locale* onde está localizado, um usuário emerge em outro.

Um *locale* é caracterizado por três propriedades chave:

a) Canais de comunicação: Cada *locale* define um grupo de nós que tem a capacidade de trocar mensagens entre si (em sua definição primordial, isso é feito através do uso de um grupo *multicast*);

b) Sistemas de coordenada locais: Cada *locale* mantém seu próprio sistema de coordenadas;

c) Relações e geometria arbitrária: A posição, orientação, forma e conteúdo de um *locale* pode ser definido arbitrariamente, sem a necessidade de qualquer tipo de coordenação global. A relação entre dois *locales* vizinhos é feita localmente (ou seja, entre cada dupla de *locales*) e de maneira unilateral (uma ligação não precisa ser bidirecional).

A Figura 8 mostra como uma casa poderia ser dividida, comportando um *locale* para cada cômodo. A sala (em vermelho) número cinco contém um usuário ativo do ambiente virtual, enquanto as salas um, oito, nove e dez mostra um conjunto de salas cujo conteúdo é de interesse do usuário (ou seja, onde modificações irão influir na sua representação local da cena).

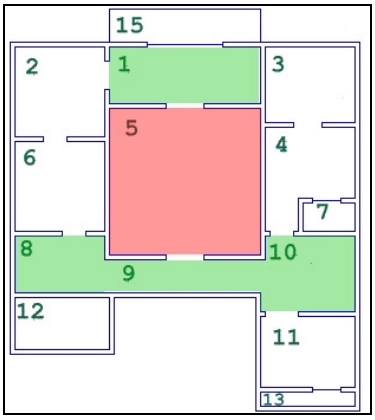

Figura 8 – Planta de uma casa separada em *locales* (cômodos numerados)

Snowdon, Greenhalgh e Purbrick (2005) expandem o conceito de *locales* para abrigar aspectos (*aspects*) diferentes de suas entidades. No sistema MASSIVE-3, todo objeto deve estar contido em apenas um *locale* e em apenas um

aspecto desse *locale*. Um aspecto define conjuntos de objetos de um determinado tipo (modelos, sons, usuários) logicamente relacionados (terreno, casas, mobília, jogadores, espectadores) e que são representados com uma determinada fidelidade (qualidade, medida de acordo com o tipo de objeto, como número de polígonos, taxa de amostragem ou largura de banda). A utilização de aspectos permite criar um ambiente virtual que suporte usuários com maior ou menor poder de processamento (que podem selecionar conjuntos de objetos mais simples ou complexos através do parâmetro fidelidade) ou que só estejam interessados em determinados tipos de entidades. Todo usuário interessado nas mudanças ocorridas em um aspecto deve "assiná-lo", registrando-se como possível destinatário para mensagens de atualização.

Além de permitir uma clara separação entre seções de um mundo virtual e fácil concatenação dessas seções, *locales* e *aspects* permitem utilizar algoritmos de oclusão de mensagens de maneira transparente, embora Purbrick e Greenhalgh (2005) definam o que chama de "políticas de seleção", onde as informações de mais de um *locale* podem ser acessadas ao mesmo tempo, de acordo com a configuração do ambiente virtual e das preferências do usuário.

Essas políticas de seleção funcionam interpretando *locales* e bordas como um dígrafo (respectivamente, nós e arcos) e percorrendo os caminhos a partir do *locale* onde o usuário está localizado, segundo um conjunto predefinido de regras. O uso de políticas de seleção permite um controle mais granulado da informação que é apresentada ao usuário, uma vez que as diversas características do ambiente virtual (como os modelos tridimensionais, os sons e os outros usuários) podem ser controladas individualmente.

Uma outra maneira de organização espacial pode ser encontrada em Activeworlds (2005), onde cada ambiente individual (mantido por pessoas diferentes e rodando como processos diferentes) é subdivido em diversas células de mesmo comprimento e largura, porém altura infinita. A introdução dessa peculiaridade pode ser explicada pela observação de que o mundo real é, em larga escala, bidimensional, quando observando as construções naturais e artificiais de nosso planeta. Esse método de particionamento espacial também é conhecido como divisão em grade, e pode ser observado na Figura 9 (onde a célula em vermelho indica a posição de um usuário e as células em verde indicam posições vizinhas que são de seu possível interesse).

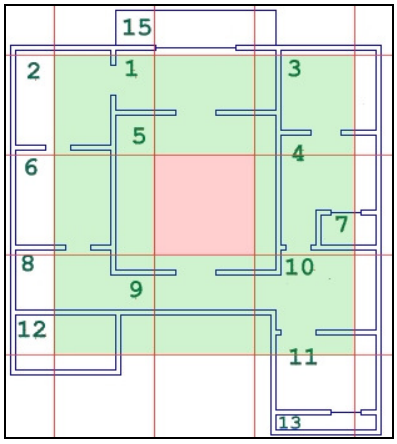

Figura 9 – Exemplo de particionamento em grade

Tanto a divisão de ambientes virtuais através de *locales*, quanto a utilização de células encontradas no Active Worlds são divisões estáticas, ou seja, estabelecidas durante a construção do cenário. Um outro modelo de subdivisão de ambientes virtuais é sugerido por Greenhalgh (2005) que utiliza *quadtrees* (árvores que contém até quatro filhos por nó) para separar o ambiente virtual em grupos de *multicast*, que são organizados dinamicamente para suportar aumento e diminuição do número de usuários do sistema.

Também dinâmica é a arquitetura proposta por Lui e Chan (2005), onde um ambiente virtual é particionado de acordo com o número de usuários e designado para diferentes servidores, visando balancear a quantidade de processamento executado em cada servidor individual.

3.4 ESCALABILIDADE E ALGORITMOS DE OCLUSÃO DE MENSAGENS

A questão da escalabilidade de um sistema DVE é um assunto primordial em seu projeto. Por escalabilidade, entende-se as condições que permitam um ambiente virtual suportar um crescente número de usuários da maneira mais transparente possível. A escalabilidade pode ser estudada em termos de gasto de largura de banda total, dos servidores e dos clientes, latência, número de mensagens trocadas, processamento dos servidores e dos clientes e do próprio código do DVE.

A escalabilidade da largura de banda e da latência é, em geral, considerada em conjunto, uma vez que o aumento de desempenho em uma muitas vezes acarreta perda em outra.

A arquitetura de comunicação cliente-servidor reduz o consumo de largura de banda dos clientes introduzindo um intermediário para todas as comunicações, resultando em aumento de latência. A redução de consumo acontece porque cada cliente só precisa enviar uma determinada mensagem uma única vez, tendo como destinatário o servidor, ao invés de enviar a mesma mensagem diretamente para todos os outros clientes. Já o aumento de latência decorre do

caminho extra que a mensagem deve percorrer: em essência, o caminho entre o cliente e o servidor. Um problema de escalabilidade introduzido por esta arquitetura, no entanto, é que a largura de banda do servidor passa a ser um possível gargalo do sistema, já que toda mensagem enviada por um cliente precisa ser replicada para N-1 clientes (onde N é o total de usuários do sistema).

Na arquitetura ponto-a-ponto ocorre o oposto. O gasto da largura de banda é distribuído entre todos os usuários, portanto cada ponto individual precisa utilizar maior quantidade deste recurso. No entanto, como a comunicação é direta (ou seja, sem intermediários) a latência é diminuída.

Já no sistema de Pellegrino e Dovrolis (2005), onde se utiliza um árbitro central para validar as operações do ambiente virtual e resolver os conflitos de consistência que possam aparecer, o gasto principal de largura de banda é feito como em um sistema P2P, enquanto o árbitro que resolve os problemas de consistência de estado utiliza menos largura de banda que um servidor na arquitetura cliente-servidor.

Com relação ao número de mensagens trocadas, as tecnologias de *broadcast* e *multicast* têm um desempenho teórico similar menor do que as mensagens trocadas por *unicast*: cada cliente precisa enviar apenas uma única mensagem que será recebida por todos os participantes do ambiente virtual. Porém, como citado na seção 3.1, as mensagens de broadcast têm escopo local, significando que não podem ser utilizadas em um ambiente virtual implantado sobre uma rede de longa distância como a Internet, além do que a tecnologia de *multicast* não está disponível para grande parte do público doméstico.

O número de mensagens trocadas entre os participantes de um ambiente virtual pode ser reduzido ainda mais com a utilização de técnicas de *dead-*

reckoning (PANTEL; WOLF, 2005). Com essa técnica, somente quando a variação entre uma propriedade de um objeto controlado pelo usuário (por exemplo, sua posição no mundo) tem uma variação superior a um valor predeterminado a informação é enviada para os outros usuários.

A latência absoluta (ou seja, o tempo efetivo que uma mensagem leva para partir de um usuário e chegar aos outros) de uma transmissão influi na percepção que os usuários têm do mundo virtual, porém a latência relativa, definida como a diferença de latência entre os usuários de um sistema, também pode determinar a qualidade da simulação. Henderson (2005) determina que a latência absoluta é mais perceptível por usuários de jogos multiplayer, e que os usuários tendem a adquirir tolerância à latência conforme seu envolvimento no jogo aumenta (conforme indicado pelo tempo gasto dentro de uma partida).

Nas arquiteturas discutidas até aqui, considerava-se que cada mensagem de um usuário é replicada em direção a todos os outros usuários do sistema. No entanto, devido às particularidades de um sistema DVE que contenha algum tipo de particionamento geográfico (ou seja, que simule um terreno, salas, etc) é possível executar uma operação que pode ser denominada como oclusão de mensagens. Essa técnica consiste em utilizar a localidade de interesse de um usuário (a característica de que em um ambiente geográfico um usuário está interessado em apenas um subconjunto de todas as suas informações) para determinar os nós que devem compartilhar informações (ZOU; AMMAR; DIOT, 2005).

O conceito de *locale* apresenta um dos modelos mais simples de oclusão de mensagens: apenas usuários dentro de uma mesma região do ambiente

virtual (ou seja, dentro de um *locale*) precisam trocar mensagens entre si, reduzindo a carga de servidores que contenham *locales* diferentes.

Com a introdução das políticas de seleção é possível definir um controle mais granulado do processo de troca de informações, aumentando ou diminuindo o número de usuários de acordo com as características do *locale* e de cada participante individual.

Os aspectos também podem funcionar como sistema de oclusão de mensagens, bastando para isso o usuário assinar ou não os aspectos relacionados à troca de mensagens. Uma outra função interessante desse conceito é permitir selecionar apenas um subconjunto lógico das mensagens que um usuário deseja receber. No exemplo de Snowdon, Greenhalgh e Purbrick (2005), em uma arena de esportes *online* os espectadores estão mais interessados nos jogadores do que em outros espectadores, portanto as políticas de seleção devem privilegiar esse aspecto do cenário.

Embora úteis para reduzir o número de mensagens trocadas, *locales* podem ainda ser unidades relativamente grandes para a determinação dos testes de oclusão de mensagens. Para resolver esse problema, utiliza-se uma outra técnica chamada de Área (ou Aura) de Interesse. Nessa técnica, cada usuário define uma área (em geral um raio) além do qual nenhum outro usuário será contatado para troca de mensagens (SMED; KAUKORANTA; HAKONEN, 2005).

Uma melhoria introduzida por Oliveira (2001) foi a generalização do conceito de área de interesse para a utilização de diversas métricas diferentes (número de usuários, distância dentro do mundo virtual, latência média, entre outros) para a determinação de quais nós devem se comunicar. Além disso, as áreas de interesse definem uma dupla borda para reduzir o gasto de performance decorrente

de múltiplas entradas ou saídas de usuários, denominadas como área de *check-in* e área de *check-out*. A primeira determina quando as atualizações de um usuário devem ser recebidas e a segunda, quando devem ser descartadas.

A escalabilidade do código de um DVE pode ser entendida como a sua capacidade de adaptar-se a novas funções ou recursos com o menor número de mudanças possível. Essa característica também pode ser estudada através da extensibilidade do sistema: como novos comportamentos podem ser adicionados ao conjunto já existente, com o menor número de alterações possível.

Nesse sentido as arquiteturas de *microkernel* como implementadas pelo sistema NPSNET-V (KAPOLKA; MCGREGOR; CAPPS, 2005) são as que apresentam melhores possibilidades de extensão, uma vez que apenas seu núcleo mínimo é invariável e todos os outros componentes estão contidos em módulos independentes.

A extensibilidade também pode ser estudada em termos de protocolo da aplicação, já que este precisa se adaptar a novas características dos dados. Os trabalhos de Fischer (2001) e Serin (2005) discutem a utilização de protocolos baseados no padrão de documentos XML. Serin estabelece o XFSP, um protocolo que pode expandido enquanto o ambiente virtual está sendo executado e que pode, inclusive, utilizar fidelidades diferentes para aumentar o desempenho ou a qualidade da simulação, conforme a necessidade do usuário.

Já a extensibilidade dos recursos do sistema (modelos tridimensionais, sons, imagens) também é um assunto importante dentro do estudo de DVEs. Nesse quesito, formatos de arquivos baseados no padrão XML suportam a adição ou remoção de características sem quebrar a compatibilidade de sistemas antigos.

Com relação à ambientes virtuais, o padrão VRML (ISO/IEC, 2005) foi a primeira tentativa de padronização (embora sem suporte nativo a múltiplos usuários por mundo), porém sua capacidade de receber novas funcionalidades era limitada. A partir dos resultados obtidos com sua utilização, o Web 3d Consortium (2005) estabeleceu o padrão *Extensible 3D* (X3D), que pode utilizar codificação XML para seus documentos, além de conter uma arquitetura modular, que permite a adição de componentes por outros desenvolvedores. O sistema de NetNodes de Araki (2005) foi o primeiro à propor um conjunto de extensões (denominadas NetNodes) para suporte à múltiplos usuários em um ambiente virtual baseado no padrão VRML (e, conseqüentemente, no padrão X3D).

3.5 SEGURANÇA

A segurança em ambientes virtuais é de extrema importância, especialmente para os MMORPGs que têm objetivos comerciais.

Uma das principais tarefas de qualquer sistema DVE é a sincronização do estado dos objetos que podem ser manipulados pelos usuários, ou seja, a manutenção da consistência entre as diversas cópias dos objetos, presentes localmente em cada um dos participantes da simulação.

Em uma arquitetura cliente-servidor, todas as atualizações feitas sobre um objeto devem ser enviadas primeiro ao servidor, que será o responsável por enviar essa atualização aos outros usuários interessados (onde “usuários interessados” pode ser definido como aqueles que passaram nos testes de oclusão

de mensagens). Nessa arquitetura, o servidor tem poder completo de decisão e mantém o ambiente altamente consistente, dependendo unicamente da qualidade dos canais de comunicação entre o servidor e os clientes.

Em uma arquitetura ponto-a-ponto a garantia de consistência já se torna mais difícil. Em primeiro lugar, é preciso descobrir quem é o nó da rede responsável por determinado objeto e então sincronizar a cópia local e a cópia remota. Nos sistemas de Knutsson et al. (2005) e Chapman (2005) isso é conseguido através da rede *overlay* ponto-a-ponto Pastry, onde um código *hash* de um determinado objeto é utilizado para descobrir-se o endereço do nó que deve armazená-lo. Uma outra alternativa para sistemas que utilizam redes P2P é atualizar todos os usuários do ambiente virtual com as alterações desejadas.

Além da modificação indevida do estado dos objetos, um outro ataque possível em jogos rodando sobre uma arquitetura DVE é a utilização da informação dos pacotes de sincronização (ou seja, as mensagens transportadas através da rede de computadores que carregam atualizações dos objetos do ambiente virtual) para ganho de vantagem. Nesse tipo de ataque, um cliente pode escolher não levar em conta mensagens que produzam efeito adverso em sua cópia local do ambiente (por exemplo, descartando uma mensagem que carrega um tiro certeiro).

Para resolver esse tipo de problema, Baughman et al. (2005) define o protocolo *Lockstep* que valida as ações dos participantes de um ambiente virtual em turnos, onde todos os usuários devem enviar as informações de ação antes que elas possam ser efetivamente realizadas.

Já Cronin (2005) introduz o *Sliding Pipeline*, que adiciona *buffers* de envio e recebimento de mensagens e transforma o protocolo Lockstep em um protocolo dinâmico, capaz de utilizar soluções de *dead-reckoning* para suavizar a

alteração de certas propriedades dos objetos (como a posição de um usuário que está se movendo).

4 EXTENSIBLE 3D

O padrão *Extensible 3D* (X3D) foi desenvolvido como uma evolução do *Virtual Reality Modelling Language* (VRML) para a criação de cenários tridimensionais interativos. Este padrão pode ser encarado como o sucessor da versão 97 do VRML, incorporando melhorias necessárias para seu aprimoramento.

As principais características que definem o padrão X3D (e que o fazem diferenciar-se do VRML original) são componentização, homogeneização do comportamento dos ambientes virtuais e formatos de codificação.

Ao contrário do padrão VRML no qual um *browser* precisa implementar toda a especificação para permitir a correta visualização dos ambientes tridimensionais, o X3D separa sua funcionalidade em blocos denominados componentes. Componentes são agrupados em *profile*s, que definem a funcionalidade mínima que um software visualizador de cenários X3D deve implementar.

Já a homogeneização do comportamento dos cenários se refere a um problema existente no padrão VRML, onde os aspectos visuais de um ambiente variam entre *browsers* diferentes. O X3D resolve este problema criando uma especificação detalhada dos resultados esperados de cada funcionalidade que pode estar presente dentro dos ambientes interativos.

Outra evolução particularmente notável foi a introdução do conceito de formatos de codificação. Enquanto arquivos VRML tinham uma formatação específica, arquivos X3D podem ser codificados de várias formas, incluindo através do padrão XML. Uma codificação baseada no formato VRML (chamada de VRML classic), no entanto, ainda pode ser usada para obter maior compatibilidade com ferramentas existentes.

Atualmente o padrão X3D é desenvolvido por uma organização internacional chamada Web 3d Cosortium ligada à ISO (*International Standards Association*) e à W3C (*World Wide Web Consortium*), sendo responsável pela modificação, atualização e correção das diversas especificações X3D. Estas especificações são abertas, podendo ser implementadas em qualquer software sem a necessidade de pagamento de royalties, e todos os membros do consórcio podem submeter novos conjuntos de componentes para inclusão em emendas oficiais. A primeira versão do padrão sancionada pela ISO foi liberada em 01 de dezembro de 2004.

As seções a seguir detalham as características do padrão X3D. Essas informações foram retiradas diretamente da especificação ISO/IEC 19775-1 (WEB 3D CONSORTIUM, 2005).

4.1 X3D BROWSERS

Os arquivos X3D (também denominados mundos virtuais) são apresentados aos usuários através de softwares denominados *browsers*. Assim

como em web *browsers*, é a responsabilidade de um *browser* X3D exibir as formas, sons e interações especificadas em um arquivo desse formato. No entanto, as funcionalidades específicas (determinadas através do *profile* requerido pelo arquivo) podem variar entre aplicações diferentes.

Uma das principais funções de um *browser* X3D é permitir a execução do mundo virtual, sua navegação e a interação entre o usuário e os diversos objetos desse cenário.

Código externo (chamado código de usuário na terminologia da especificação X3D) também pode interagir com as informações presentes dentro do mundo virtual (ou mais tecnicamente, com o grafo de cena que representa um arquivo X3D) através da *Scene Access Interface*, seja através de nós *Script* ou de maneira dependente de implementação.

4.2 CAMPOS, NÓS E PROFILES

Em um arquivo X3D, as informações sobre as entidades e suas relações lógicas e temporais são representadas em unidades denominadas de objetos. Um objeto que defina uma unidade de armazenagem (por exemplo, um inteiro ou uma referência a outro objeto) e eventos é denominado campo (field). Já unidades mais complexas, compostas por conjuntos de campos são chamadas nós (*nodes*). A semântica de cada nó, qual tipo de informação ele representa e como um *browser* deve apresentá-lo para um usuário é definida dentro da especificação do padrão X3D.

A especificação de um campo deve indicar, além do tipo de valor aceito, o modelo de acesso oferecido, tanto para outros objetos quanto para código de usuário acessado através da SAI. Os tipos de acesso disponíveis são: *inputOnly* (que define um campo que pode apenas iniciar eventos – em essência, um campo apenas para escrita), *outputOnly* (usado para definir campos que lançam eventos, sinalizando-os para outros campo através de rotas – em essência, um campo apenas de leitura), *inputOutput* (uma combinação dos dois tipos anteriores) e *initializeOnly* (o valor do campo pode ser apenas inicializado e nunca mais alterado).

Nós podem ser implementados dentro do próprio *browser* (no caso dos nós oficiais, contidos dentro da especificação) ou através de estruturas denominadas de protótipos. A definição de protótipos data da criação do padrão VRML, e permite estender a funcionalidade de um arquivo com novos nós que contenham um número arbitrário de campos.

Um conjunto de nós que define uma funcionalidade correlacionada é definido como um componente, e um grupo de componentes é agrupado em um *profile*.

Um *profile* é um contrato de garantia que um *browser* faz com relação aos autores de arquivos X3D: todo *browser* que declara suportar um *profile* deve implementar os nós contidos dentro deste, no nível (*level*) mínimo definido. Um nível define as características (em geral, os campos) disponíveis em cada nó, e é cumulativo com relação aos níveis anteriores.

A Figura 10 ilustra o esquema básico de organização de *profile*s, componentes (dois componentes são mostrados: *Geometry3D* e *Grouping*), nós (*Box* e *Sphere* do componente *Geometry3D* e *Transform* do componente *Grouping*) e alguns de seus campos constituintes (*size*, *radius*, *translation* e *rotation*).

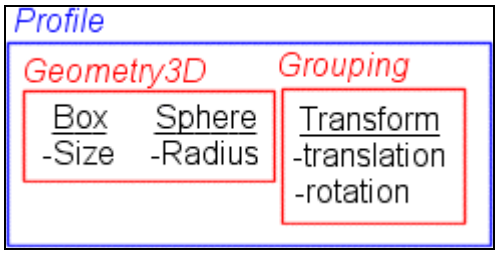

Figura 10 – Esquema de organização de *profile*s, componentes, nós e campos

4.3 GRAFOS DE CENA (SCENE GRAPHS)

Os objetos presentes em um arquivo X3D são organizados de maneira hierárquica reproduzindo um padrão de nós e arcos direcionados conhecido como grafo de cena. Um grafo de cena é iniciado com os nós-raiz (*root nodes*) que não têm relação de parentesco com (não são “filhos” de) nenhum outro nó. Uma relação de parentesco determina que um nó é “filho” (ou sucessor) de outro (seu “pai” ou antecessor).

O padrão X3D especifica dois grafos separados: um para as transformações (*transformation hierarchy* – hierarquia de transformações) e um para comportamentos (*behaviour graph* – grafo de comportamento).

A hierarquia de transformações define um conjunto cumulativo de operações que posiciona e orienta as formas geométricas que podem ser visualizadas dentro de um *browser*. O nó mais utilizado para efetuar essas operações é denominado *Transform*, e as três operações básicas executadas sobre as formas geométricas são: translação (*translation*) que modifica a posição de um

objeto por uma determinada quantidade, rotação (*rotation*) que define um eixo e um ângulo de rotação do sistema de coordenadas em relação a esse eixo e escala (*scale*) que permite alterar o tamanho aparente das formas. Por “cumulativo” entende-se que as transformações executadas em um nó são um reflexo de todas as operações executadas no seu conjunto de nós antecessores. O padrão X3D define como metros a unidade utilizada para coordenadas espaciais, segundos para coordenadas temporais, radianos para ângulos e valores normalizados ([0..1] [0..1] [0..1]) para o espaço de cores RGB.

O grafo de comportamentos é o conjunto de conexões (rotas) entre campos dos nós de um arquivo X3D que é ativada quando um evento é disparado de um nó script ou sensor. A maneira como essa hierarquia é processada é determinada pela especificação do modelo de eventos.

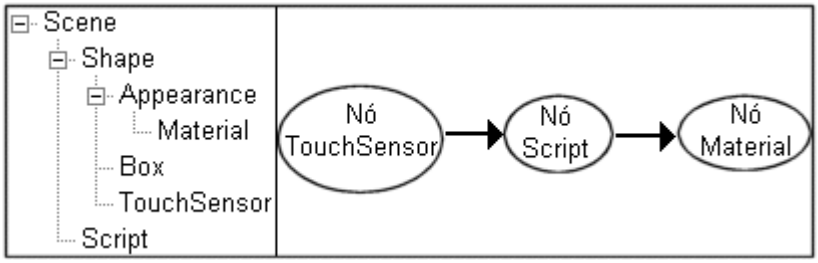

Figura 11 – Grafos (transformações e comportamentos) do padrão X3D

A Figura 11 mostra um possível arquivo X3D, ilustrando o uso dos dois grafos de cena. A primeira coluna mostra a hierarquia de transformações geométricas, enquanto a segunda coluna mostra como um conjunto de rotas pode ser estabelecido entre diversos nós, dando origem ao grafo de comportamentos.

4.4 MODELO DE EVENTOS

A geração de comportamento (interativo ou temporal) dentro de um arquivo X3D é determinada pelo modelo de eventos. Um evento é uma entidade que informa uma mudança do valor de um campo X3D para outros nós do grafo de cena (por exemplo, eventos *fraction_changed* do nó *TimeSensor*) e o conjunto de eventos propagados em decorrência de um evento inicial é chamado de cascata de eventos (*event cascade*).

Todo evento indica o campo onde houve uma mudança, além de carregar um identificador de quando esse evento foi lançado. Esse indicador (conhecido como *timestamp*) é usado para determinar a causalidade dos eventos (em essência, qual a seqüência temporal dos eventos) e para tratamento dos laços (*loops*) de rotas. Uma das especificações do padrão X3D é que a causalidade deve ser sempre preservada: a execução de eventos deve sempre ser efetuada com valores de *timestamp* crescentes (ou seja, um evento nunca pode ser executado depois de outro com *timestamp* superior ao seu). A execução de eventos simultâneos (onde simultaneidade é definida como eventos com *timestamps* idênticos) pode ser implementada pelo *browser* de maneira arbitrária.

Os eventos trafegam pelo grafo de comportamentos através de rotas (*routes*). Uma rota liga um campo de saída (ou seja, um que tenha tipo de acesso *outputOnly* ou *inputOutput*) à um campo de entrada (tipos de acesso *inputOnly* e *inputOutput*). Todo evento que chega em um campo de entrada modifica as propriedades do nó que o contém, e todo evento lançado de um campo de saída indica uma alteração da propriedade relacionada.

Uma possibilidade criada pelo sistema de rotas, no entanto, são os laços (*loops*) que podem ser formados quando um evento E gerado em um nó N atravessa diversas rotas até eventualmente chegar novamente ao nó N provocando um novo ciclo de roteamento de E. Caso esse laço fosse permitido executar continuamente, o *browser* entraria em uma repetição infinita (pois precisaria manter a regra da causalidade, não gerando novos eventos enquanto E não fosse completamente tratado), impedindo a continuidade da visualização do mundo. A especificação X3D resolve este problema determinando que apenas um evento possa ser gerado em cada campo de cada nó para cada *timestamp* individual, portanto a implementação dos *browsers* deve ser capaz de identificar essas situações.

Dois outros mecanismos decorrentes do sistema de rotas são o *Fan-In* e o *Fan-Out*. *Fan-ins* ocorrem quando duas ou mais rotas tem o mesmo campo de destino (em outras palavras, quando dois ou mais campos de entrada estão ligados ao mesmo campo de saída). Já os *Fan-outs* representam a situação inversa: quando um campo é fonte de mais de uma rota (ou seja, um campo de saída está ligado a vários campos de entrada).

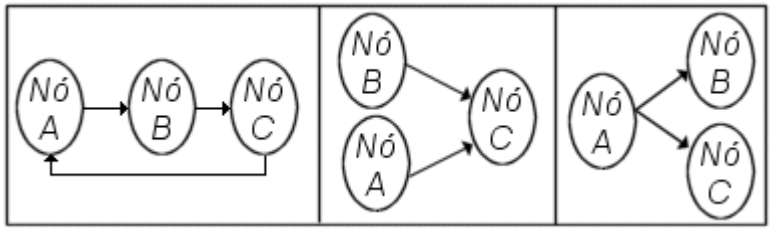

Figura 12 – *Loops*, *fan-ins* e *fan-outs*

A Figura 12 exemplifica as três situações descritas anteriormente: Um *loop* ocasionado pela ligação em cascata dos nós A, B e C, um *fan-in* criado pela ligação de A e B com o nó C e um *fan-out* iniciado por A em direção aos nós B e C.

4.5 PROTÓTIPOS

O mecanismo de protótipos permite estender a funcionalidade de arquivos X3D através da definição de novos tipos de nós utilizando como base àqueles pré-existentes na especificação. Cada novo nó criado como um protótipo deve ter um nome único e pode ser utilizado dentro de mundos virtuais da mesma maneira que os nós X3D originais.

Um protótipo pode ser definido de duas maneiras: dentro do próprio arquivo onde ele será usado ou em um arquivo diferente que poderá ser importado. Nesse segundo caso, um protótipo é chamado de protótipo externo (*ExternProto*), e para utilizá-lo basta declarar sua existência e indicar onde (em qual arquivo) sua definição está localizada.

Todo protótipo é definido em duas etapas. A seção de interface especifica quais novos campos esse protótipo declara, seus nomes, tipos de acesso e valores padrão. Em seguida cria-se o corpo do protótipo, que contém um número arbitrário de nós que serão utilizados como base dessa nova classe. Finalmente, especificam-se rotas (*routes*) de comunicação dentro dos nós contidos na seção de corpo e (opcionalmente) conexões entre os campos dos nós internos e os campos identificados na seção de interface.

A especificação X3D estabelece que apenas o primeiro nó declarado no corpo de um protótipo será adicionado ao grafo de cena, e conseqüentemente visualizado. No entanto, essa aparente limitação é resolvida declarando o primeiro nó como um grupo (*Group*) que é capaz de conter outros nós como seu filho.

4.6 NÓS SENSORES

A entrada de dados executada pelo usuário em cenários interativos dentro do padrão X3D é feita através de nós especiais chamados de sensores. Esse tipo de nó é o iniciador de uma cascata de eventos, sendo os principais tipos de sensores a manipulação da geometria pelo usuário, sua movimentação pelo cenário ou passagem de tempo. Nós de *script*, com a utilização da SAI também são capazes de iniciar eventos, porém são discutidos na seção seguinte.

Entre nós sensores incluem-se *TouchSensor*, *TimeSensor* e *Collision*.

O nó *TouchSensor* permite a detecção de interação do usuário através de dispositivos de ponteiro. O padrão X3D não define o que é esse dispositivo, sendo deixado a cargo de implementações. Em um *browser* rodando sobre computadores *desktop*, esse nó utiliza o cursor do mouse como indicador.

O nó *TimeSensor* é o responsável por iniciar eventos relacionados à passagem de tempo e pode ser usado para eventos contínuos (como animações), eventos periódicos (que ocorrem a cada minuto, por exemplo) ou eventos únicos, iniciados em um tempo específico (como um evento iniciado quanto o *TimeSensor* atinge um determinado valor).

Já o nó *Collision* é responsável pela detecção de colisão entre a câmera do usuário e um determinado conjunto de formas geométricas.

4.7 NÓS DE SCRIPT

Para permitir extensão dos mundos virtuais, o padrão X3D define um nó para introdução de *scripts* que pode receber, processar e enviar eventos dos grafos de cena através da *Scene Access Interface* (SAI). A SAI é um conjunto de definições relacionadas aos serviços que um ambiente de programação (linguagem, interpretador, etc) deve oferecer para que o código externo (ou código de usuário) possa interagir com o conteúdo de uma cena tridimensional.

O nó *Script* é capaz de utilizar código contido dentro do próprio arquivo X3D (técnica chamada de *inline scripting*) ou em outros arquivos que possam ser localizados através de URLs.

Em sua definição primordial, o padrão X3D não define quais linguagens de script um *browser* deve suportar, sendo isso uma característica dependente de implementação. Desde sua criação, no entanto, duas linguagens já tiveram uma especificação aprovada pelo *Web 3d Consortium*. São elas: ECMAScript (uma padronização da linguagem *javascript* encontrada em *browsers* da *web* - ECMA INTERNATIONAL, 1999) e a linguagem de programação Java (GOSLING, 2005).

4.7.1 ECMAScript

A linguagem ECMAScript foi aprovada como o padrão ISO/IEC 16262 em 1998 visando criar uma linguagem para a computação de funcionalidades

dinâmicas em *browsers* da *World Wide Web* que não estivesse presa a vendedores específicos.

Esta é uma linguagem orientada a objetos com uma sintaxe construída para se assemelhar à linguagem de programação Java, com certas provisões (como possibilidade de utilizar variáveis não-tipificadas) que a tornam mais simples para usuários leigos.

A ECMAScript não contém métodos de entrada ou saída, e depende exclusivamente de um ambiente computacional (conhecido como *host environment*) que proporcione esses serviços, como por exemplo um *browser* web.

A mais recente adição a essa linguagem foi a introdução de extensões para manipulações de arquivos XML, no padrão denominado E4X (ECMAScript for XML).

O padrão ISO/IEC 19777-1:200x descreve os detalhes da ligação entre a ECMAScript e a *Scene Access Interface* (SAI) caso um *browser* deseje oferecer essa linguagem para execução de *scripts*.

4.7.2 Java

A linguagem Java, desenvolvida originalmente para utilização como uma linguagem embutida (embedded) alcançou alto índice de difusão devido à sua inclusão em *browsers web* no início da popularização da Internet.

Essa é uma linguagem Orientada a Objetos chamada híbrida por ser tanto compilada quanto interpretada. Arquivos fonte na linguagem Java são processados por um compilador, que gera um código denominado *byte-code*, que não é código de máquina nativo, porém algo similar, independente de plataforma. A execução de um programa (ou mais especificamente, de uma classe) Java envolve a interpretação desse código em uma máquina virtual (chamada JVM – Java Virtual Machine) que faz a transformação final desse código para o código de máquina nativo de um ambiente computacional.

A razão pela qual a linguagem Java implementa esse processo é permitir a distribuição de programas em forma binária que possam ser executados no maior número possível de plataformas. Como é a JVM que faz a transformação final entre o *byte-code* e o código nativo, apenas esse software precisa ser implementado em linguagens de programação tradicionais como C.

Todo código de um *software* escrito em linguagem Java deve estar contido em uma estrutura lógica denominada de Classe. Uma classe contém atributos (informações) e métodos (operações realizadas sobre esses atributos) e pode ser entendida como a especificação das características e comportamentos dos objetos (instâncias) dessa classe. O processo de transformação de uma classe para um objeto é denominado instanciação e é controlado por uma entidade especial, chamada *Classloader*.

4.8 NÓS DE FORMAS GEOMÉTRICAS

A representação visual de cenários tridimensionais dentro do padrão X3D é feita através dos nós de formas geométricas (*geometric objects*).

Todo conjunto de objetos que apresenta algum resultado visual precisa utilizar dois nós especiais, chamados *Appearance* e *Shape* dentro de um cenário X3D. O nó *Appearance* define características como o material, a cor e a textura das formas geométricas. Já o nó *Shape* é utilizado para conectar um nó *Appearance* e um conjunto de nós geométricos. Entre esses nós incluem-se primitivas como cubos (nó *Box*) e esferas (nó *Sphere*) e um nó denominado *IndexedFaceSet*. Este nó é responsável por representar conjuntos arbitrários de vértices e polígonos, possibilitando a inserção de modelos personalizados (como casas, castelos, móveis, etc) em cenas.

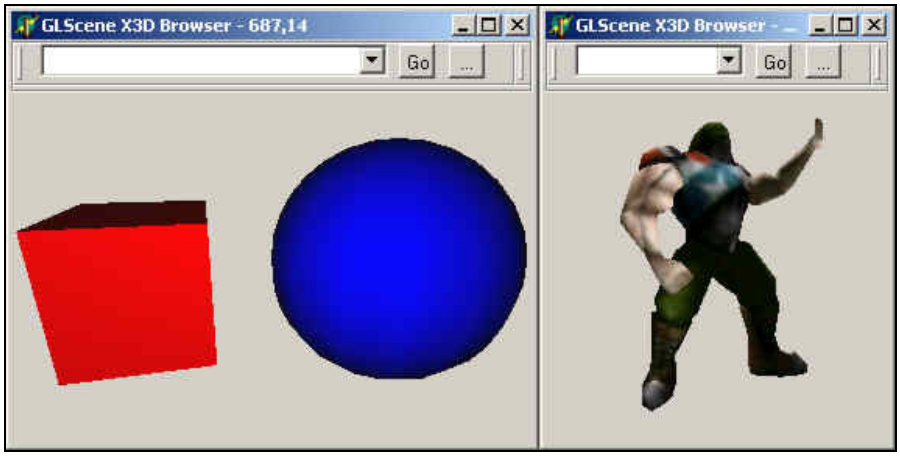

Figura 13 – Exemplo de nós *Box*, *Sphere* e *IndexedFaceSet*

A Figura 13 mostra dois arquivos diferentes: o primeiro contendo dois nós geométricos (*Box* e *Sphere*) e o segundo contendo um nó *IndexedFaceSet* com um conjunto de polígonos que recriam um modelo tridimensional com formato humanóide.

5 DESENVOLVIMENTO

Os principais conceitos envolvidos na implementação deste projeto serão detalhados nas próximas seções. Em primeiro lugar, será analisado o conjunto de nós X3D que permitem a sincronização dos objetos em ambientes virtuais multiusuário, chamados *NetNodes*. Serão oferecidas informações sobre a arquitetura geral do código e de cada protocolo implementado.

Em seguida, será analisado o protótipo de *browser* X3D construído para este projeto, denominado GLX3D. Serão apresentadas as razões para seu desenvolvimento, bem como informações sobre sua estrutura interna e alguns dos principais subsistemas necessários para a visualização de arquivos X3D.

5.1 DEFINIÇÕES

Antes de apresentar os detalhes sobre a implementação do projeto, faz-se necessário o estabelecimento de alguns termos que possam ser utilizados de maneira clara e objetiva nas próximas seções.

Em primeiro lugar, definiremos um ambiente virtual como um conjunto de arquivos X3D logicamente relacionados através de nós *Inline* e/ou *ExternProtos*,

cujo conteúdo pode ser compartilhado entre diversos *browsers* conectados através de uma rede de computadores e que estão semanticamente relacionados de alguma maneira. A expressão “semanticamente relacionados” se refere às características visuais e comportamentais do ambiente virtual, e pode ser informalmente entendido como “alguma relação temática, estabelecida pelo autor dos arquivos”.

Definiremos como usuário um *browser* participante de um ambiente virtual, seja ele controlado por um operador humano ou um processo automático. Um usuário é capaz de iniciar eventos no ambiente virtual através dos nós sensores contidos nos arquivos que compõem tal ambiente ou de código externo, contido em nós script.

Definiremos como sincronização (de objetos) os processos que permitem o compartilhamento das informações dos eventos X3D (propagados localmente através das rotas) entre os usuários do ambiente virtual com o objetivo de manter a consistência visual e comportamental deste ambiente.

Definiremos como um protocolo de comunicação um conjunto de regras que ditam como dois ou mais usuários compartilham informações através da rede de computadores subjacente.

Definiremos como uma mensagem (ou pacote) um conjunto de informações estruturadas que é trocada entre dois usuários, seguindo-se as especificações de um protocolo em particular.

5.2 SISTEMA DE NETNODES

A principal contribuição deste trabalho para o estado da arte em DVEs foi o estabelecimento de um sistema que permita a sincronização de ambientes virtuais baseado no formato de arquivo X3D através de redes de computadores conectadas pelo protocolo TCP/IP.

Para este fim, foi estabelecido um conjunto de nós X3D que interagem com a rede de comunicações subjacente, enviando e recebendo as informações dos eventos ocorridos em um determinado *browser*. Esse conjunto de nós é chamado genericamente de NetNodes, e o sistema que permite a troca de informações, Sistema de NetNodes.

O Sistema de NetNodes realiza seu trabalho através do envio e recebimento de mensagens entre os usuários do ambiente virtual. Essas mensagens representam eventos ocorridos dentro dos *browsers* X3D utilizados pelos usuários e que são roteados através do grafo de comportamentos de um (ou mais) dos arquivos que compõem o ambiente virtual.

A maior contribuição para o sistema de NetNodes foi obtida do trabalho de Araki (2005), que foi estendido para permitir a adição de protocolos arbitrários e criação dinâmica de nós em ambientes virtuais baseados no padrão X3D.

A seguir, serão apresentados detalhes sobre o funcionamento desse sistema e os diversos protocolos implementados. A especificação do sistema de NetNodes proposta para inclusão no padrão X3D está contida no Apêndice A. Já a especificação do conjunto inicial de protocolos pode ser encontrada no Apêndice B.

5.2.1 Arquitetura do sistema

O Sistema de NetNodes funciona através de dois conjuntos de nós X3D: nós que representam emissores individuais de eventos (chamados também de NetNodes), e nós gerenciadores (NetNodeManagers), que representam os protocolos utilizados na troca de mensagens entre os usuários do ambiente virtual.

O código para esse sistema foi escrito em linguagem Java preparado para inclusão em nós *Script* utilizando a *Scene Access Interface*. A implementação através da dupla Java/SAI foi escolhida por possibilitar fácil extensibilidade dos *browsers* X3D com código arbitrário. O acesso à grande quantidade de bibliotecas Java disponíveis (em especial às bibliotecas de rede, que não podem ser encontradas na linguagem ECMAScript) foi essencial para o desenvolvimento do sistema de NetNodes. Além disso, durante os estágios iniciais do projeto (enquanto o *browser* GLX3D não havia sido desenvolvido) o *browser* Xj3D (2005) foi usado. Quando suporte ao Java/SAI foi adicionado ao GLX3D, o código do sistema de NetNodes pôde ser movido para o novo *browser* sem qualquer mudança.

No sistema desenvolvido por Araki (2005) os NetNodes são responsáveis pela comunicação direta com a rede, o que não acontece no sistema proposto acima. Portanto, é preciso haver algum mecanismo de comunicação entre NetNodes e NetNodeManagers para que os eventos sejam corretamente sincronizados entre os usuários.

A especificação do sistema de NetNodes (Apêndice A) não dita a maneira pela qual os NetNodes e os NetNodeManagers trocam informações entre si. Na implementação desenvolvida, essa comunicação foi feita através da troca de mensagens de rede realizadas entre interfaces *loopback* (ou seja, cujo envio e

recebimento acontecem na mesma máquina). Tanto NetNodes quanto NetNodeManagers contém *sockets* de comunicação local, onde são enviados e recebidos datagramas UDP contendo os eventos individuais de um par NetNode/NetNodeManager.

Esse modelo de comunicação foi escolhido para superar um problema encontrado no *browser* Xj3D. Como o código Java de nós *scripts* são carregados em *classloaders* diferentes, eles não podem trocar informações através dos mecanismos usuais dessa linguagem (em essência, objetos, métodos, etc). Por essa razão, cada NetNode e NetNodeManager aloca um canal de comunicação UDP bidirecional.

O processo de sincronização de um ambiente virtual consiste, portanto, na transmissão de informação de um NetNode inicial, para um conjunto de NetNodeManagers que enviam uma (ou mais) mensagens através da rede de comunicação. Essas mensagens são recebidas pelo NetNodeManager em um usuário remoto que envia o evento para o NetNode local correspondente. A Figura 14 ilustra o processo de sincronização de dois usuários (*Host 1* e *Host 2*) que compartilham o evento E (iniciado pelo nó A no *Host 1*) através da rede de comunicação.

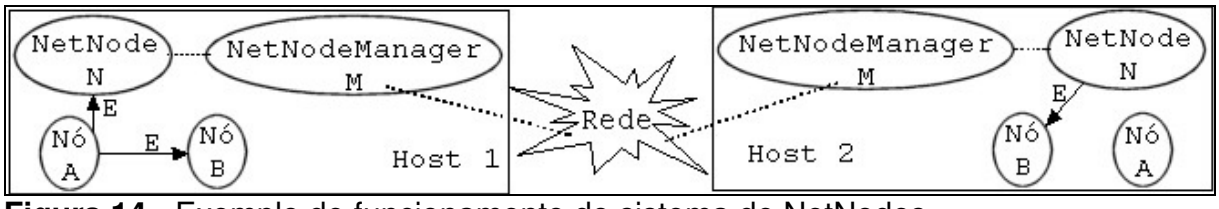

Figura 14 - Exemplo de funcionamento do sistema de NetNodes

5.2.2 NetNodes de campos

Os NetNodes são os indicadores individuais de eventos dentro do ambiente virtual. Para cada evento que se deseja sincronizar através da rede (por exemplo, para cada campo de um nó X3D que deve se manter consistente nos usuários do ambiente) deve existir um NetNode correspondente. Existe um tipo de NetNode para cada tipo de campo da especificação X3D (ou seja, o tipo *SFBool* tem um nó correspondente *SFBoolNetNode*, SFVec3f corresponde à SFVec3fNetNode, e assim por diante). As únicas exceções a essa regra são os tipos *SFNode* e *MFNode* que não possuem NetNodes correspondentes, uma vez que eles correspondem apenas à referências internas, que têm pouco significado por si só. A Figura 15 mostra a especificação do tipo abstrato X3DNetNode, que é o ancestral de todos os tipos concretos (como *SFBoolNetNode*) de NetNodes.

```
X3DNetNode : X3DNode {
  SFNode    [in,out] metadata NULL [X3DMetadataObject]
  SFBool    [in,out] active TRUE
  MFNode    [in,out] netNodeManagers []
  SFString  []       identification ""

  # One of each:
  fieldType [in]     set_value
  fieldType [Out]    value_changed
}
```

Figura 15 - Especificação abstrata dos NetNodes

O propósito de um NetNode é inserir-se em uma cascata de eventos do grafo de comportamentos dos arquivo X3D que compõem o mundo virtual, propagando os eventos de e para NetNodeManagers. Em outras palavras, um NetNode é usado para indicar quais eventos serão sincronizados dentro do ambiente virtual, e quais protocolos serão utilizados para sincronizar esses eventos.

Todos os nós que representam NetNodes contém três campos básicos: *active*, *netNodeManagers* e *identification*.

O campo *active* é utilizado para indicar se o NetNode em particular está enviando e recebendo eventos dos NetNodeManagers ao qual está associado. Caso seu valor seja falso, nenhum evento será trocado entre este NetNode, os NetNodeManagers cadastrados e o grafo de comportamentos ao qual este nó pertence.

O campo *netNodeManagers* indica quais NetNodeManagers (em outras palavras, quais protocolos) devem ser usados para efetuar a sincronização entre os diversos usuários do ambiente virtual.

O campo *identification* é utilizado como identificador único para um NetNode. Para que o sistema de NetNodes funcione corretamente, não podem haver dois nós desse tipo com o mesmo valor para este campo.

Os campos set_*value* e *value*_changed têm o tipo compatível com o NetNode particular ao qual pertencem (ou seja, o nó *SFBoolNetNode* implementa esses campos com o tipo *SFBool*, o nó *SFVec3fNetNode* como *SFVec3f* e assim por diante). O primeiro campo é utilizado para iniciar um evento de sincronização; sempre que um valor é escrito nesse campo, o NetNode deve notificar os NetNodeMangers cadastrados de que um novo evento deve ser enviado para os usuários remotos. Já o segundo campo é utilizado para saída de eventos, sendo acionado quando uma mensagem de sincronização é recebida por um dos NetNodeManagers.

No código Java, a classe básica utilizada na implementação dos NetNodes de campos foi *d3net.externimpl.BaseNetNode*. Os NetNodes propriamente ditos foram implementados no mesmo pacote (*d3net.externimpl*)

seguindo a nomenclatura apresentada anteriormente (*d3net.externimpl.SFBoolNetNode* para o tipo *SFBoolNetNode* e assim por diante).

5.2.3 Protocolo HOTP

Protocolos de comunicação ponto-a-ponto sofrem de um empecilho para a realização de seus trabalhos que pode ser denominado de “problema de descoberta” (*discovery problem*). Esse problema diz respeito à como um usuário do ambiente virtual é capaz de encontrar (ou “descobrir”) os endereços (dentro do contexto de protocolos IP, isso significa o endereço IP mais a porta de comunicação) dos outros participantes do sistema.

Durante a realização deste projeto, foi definido o protocolo *HTTP Occlusion Table Protocol* (Protocolo de Tabelas de Oclusão HTTP - HOTP), responsável por armazenar os endereços dos participantes de um ambiente virtual. Esse protocolo trabalha em conjunto com o protocolo HTTP e servidores *web* convencionais.

Uma tabela de oclusão é definida como um arquivo texto onde cada linha pode estar em branco, conter um comentário (caso ela seja iniciada pelo caractere “#”) ou ser o endereço de um participante do ambiente virtual. Qualquer usuário que deseje participar desse ambiente e que utilize um protocolo que declare implementar o HOTP deve obter o conteúdo dessa tabela através de uma requisição HTTP (em essência, requisitar o recurso HTTP que representa a tabela de oclusão). Nesse momento, o software gerenciador da HOTP (denominado servidor HOTP)

deve ser capaz de adicionar o endereço do usuário que requisitou a tabela, registrando-o para futuros participantes do ambiente virtual.

A implementação do servidor das tabelas do protocolo HOTP foi feita na linguagem PHP (2005), por ser uma linguagem largamente presente em soluções de hospedagem de *websites*. Essa escolha permite uma ágil instalação de um ambiente virtual multiusuário relacionado à temática de um *website* específico.

Já a parte cliente do protocolo HOTP foi implementada em linguagem Java, nas classes *d3net.net.HOTPAddressCollector* (uma interface utilizada para definir os métodos básicos necessários em um coletor de endereços que se comunica com uma tabela de oclusão), *d3net.net.BaseHOTPImpl* (responsável pela comunicação com o servidor através do protocolo HTTP e que implementa a interface *HOTPAddressCollector*) e *d3net.net.HOTPListener* (que deve ser implementado nas classes que desejam executar processamento quando o endereço de um usuário remoto for recebido).

5.2.4 Protocolo SPCP

O primeiro protocolo implementado através do sistema de NetNodes foi chamado de *Simple Peer Consistency Protocol* (Protocolo de Consistência Simples de Pares - SPCP). O objetivo desse protocolo é permitir a sincronização de eventos ocorridos dentro do grafo de comportamentos de um ambiente virtual da maneira mais simples e direta possível, sem muitas considerações sobre sua escalabilidade.

Este foi o primeiro protocolo implementado e se destina a demonstrar a viabilidade de construção de DVEs de acordo com a arquitetura de NetNodes.

O protocolo SPCP utiliza o protocolo de transporte UDP para envio de mensagens e uma arquitetura de rede ponto-a-ponto, com todos os usuários do ambiente virtual trocando mensagens entre si sem a intermediação de um servidor comum. Na prática, isso significa que em um ambiente virtual com N usuários, todo usuário deverá utilizar suficiente largura de banda para enviar uma mensagem à N-1 usuários. Caso um protocolo cliente-servidor fosse utilizado, cada evento gerado por um usuário provocaria o gasto de largura de banda de exatamente uma mensagem (destinada ao servidor). O servidor, no entanto, teria de arcar com o gasto das (N-1) mensagens restantes.

A escolha de um protocolo ponto-a-ponto ao invés de um cliente-servidor se deu pela menor complexidade envolvida em sua construção e por esse tipo de protocolo dispensar qualquer tipo de software extra (no caso do de um protoco C/S haveria a necessidade de um software servidor ser constantemente executado para que os usuários pudessem trocar mensagens entre si). O protocolo de transporte UDP foi escolhido pela mesma razão (simplicidade) e por não fazer quaisquer garantias com relação à entrega e ordenação dos pacotes (o que pode ser inconveniente quando a taxa de mensagens enviadas é relativamente alta).

O protocolo SPCP é representado em arquivos X3D através do nó SPCPNetNodeManager, tendo como valor do campo protocol o texto “SPCP”. A sua implementação Java foi feita na classe *d3net.externimpl.spcp.SPCPNetNodeManager*, onde o método *syncUpdatePacket* é implementado da maneira apropriada.

Esse nó (*SPCPNetNodeManager*) contém apenas um novo campo (não presente em outros NetNodeManagers) que é o campo *tableURLs*. Esse campo indica o endereço das tabelas de oclusão utilizadas com o protocolo HOTP, que é usado para a descoberta dos usuários participantes do sistema.

O protocolo SPCP deve ser encarado como prova-de-conceito do sistema de NetNodes, com utilização (prática) restrita à redes locais. Protocolos mais avançados para o tipo de trabalho executado pelo SPCP são uma possível fonte para futuras pesquisas.

5.2.5 Protocolo SLCSP

O protocolo SPCP é suficiente para sincronizar usuários distribuídos através de uma rede de longa distância (como a internet) enquanto eles utilizam o ambiente virtual. No entanto, esse protocolo não faz qualquer referência à como essas informações devem ser armazenadas para futuros usuários do sistema. Caso o ambiente virtual sofra alterações (por exemplo, nas posições dos diversos objetos), novos usuários que visitem esse ambiente encontrarão um estado diferente daquele estabelecido pelos usuários anteriores. A essa questão dá-se o nome de problema dos atrasados (*late-comer problem*).

A solução para o problema dos atrasados consiste em armazenar de alguma maneira os valores dos eventos iniciados por usuários do ambiente virtual e disponibilizar esses valores para novos usuários do sistema.

Para isso foi desenvolvido o protocolo *Simple Late-Comer Synchronization Protocol* (Protocolo Simples de Sincronização de Atrasados – SLCSP). Esse protocolo implementa um novo NetNodeManager (denominado *SLCSPNetNodeManager*) que, ao receber um evento de um NetNode, envia-o à um servidor *web*, responsável por armazenar a identificação do NetNode e o valor desse evento para futuras consultas.

O protocolo SLCSP funciona sobre o protocolo HTTP, enviando uma requisição POST em intervalos de tempo determinados, agregando todos os eventos ocorridos no ambiente virtual. Esses eventos são armazenados em um arquivo texto denominado tabela SLCSP através de um script escrito em linguagem PHP (2005).

No momento em que um novo usuário deseja conectar-se a um ambiente virtual que utiliza o protocolo SLCSP, uma requisição é feita ao servidor, que retorna a tabela contendo o último valor dos eventos armazenados. Essa tabela é então processada, e um evento é simulado no *browser* do novo usuário, fazendo com que o cenário tridimensional adquira as características que foram produzidas pelas mudanças cumulativas dos usuários anteriores.

5.2.6 Protocolo SPP

Embora o protocolo SLCSP resolva o problema dos atrasados e o protocolo SPCP permita sincronização dos usuários *online* em um determinado momento, uma última funcionalidade foi implementada no sistema: adição dinâmica de nós X3D em ambientes virtuais.

Essa funcionalidade é importante especialmente para os sistemas que simulam comunidades virtuais, onde os usuários do ambiente podem adicionar novos objetos para construção do cenário tridimensional.

Para incluir essa nova função, foi criado um novo protocolo denominado *Simple Persistency Protocol* (Protocolo Simples de Persistência – SPP).

Esse protocolo utiliza (assim como o SLCSP) servidores *web* e requisições HTTP para realizar seu trabalho. Ele também utiliza um arquivo X3D principal (denominado *Master World File* - MWF) que contém referências a um conjunto de arquivos de objeto (*object files*).

Duas operações que podem ser executadas sobre esse conjunto de arquivos também são definidas: *uploadFile* e *createObject*.

A operação *uploadFile* é responsável pelo envio de arquivos para o servidor SPP. Todo tipo de arquivo é aceito e nenhuma restrição de segurança foi implementada neste protocolo. Essa operação utiliza o protocolo HTTP para transferência de informações, codificando o arquivo sendo enviado em uma requisição do tipo POST.

Já a operação *createObject* permite a inserção de novos nós X3D no *Master World File*. Ao executar essa operação, um usuário deve especificar o nome do *Object File* que será incluído no ambiente virtual através de um nó *Inline* contido no MWF. O resultado dessa operação (que é retornada no documento gerado pela requisição HTTP que a especificou) contém um conjunto de nós X3D que deve ser adicionado à instância local do *browser* para que o ambiente se mantenha consistente. Esse conjunto de nós também pode ser sincronizado através da rede (utilizando o protocolo SPCP, por exemplo) para permitir visualização em tempo real da construção do cenário tridimensional.

O protocolo SPP não realiza qualquer tipo de verificação de segurança ou validação do conteúdo dos arquivos de objeto enviados. Essas considerações poderão ser incluídas em futuras versões desse sistema e representam outra possível fonte de pesquisa.

5.2.4 Diferenças entre o sistema proposto e o sistema de Araki

Tanto no sistema de Araki, quanto nas extensões propostas neste trabalho, existe uma etapa intermediária de comunicação entre os NetNodes em diferentes usuários. No sistema de Araki, um processo denominado Con*sistency Manager* localizado em outro ponto da rede é responsável por receber todas as mensagens enviadas por um NetNode e repassá-las aos outros usuários. Esse mecanismo torna a arquitetura de comunicações do sistema de NetNodes implicitamente Cliente-Servidor, uma vez que o único ponto de contato entre os usuários é um intermediário localizado em outro ponto da rede. A Figura 16 ilustra esse processo.

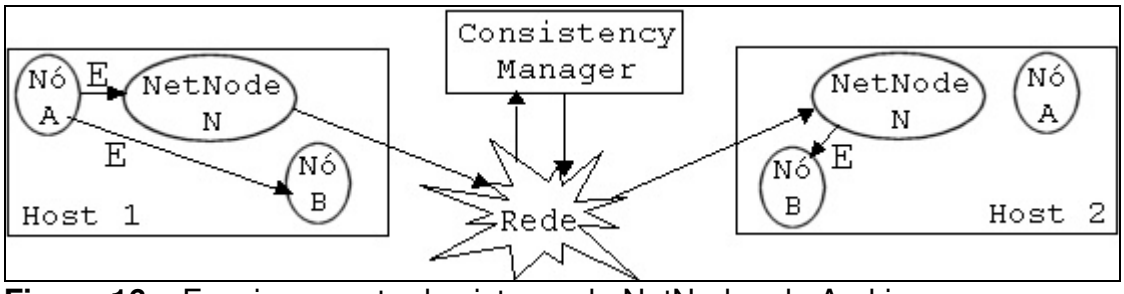

Figura 16 – Funcionamento do sistema de NetNodes de Araki

Já na proposta mostrada neste trabalho, o *Consistency Manager* é eliminado em favor de um outro nó denominado NetNodeManager. Como visto

anteriormente, um NetNodeManager implementa algum protocolo de comunicação, cujas características não são conhecidas *a priori*.É importante ressaltar que o uso de NetNodeManagers não impede a utilização de algum tipo de servidor. Essa questão, no entanto, é deixada a cargo dos protocolos individuais e não do modelo abstrato de sincronização em si.

Outra diferença entre os dois trabalhos pode ser encontrada na definição de um NetNode feita por Araki. Sua definição contém mais campos e carrega maior funcionalidade, fato ocasionado pela implementação, dentro do próprio NetNode, de toda funcionalidade implícita por seu protocolo de comunicação único. A Figura 17 mostra a especificação de ambos modelos, tornando fácil a visualização das diferenças.

```
X3DNetNode : X3DNode {
  SFNode    [in,out] metadata NULL [X3DMetadataObject]
  SFBool    [in,out] active TRUE
  MFNode    [in,out] netNodeManagers []
  SFString  []       identification ""

  # One of each:
  fieldType [in]     set_value
  fieldType [Out]    value_changed
}
```

```
NetXXX : X3DNode {
  SFString   [] purpose
  SFInt32    [] capacity
  SFBool     [in] request_right
  SFBool     [out] hasRight
  SFTime     [in,out] waitTime
  SFString   [] frequency
  SFString   [] debug

  fieldType  [in] request_operate
  fieldType  [out] value_caught
}
```

Figura 17 - Especificação abstrata de um NetNode usada neste trabalho e os NetNodes de Araki

Os campos *request_operate* e *value_caught* têm significado semelhante aos campos *set_value* e *value_changed* (respectivamente) e são utilizados para iniciar o processo de sincronização e enviar eventos recebidos através da rede.

Os campos *capacity*, *request_right*, *hasRight* e *wait_time* controlam acesso de escrita ao evento *request_operate* do NetNode e correspondem à implementação de um algoritmo de semáforos, comumente encontrado em sistemas de IPC (*Inter Process Communication*, ou comunicação entre processos).

O campo *capacity* indica o número de *browsers* que podem efetuar uma mudança (ou seja, enviar eventos) ao mesmo tempo em um NetNode em particular.

O campo *request_right* é usado para obter acesso de escrita aos outros campos do NetNode. Quando um valor *true* (verdadeiro) é enviado para esse campo, *Consistency Manager* recebe uma mensagem que decrementa o semáforo correspondente (caso a capacidade total de usuários concorrentes não tenha sido atingida) e habilita o envio de eventos pelo *browser* requisitante. Um valor *false* (falso) enviado a esse campo, ao contrário, incrementa o semáforo liberando o NetNode para outros usuários.

O campo *wait_time* funciona como um intervalo de *timeout* para as requisições enviadas ao campo *request_right*. Já *has_right* é responsável por indicar se o *browser* foi bem sucedido na operação de requisição do semáforo (ou seja, se o NetNode está apto a enviar eventos pela rede).

O campo *debug* é utilizado para detecção de laços (*loops*), um possível problema introduzidos pelo sistema de NetNodes. Esse processo é causado por uma cascata de eventos onde um NetNode recebe eventos iniciados por outro NetNode, seja através de um laço no grafo de comportamentos ou por uma conexão direta entre dois NetNodes diferentes.

Já o campo *purpose* é usado pelo *Consistency Manager* armazena o propósito de um NetNode em particular. Dois valores podem ser encontrados nesse campo: TRIGGER ou CONSISTENCY. O primeiro valor indica que esse NetNode é usado para sincronizar animações e nenhum procedimento é feito durante sua inicialização. Já o segundo valor provoca um evento inicial de sincronização durante a inicialização do NetNode para que o estado atual do ambiente virtual (caso ele

tenha sido modificado por usuários anteriores) possa ser obtido do Consistency Manager.

Finalmente, o campo *frequency* indica a freqüência com que os eventos são sincronizados através da rede. Os dois valores possíveis para esse campo são DISCRETE e CONTINUOUS. Quando o primeiro tipo de sincronização é escolhido, todo evento gerado pelo NetNode é enviado ao *Consistency Manager*. Já no segundo tipo, os eventos são sincronizados com uma taxa constante de atualização (o que é útil em cascatas geradas por *TimeSensors*, por exemplo).

Os campos *purpose*, *capacity*, *request_right*, *has_right*, *waitTime*, *frequency* e *debug* não existem na nova definição proposta para os NetNodes. Protocolos que desejem implementar essas funcionalidades (acesso compartilhado, eventos contínuos, sincronização inicial) devem ser especificados de maneira independente ao sistema de NetNodes. Isso torna possível reduzir a complexidade da implementação desses nós, ao mesmo tempo em que se permite incluir (e caso seja necessário, substituir) características individuais de seu comportamento.

5.3 PROTÓTIPO DE *BROWSER* X3D (GLX3D)

Na primeira fase da implementação do sistema de NetNodes foi utilizado o *browser* Xj3D (2005) para a visualização dos ambientes virtuais. No entanto, durante essa fase alguns problemas foram encontrados que impossibilitaram a implementação do protocolo SPP. Em particular, a adição dinâmica de nós nos grafos de cena não era uma operação suportada pela versão

da SAI implementada nesse *browser*. A Figura 18 mostra uma cena simples sendo executada dentro no *browser* Xj3D.

Figura 18 – Interface do *browser* Xj3D

Visando completar o sistema de NetNodes, foi desenvolvido um *browser* experimental denominado GLX3D, com suporte aos nós básicos do padrão X3D e às operações da SAI que possibilitam a criação dinâmica de ambientes virtuais.

A linguagem Delphi (Borland, 2005) foi escolhida para implementação, por possibilitar rápido ciclo de desenvolvimento (a versão inicial do *browser* com suporte a SAI foi desenvolvida em duas semanas) e por já contar com ferramentas de alto nível destinadas às tarefas necessárias de um *browser* X3D (em particular, representação de cenas tridimensionais).

Para a visualização dos ambientes foi utilizada a biblioteca GLScene (2005), destinada à criação de aplicações tridimensionais. A GLScene vêm sendo desenvolvida em regime de código aberto à 6 anos e conta com uma ampla gama de componentes que permitem rápido desenvolvimento de cenários tridimensionais.

Já a comunicação de rede do *browser* foi realizada com a biblioteca Indy (2005), também de código aberto e com grande aceitação pela comunidade de programadores Delphi (tendo sido, inclusive, incluída em versões mais novas desse programa). A Figura 19 mostra uma cena simples sendo executada no *browser* GLX3D.

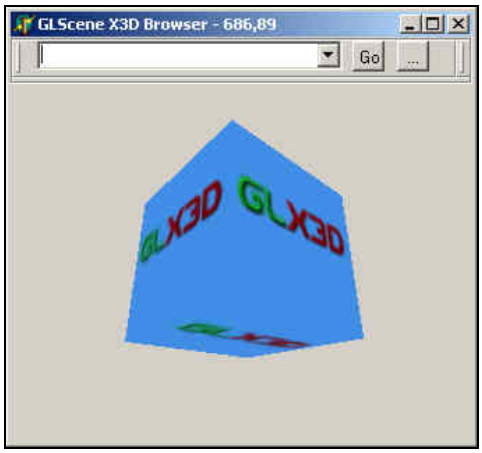

Figura 19 – Interface do *browser* GLX3D

5.3.1 Arquitetura do sistema

A arquitetura do código do *browser* GLX3D foi criada a partir das operações definidas na especificação da SAI do padrão X3D, sofrendo grande influência da especificação das ligações entre a linguagem Java e a SAI (WEB 3D CONSORTIUM, 2005). A razão dessa arquitetura foi facilitar a interface entre o código Delphi do *browser* e código externo da SAI.

Os diversos componentes X3D foram implementados dentro do *browser* GLX3D em diferentes classes, representando diferentes nós. A classe base

para todas as implementações de nós X3D é a classe *TGLXNodeHandler*, que contém os principais atributos e métodos que permitem a visualização dos arquivos.

Já os tipos de campos (*SFBool, SFString*) foram implementados como classes derivadas de *TGLXX3DField*, sendo seus principais métodos *setValue* e *getValue*, que para cada tipo de campo utilizam parâmetros de entrada e saída diferentes (por exemplo, a implementação do tipo X3D *SFBool* na classe *TGLXSFBool* utiliza métodos com o tipo nativo *boolean*).

5.3.2 Processamento de arquivos X3D

Na versão atual, o *browser* GLX3D suporta apenas a codificação XML dos arquivos X3D. Ao ser carregado do sistema de arquivos (ou lido através de uma requisição HTTP), uma representação DOM do arquivo X3D é criada. Essa representação consiste em um conjunto hierárquico de objetos, onde cada item contém as informações sobre um elemento XML, tendo sido utilizada a biblioteca de código aberto SimpleXML (Vlaslov, 2005) para realizar essa conversão. O segundo passo é processar os elementos um a um, dando origem aos objetos que realizam a renderização da cena tridimensional.

Todo o processo de conversão é feito no método abstrato *handleNode* da classe *TGLXNodeHandler*, sendo implementado em cada classe que representa um tipo diferente de nó da especificação X3D. Nesse método, os campos de um nó são lidos da representação DOM, os valores padrão são inicializados e, caso o nó

em particular tenha uma representação visual (como o nó Box que representa um cubo tridimensional) o objeto correspondente da GLScene também é criado.

5.3.3 Interface com código Java através da SAI

Uma das funcionalidades requeridas pelo *browser* GLX3D para a conclusão deste projeto era a possibilidade de inclusão de código adicional através da SAI, mais especificamente, de código Java acessado pela SAI.

A interação entre código Delphi e código Java é feito através da chamada *Java Native Interface* (JNI). A JNI é uma tecnologia que possibilita a chamada de código Java a partir de programas comuns (chamados de programas nativos) e chamada de código nativo a partir de classes Java. A conversão JNI-JEDI (MEAD, 2005) foi utilizada para integrar as chamadas JNI ao Delphi.

No código Delphi, a classe *TGLXScript* implementa o nó *Script*, e é responsável por obter os arquivos necessários para executar o código externo. Já a classe *TGLXJavaSaiHandler* é a responsável por implementar a ligação propriamente dita, instanciando a classe principal do script e inicializando os valores necessários.

Já no código Java, uma classe chamada *org.glx3d.EntryPoint* foi criada, responsável por receber e enviar eventos do código Delphi, permitindo troca bidirecional de informações.

Uma DLL (chamada dji.dll) também foi criada para permitir a troca de informações entre o código Java e o código Delphi. Sua única função é redirecionar

chamadas ao método query4message feitas em objetos da classe *org.glx3d.EntryPoint* para o *browser*.

5.4 APLICAÇÃO DESENVOLVIDA

Visando melhor demonstrar a viabilidade do sistema de NetNodes, uma aplicação baseada no conceito de comunidades virtuais foi criada. Essa aplicação consiste em um ambiente virtual constituído por um conjunto de arquivos X3D, onde a posição e orientação dos objetos visuais presentes podem ser livremente alteradas pelos usuários. As alterações efetuadas no cenário são sincronizadas através dos participantes e armazenadas em um servidor web para futuros usuários. Existe também a possibilidade de inserção de novos objetos e o envio de arquivos para o servidor, o que permite alteração dinâmica do ambiente.

O protocolo SPCP é utilizado para sincronização das alterações feitas no ambiente virtual através dos usuários presentes em um determinado momento. O protocolo SLCSP é usado para armazenamento dos eventos em um servidor *web*, e o protocolo SPP é utilizado para envio de arquivos e inclusão de objetos no cenário.

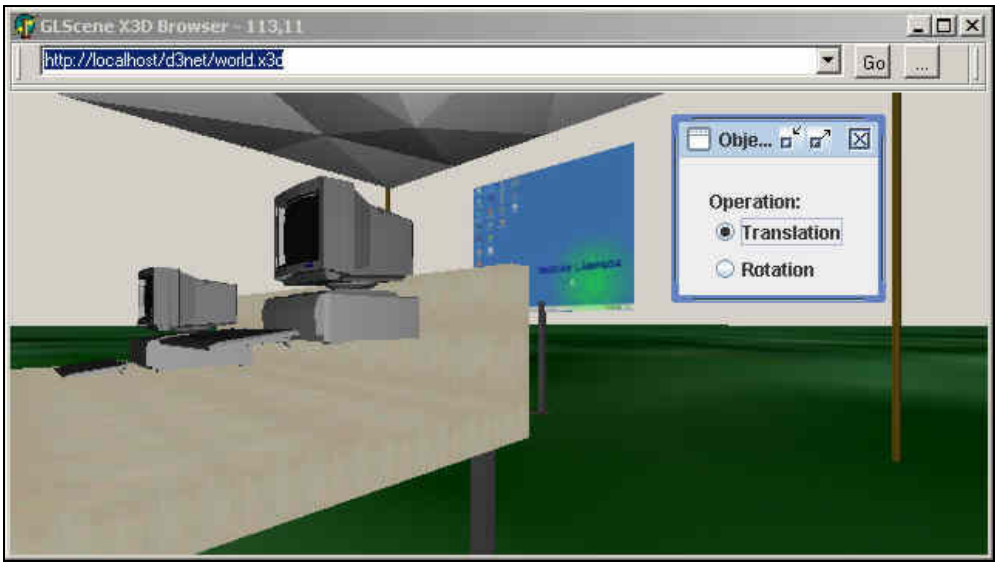

Figura 20 – Cenário da aplicação, mostrando a manipulação de um objeto (computador)

Os usuários do ambiente virtual podem mover-se livremente, visualizando os diversos objetos contidos no cenário. Para manipular um objeto (movê-lo ou rotacioná-lo) basta efetuar um clique sobre sua geometria. Uma tela contendo as operações que podem ser executadas é mostrada e o usuário pode, então, efetuar as alterações desejadas. Como toda movimentação de um objeto é sincronizada, qualquer participante do ambiente virtual verá imediatamente as alterações. A Figura 20 mostra a janela de operações quando uma translação está sendo executada sobre um objeto (neste exemplo, um computador).

A operação de envio (*upload*) de arquivos para os servidor é feito através de um objeto especial, denominado *uploadBox*. Ele pode ser encontrado no centro do ambiente virtual, e ao contrário dos outros objetos não pode ser movimentado. Ao ser acionado, uma janela padrão de escolha de arquivos é aberta e o arquivo selecionado é enviado (exceto se o usuário cancelar essa operação, fechando a janela de escolha através da opção "*cancel*"). A Figura 21 mostra a janela de seleção de arquivos, junto ao *uploadBox*.

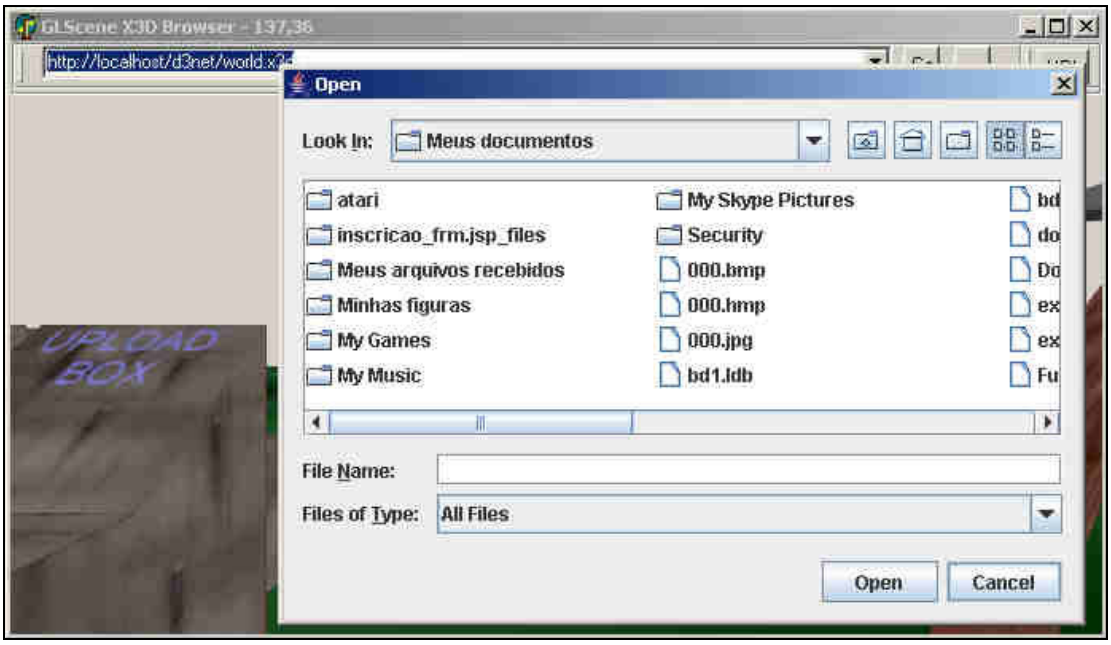

Figura 21 – Janela para seleção de arquivos

A criação de objetos também é executada de maneira semelhante, através de um objeto denominado *createBox*. Ao ser ativado, uma janela pedindo o nome do arquivo do objeto que será criado é pedido. Quando fechada, o processo de inserção de um novo nó *Inline* é iniciado da maneira especificada no protocolo SPP.

A última funcionalidade inserida na aplicação foi a capacidade de bate-papo entre os participantes do ambiente virtual, com sincronização efetuada através do protocolo SPCP. A janela de bate-papo, que inicia o envio das linhas de texto, pode ser acionada através de outro objeto especial, chamado *chatBox*, e comporta duas seções distintas; primeira é a área onde as mensagens enviadas e recebidas são lidas; a segunda é a área de edição, onde insere-se o texto que será enviado aos outros usuários ou dois comandos especiais, desenvolvidos para aumentar as possibilidades de interação.

O primeiro comando (“/me”) é utilizado para enviar uma mensagem do tipo *emote*, onde o apelido da pessoa é enviado de maneira diferenciada

(normalmente utilizado para simular ações efetuadas por texto). Por exemplo, “/me sonha acordado” irá gerar a linha “Matheus sonha acordado” para um usuário cujo apelido seja “Matheus”.

Já o segundo comando (“/setnick”) é utilizado para trocar o apelido do usuário. Ao ser executado, uma mensagem é enviada aos usuários presentes, alertando sobre a troca efetuada. É preciso salientar que nenhuma provisão foi feita para evitar o uso de apelidos idênticos entre usuários diferentes.

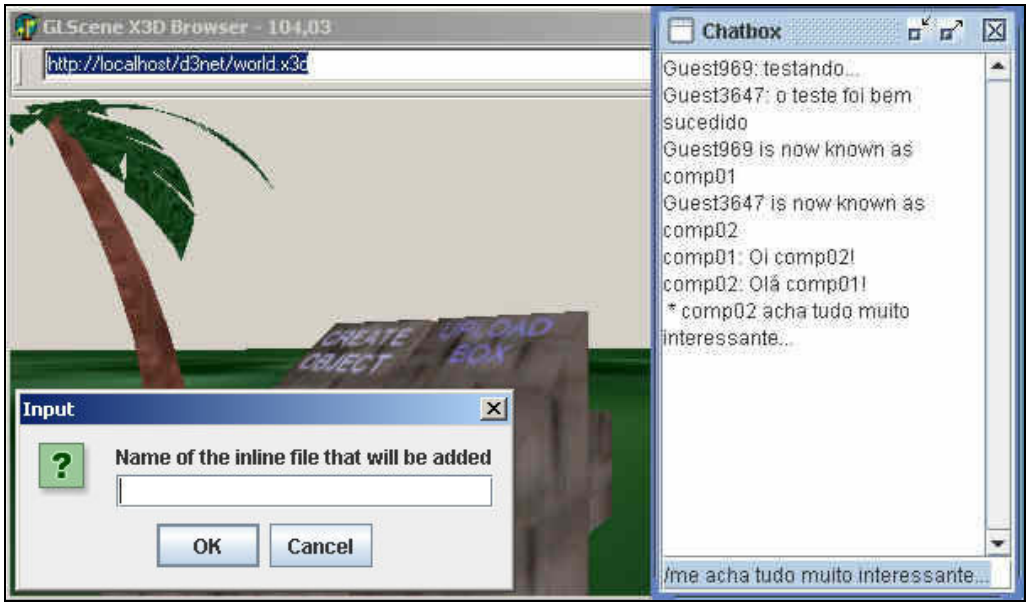

Figura 22 – *CreateBox*, *uploadBox*, janela de *Chat* e janela de criação de objetos

A Figura 22 mostra as caixas de criação de objeto (*CreateBox*) de envio de arquivos (*UploadBox*) e a janela de bate-papo. O objeto que ativa a janela de bate-papo (*ChatBox*) está localizado atrás das duas caixas.

5.5 USABILIDADE DO SISTEMA

Visando aferir a usabilidade da aplicação desenvolvida, um conjunto de entrevistas foi realizado. As entrevistas foram feitas com uma metodologia semelhante àquela empregada por Köykkä (et al, 2005): seis tarefas foram propostas a examinadores não familiarizados com o uso prático do ambiente virtual, porém com extenso conhecimento de informática e internet. Cinco entrevistas foram realizadas, em seções individuais e máquinas diferentes: Uma entrevista foi realizada em um computador com processador Pentium IV de 2,66GHz, 248MB de memória principal, adaptador de vídeo SiS 650 utilizando driver versão 5.1.2600.2180. As outras entrevistas foram realizadas em um Pentium IV 1.60GHz, 128MB de memória principal, adaptador de vídeo NVidia Vanta e driver versão 1.5.7.0. A razão da troca de computadores foi o melhor desempenho obtido no segundo (ocasionado pela presença de uma placa aceleradora 3D).

As seis tarefas propostas para realização foram (em devida ordem de execução):

a) Movimentação da câmera pelo cenário
b) Conversação através da janela de bate-papo
c) Envio de um arquivo para o servidor
d) Criação de um novo objeto
e) Posicionamento do objeto criado anteriormente
f) Giro do objeto criado anteriormente

O pequeno número de entrevistas dificulta a obtenção de informações concretas, no entanto dois comportamentos se destacaram durante a análise dos resultados: em primeiro lugar, durante a execução da tarefa número dois os

entrevistados tentaram ativar a janela de bate-papo através de um clique em um objeto semelhante a um computador. Já nas tarefas cinco e seis todos os entrevistados tentaram, de uma maneira ou outra, movimentar (ou girar) o objeto pressionando o botão do mouse sobre sua geometria e movimentando a câmera (através das setas do teclado), uma operação não prevista durante a criação da interface do usuário.

Ambas situações apontam para possíveis melhorias do sistema através de uma utilização mais acurada das metáforas da interface. A racionalização por trás da primeira situação (clique sobre um computador para acionar a janela de bate-papo) se deve à intrínseca conexão entre o computador e a internet (e, portanto, programas de bate-papo tradicionais). Já a segunda atitude (tentativa de “arrastar” o objeto, “segurando-o” com o mouse e movimentando-se) fornece outra pista sobre a percepção dos usuários das formas de interação com o cenário.

CONSIDERAÇÕES FINAIS

O desenvolvimento deste trabalho foi difícil, porém gratificante. O conhecimento adquirido durante sua realização vai além do estritamente técnico, necessário para a simples construção das aplicações descritas e envolveu participação em consórcios internacionais de padronização, estudo e implementação de especificações, e todo o processo envolvido na criação e modificação dessas especificações.

A principal contribuição para o estado da arte em tecnologias de Ambientes Virtuais Distribuídos foi o estabelecimento de um novo componente para o padrão X3D que permite a criação de ambientes virtuais dinâmicos. Esse componente foi formalizado em uma especificação, semelhante àquela do padrão internacional X3D e foi dividida em duas seções: uma parte abstrata, que define um conjunto de nós X3D usados no processo de sincronização de ambientes virtuais (denominados genericamente de NetNodes), e uma parte que define três protocolos simples para utilização com esse conjunto de nós.

Embora os protocolos especificados permitam construir uma aplicação baseada no conceito de comunidades virtuais, eles consistem apenas na primeira etapa para o desenvolvimento de uma solução completa para DVEs. A principal fonte de futuras pesquisas sugerida por este trabalho é a inclusão de protocolos mais avançados presentes na literatura (que incorporam elementos de

escalabilidade e segurança) e sua integração consistente com os diversos elementos do padrão X3D.

A inclusão do conjunto de nós criados para este projeto na especificação oficial do padrão X3D é outro desenvolvimento futuro, e será levado à discussão junto ao consórcio Web3D.

Paralelamente ao desenvolvimento do sistema de NetNodes foi criado um *browser* X3D, denominado GLX3D para a representação dos arquivos com codificação XML e com suporte à código externo Java. Essa aplicação foi desenvolvida para permitir a conclusão do sistema de NetNodes, uma vez que o único outro *browser* com suporte à código Java existente, chamado Xj3D, contém limitações que impediram sua completa utilização durante a realização deste projeto. Embora já conte com os recursos necessários para a representação de cenas tridimensionais simples, muito trabalho ainda deve ser feito para torná-lo completamente compatível com a especificação X3D, constituindo este um outro caminho para futuros trabalhos.

Para demonstrar a viabilidade do sistema de NetNodes, uma aplicação baseada no conceito de comunidades virtuais foi criada. Nela, os usuários podem criar dinamicamente o cenário virtual, que é armazenado para futuros acessos em um servidor web. Finalmente, um pequeno teste de usabilidade foi efetuado, onde duas possíveis melhorias do sistema, ambas relacionadas à metáfora criada para interface do usuário foram descobertas e são sugeridas para futuros sistemas de ambientes virtuais distribuídos.

REFERÊNCIAS

ACTIVEWORLDS. *Immersive virtual reality worlds (1997-2005)*. Disponível em: <http://www.activeworlds.com>. Acesso em: 09 abr. 2005.

ASHER, Mark. *2003 MMOs: The history of massively multiplayer online games*. Gamespy, 2003. Disponível em: <http://archive.gamespy.com/amdmmog/week4/>. Acesso em: 12 abr. 2005.

ARAKI, Yoshiaki. V*SPLUS: A High-level multi-user extension library for interactive VRML worlds*. In: Proceedings of the third symposium on Virtual reality modeling language, Monterey, 1998. Disponível em <http://doi.acm.org/10.1145/271897.274369>. Acesso em 10 mar. 2005.

BARRUS, J. W.; WATERS, R. C.; ANDERSON, D. B. *Locales and beacons:* eficient and precise support for large multi-user virtual environments. Massachussets: Mitsubishi Eletronic Research Laboratory, 1996. Disponível em: <http://www.merl.com/reports/docs/TR95-16a.pdf>. Acesso em: 26 fev. 2005.

BAUGHMAN, et al. *Cheat-proof playout for centralized and serverless online games*. Department of Computer Science, University of Massachusetts, Amherst, 2001. Disponível em: <http://prisms.cs.umass.edu/brian/pubs/baughman.infocom01.pdf>. Acesso em: 26 fev. 2005.

BILLINGHURST, M.; WEGHORST S.; FURNESS III T. *Shared space:* an augmented reality approach for computer supported collaborative work. University of Washington, Seattle, 1997. Disponível em: <http://www.hitl.washington.edu/publications/r-97-1/>. Acesso em: 09 abr. 2005.

BLUEHOSTING. *Servidor de hospedagem de páginas web (2003-2005)*. Disponível em: <http://www.bluehosting.com.br>. Acesso em: 12 abr. 2005.

BORLAND. *Delphi*. Disponível em: http://www.borland.com. Acesso em: 14 out. 2005.

BOURAS, Christos; FILOPOULOS, Alexandros. *Distributed virtual reality environments over web for distance education*. Bologna, 1998. Disponível em: <http://ru6.cti.gr/ Publications/269.pdf>. Acesso em: 09 mar. 2005.

BRUTZMAN, Don et al. *NPSNET-V: a new beginning for dynamically extensible virtual environments*, 2000. Department of Computer Science, Naval Postgraduate School, Monterey.

Disponível em: <http://www.npsnet.org/~npsnet/v/publications/ieeecga2000.pdf>. Acesso em: 26 fev 2005.

CHAPMAN, Nicholas. *Algorithms for a decentralised, multi-user 3D world*. 2004. Tese (Bachelor of Computer Science with Honors) - Victoria University of Wellington, Wellington. Disponível em: <http://203.96.147.59/~nickc/docs/Chapman04.pdf>. Acesso em: 09 mar. 2005.

CHURCHILL, Elizabeth F.; BLY, Sara. *Virtual environments at work:* ongoing use of MUDS in the Workplace. In: Proceedings of the international joint conference on Work activities coordination and collaboration, San Francisco, 1999. Disponível em: <http://doi.acm.org/10.1145/295665.295677>. Acesso em: 13 abr. 2005.

CIRCLEMUD. *CircleMud: multi user dungeon codebase*, 1992-2005. Disponível em: <http://www.circlemud.org/general.html>. Acesso em: 13 abr. 2005.

CRONIN, Eric; FILSTRUP, Burton; JAMIN, Sugih*; Cheat-proofing dead reckoned multiplayer games (Extended Abstract)*, Conference on Application and Development of Computer Games, Hong Kong, 2003. Disponível em: <http://warriors.eecs.umich.edu/games/papers/adcog03-cheat.pdf>. Acesso em: 26 fev. 2005.

DAHMANN, Judith S.; FUJIMOTO, Richard M.; WEATHERLY, Richard M. *The Department of Defense high level architecture*. In: Winter Simulation Conference, Atlanta, 1997. Disponível em: <http://www.cc.gatech.edu/computing/pads/PAPERS/DOD_High_Level_Arch.pdf>. Acesso em: 12 abr. 2005.

EVERQUEST. *Everquest*. Sony Online Entertainment, 1999-2005. Disponível em: <www.everquest.com>. Acesso em: 13 abr. 2005.

EVERQUEST2. *Everquest II*. Sony Online Entertainment, 2004-2005. Disponível em: <http://everquest2.station.sony.com/>. Acesso em: 12 abr. 2005.

FEIJÓ, Bruno; KOZOVITZ, Lauro Eduardo. *Arquiteturas para jogos massive multiplayer*. Departamento de Informática Puc-Rio, Rio de Janeiro, 2003. Disponível em: <ftp://ftp.inf.puc-rio.br/pub/docs/techreports/03_36_kozovits.pdf>. Acesso em: 26 fev. 2005.

FISCHER, William D. *Enhancing network communication in Npsnet-V virtual environments using xml-described dynamic behavior (DBP) protocols*, 2001. Tese (Master of Science in Computer Science) - Naval Postagraduate School, Monterey.

GIBSON, William. *Neuromancer*. 3. ed. Aleph, São Paulo, 2003.

GLSCENE. *OpenGL solution for Delphi*. Disponível em: <http://www.glscene.org>. Acesso em: 14 out. 2005.

GOSLING, James et al. *Java Language Specification*. 3. ed. Addison-Wesley Professional, Boston, 2005.

GREENHALGH, Chris. *Spatial scope and multicast in large virtual environments*. The University of Nottingham, Nottingham, 1996. Disponível em: <http://www.crg.cs.nott.ac.uk/research/publications/papers/NOTTCS-TR-96-7.pdf>. Acesso em: 09 abr 2005.

HAN, Seunghyun et al. *Scalable network support for 3D virtual shopping mall*. In: International Conference on Virtual Systems and Multimedia, Gyeongju, 2002. Disponível em: <http://cds.icu.ac.kr/research/area/dve/data/vsmm02.pdf>. Acesso em: 13 abr. 2005.

HARDT, James; WHITE, Kevin. *Distributed interactive simulation (DIS)*. University of Central Florida, 1998. Disponível em: <http://www-ece.engr.ucf.edu/~jza/classes/4781/DIS/project.html>. Acesso em: 12 abr. 2005.

HENDERSON Tristan Nicholas Hoang. *The effects of relative delay in networked games*. University of London, London 2003. Disponível em: <http://www.cs.dartmouth.edu/~tristan/pubs/thesis.pdf>. Acesso em: 11 abr. 2005.

HISTORYOFMUDS. *The history of MUDs:* part II. Gamespy, 2001. Disponível em: <http://archive.gamespy.com/articles/january01/muds1/index4.shtm>. Acesso em: 13 abr. 2005.

HU, Shun-Yun; LIAO, Guan-Ming. *Scalable peer-to-peer networked virtual environment*. Taiwan, 2004. Disponível em: <http://doi.acm.org/10.1145/1016540.1016552>. Acesso em: 09 mar. 2005.

INTERNATIONAL GAME DEVELOPERS ASSOCIATION. *2004 Persistent worlds whitepaper*. 2004. Disponível em: <http://www.igda.org/online/IGDA_PSW_Whitepaper_2004.pdf>. Acesso em: 26 fev. 2005.

ISO/IEC. *The virtual reality modeling language specification:* ISO/IEC 14772-1:1997, 1997. Disponível em: <http://www.web3d.org/x3d/specifications/vrml/ISO-IEC-14772-IS-VRML97WithAmendment1>. Acesso em: 26 fev. 2005.

KAPOLKA, Andrzej; MCGREGOR, Don; CAPPS, Michael. *A unified component framework for dynamically extensible virtual environments*. Proceedings of the Fourth ACM International Conference on Collaborative Virtual Environments, Bonn, 2002. Disponível em: <http://doi.acm.org/10.1145/571878.571889>. Acesso em: 11 abr. 2005.

KENT, Steven. Alternate Reality: *The history of massively multi-player online games*. Gamespy, 2003. Disponível em: <http://archive.gamespy.com/amdmmog/week1/>. Acesso em: 12 abr. 2005.

KNUTSSON, Björn et al. *Peer-to-peer support for massively multiplayer games*. In: Proceedings of INFOCOM 2004, Hong Kong, 2004. Disponível em: <http://www.cis.upenn.edu/%7Ehhl/Papers/infocom04.pdf>. Acesso em: 26 fev. 2005.

KÖYKKÄ, Maria et al. *Usability heuristic guidelines for 3D multiuser worlds*. In: Proceedings of the 1999 Conference on Computer Human Interaction, Wagga Waggam, 1999. Disponível em: <http://www.comlab.hut.fi/hft/publications/usability.pdf>. Acesso em: 9 mar. 2005.

LARZON, Lars-Åke; DEGERMARK, Mikael; PINK, Stephen. *UDP Lite for real time multimedia aplications*. Extended Enterprise Laboratory, Bristol, 1999. Disponível em: <http://www.hpl.hp.com/techreports/1999/HPL-IRI-1999-001.pdf>. Acesso em: 09 abr. 2005;

LINEAGE. *Lineage* (1998-2005). Disponível em: <http://www.lineage.com/>. Acesso em: 13 abr. 2005.

LINEAGE2. *Lineage II* (2004-2005). Disponível em: <http://www.lineage2.com>. Acesso em: 09 abr. 2005.

LUI, John; CHAN, M. *An efficient partitioning algorithm for distributed virtual environment systems*. In: IEEE Transactions On Parallel And Distributed Systems, Vol. 13, No. 1. IEEE, 2002. Disponível em: <http://dx.doi.org/10.1109/71.993202>. Acesso em: 09 abr. 2005.

MARSHALL, D.; DELANEY, D.; MCLOONE, S.; WARD, T. *Challenges in modern distributed interactive application design*. National University of Ireland, Maynoth, 2004. Disponível em: <http://www.cs.nuim.ie/research/reports/2004/nuim-cs-tr-2004-02.pdf>. Acesso em: 12 abr. 2005.

MEAD, Matthew. *Using the Java Native Interface with Delphi*. Disponível em: <http://home.pacifier.com/~mmead/jni/delphi/>. Acesso em: 17 out. 2005.

MERIDIAN59. *Meridian 59* (1996-2005). Disponível em: <http://www.meridian59.com>. Acesso em: 12 abr. 2005.

METEORUS. *Webgame acessado através de browsers* (2002-2005). Disponível em: <http://www.meteorus.com.br>. Acesso em: 26 fev. 2005.

MUDCONNECT. *The mud connector:* mud listings (1994-2005). Disponível em: <http://www.mudconnect.com>. Acesso em: 09 abr. 2005.

OLIVEIRA, Jauvane Cavalcante de. *Issues in Large Scale Collaborative Virtual Environments*, 2001. 135 f. Tese (Doctor of Philosophy) - University of Ottawa, Ottawa.

ONDREJKA, Cory. *Escaping the gilded cage:* user created content and building the metaverse. In: State of Play: Law, Games and Virtual Worlds, New York, 2004. Disponível em: <http://www.nyls.edu/pdfs/v49n1p81-101.pdf>. Acesso em: 09 abr. 2005.

PANTEL, Lothar; WOLF, Lars C. *On the suitability of dead reckoning schemes for games*. In: Proceedings of the 1st workshop on Network and system support for games, Bruanschweig, 2002. Disponível em: <http://doi.acm.org/10.1145/566500.566512>. Acesso em: 11 abr. 2005.

PELLEGRINO, Joseph D.; DOVROLIS, Constantinos. *Bandwidth requirement and state consistency three multiplayer game architectures*. In: Proceedings of the 2nd workshop on Network and system support for games, Redwood City, 2003. Disponível em: <http://doi.acm.org/10.1145/963900.963905>. Acesso em: 11 abr. 2005.

PHP. Scripting language. Disponível em: http://www.php.net/ Acesso em: 14 out. 2005.

PLANESHIFT. *Planeshift.* Atomic Blue Corporation, 2001-2005. Disponível em: <http://www.planeshift.it/main_01.html>. Acesso em: 12 abr. 2005.

PURBRICK, James; GREENHALGH, Crhis. *Extending locales: awareness management in MASSIVE-3*. School of Computer Science and Information Technology, University of Nottingham, 1999. Disponível em: <http://www.crg.cs.nott.ac.uk/research/systems /MASSIVE-3/docs/locales-submit.pdf>. Acesso em: 26 fev. 2005.

SECONDLIFE. *Second Life* (2004-2005). Disponível em: <http://secondlife.com>. Acesso em: 09 abr. 2005.

SERIN, Ekrem. *Design and test of the cross-format schema protocol (XFSP) for networked virtual environments*. 2003. Tese (Master of Science in Computer Science) - Naval Postgraduate School, Monterey. Disponível em: <http://www.npsnet.org/~npsnet/v /theses/serin.pdf>. Acesso em: 26 fev. 2005.

SMED, Jouni; KAUKORANTA, Timo; HAKONEN, Harry. *A review on networking and multiplayer computer games*. Turku: Turku Centre for Computer Science, 2002. Disponível em: <http://www.tucs.fi/publications/attachment.php?fname=TR454.pdf>. Acesso em: 26 fev. 2005.

SNOWDON, Dave; GREENHALGH, Chris; PURBRICK, Jim. *Inside MASSIVE-3:* flexible support for data consistency and world structuring. In: Proceedings of the Third International Conference on Collaborative Virtual Environments, San Francisco, 2000. Disponível em: <http://doi.acm.org/10.1145/351006.351027>. Acesso em: 12 abr. 2005.

SWGALAXIES. *Star Wars galaxies: MMORPG*. Sony Online Entertainment, 2003-2005. Disponível em: <http://starwarsgalaxies.station.sony.com/>. Acesso em: 12 abr. 2005.

STEUER, Jonathan. *Defining virtual reality: dimensions determining telepresence*. Stanford University, Stanford, 1993. Disponível em: <http://www.cybertherapy.info/pages /telepresence.pdf>. Acesso em: 09 abr. 2005.

TANENBAUM, Andrew S. *Redes de computadores*. 3. ed. Campus, Rio de Janeiro, 1997.

UO. *Ultima Online*. Origin System, 1995-2005. Disponível em: <http://www.uo.com/>. Acesso em: 12 abr. 2005.

WEB 3D CONSORTIUM. *Extensible 3D and standard authoring interface specification:* ISO/IEC final draft international standard 19775:200x, 2003. Disponível em: <http://www.web3d.org/x3d/specifications/ISO-IEC-19775-IS-X3DAbstractSpecification>. Acesso em: 26 fev. 2005.

WIKIPEDIA. *History of the internet*, 2005. Disponível em: <http://en.wikipedia.org /wiki/History_of_the_Internet>. Acesso em: 13 abr. 2005.

WIKIPEDIAb. *World of warcraft*, 2005. Disponível em: <http://en.wikipedia.org/wiki /World_of_Warcraft>. Acesso em: 13 abr. 2005.

WIKIPEDIAc. *Lineage II*, 2005. Disponível em: <http://en.wikipedia.org/wiki/Lineage_II>. Acesso em: 13 abr. 2005.

WOW. *World of Warcraft* (2004-2005). Disponível em: <http://www.worldofwarcraft.com/>. Acesso em: 09 abr. 2005.

XJ3D. *The Xj3D project*. Disponível em: <http://www.xj3d.org>. Acesso em: 13 abr. 2005.

ZHANG, Beichuan; JAMIN, Sugih; ZHANG, Lixia. Host Multicast: *A framework for delivering multicast To end users*. In: Proceedings IEEE Infocom, New York, 2002. Disponível em: <http://ccl.cnu.ac.kr/seminar/2003/data/383(HM).pdf>. Acesso em: 09 abr 2005.

ZOU, Li; AMMAR, Mostafa H.; DIOT, Christophe. *An evaluation of grouping techniques for state dissemination in networked multi-user games*. Disponível em: <http://www.cs.ubc.ca /~krasic/cpsc538a/papers/zou99evaluation.pdf>. Acesso em: 09 mar. 2005.

APÊNDICE A

ESPECIFICAÇÃO DO SISTEMA DE NETNODES SEGUNDO O MODELO UTILIZADO PELO CONSÓRCIO WEB3D

[X] NetNode Component

[X].1 Introduction

[X].1.1 Name

The name of this component is "NetNode". This name shall be used when referring to this component in the COMPONENT statement (see 7.2.5.4 Component statement).

[X].1.2 Overview

This clause describes the NetNode component of this part of ISO/IEC 19775. This includes how events can be synchronized accross a network, allowing a virtual environment system to be built. Table [X].1 provides links to the major topics in this clause.

[X].2 Concepts

[X].2.1 Network Synchronization

Throughout this document, the term "virtual environment" will be used to indicate a set of semantically related X3D files that can be generally understood as a virtual world (i.e. a set of X3D files that have some form of semantic relation and together make up a system that maintains an arbitrary level of consistency).

In order to allow the contents of a virtual environment to remain consistent throughout all its users, the properties of each entity must be synchronized accross the network(s) that connect said users. In an X3D context, this consists of synchronizing the changes that occur in a node's field accross all *browsers* accessing a specific X3D file (or world).

Some issues, however, must be taken into account when designing the synchronization system. Different applications have different requirements in regard to what information needs to be synchronized, when that process should (or can) occur and which users are allowed to such events. The scalability and extensibility of the system are major features, and usually involve performance tradeoffs, in terms of processing power, memory consumption, bandwidth requirements and message latency. These issues give rise to the design of the overall system and the specification of a set of protocols that are used on entity synchronization.

The NetNode system for X3D synchronization addresses these issues by stablishing an abstract interface where multiple network architectures and synchronization protocols can be used, as suited by each particular application utilizing X3D files.

This system largely builds upon the concepts defined by [ARAKI98].

[X].2.2 NetNodes

Netnodes are defined as X3D nodes that can exchange data among different *browsers* using a set of NetNodeManagers. Only base field type information can be exchanged through a NetNode, and there exists one NetNode for each defined base field type (e.g. SFBool has a NetSFBool NetNode, SFColor has a NetSFColor, and so on) except for SFNode and MFNode fields.

NetNodes can send and recieve information by means of the event routing model, provided by the X3D specification and by assigning them to a valid NetNodeManager, which is the entity responsible for actual information transmission, utilizing a particular network architecture and protocol(s).

[X].2.3 NetNodeManagers

NetNodeManagers are entities that transmit information between *browsers*. The network architecture (i.e. whether it is a client-server or peer-to-peer network), the protocols (if it is reliable, unreliable, has provisions for security, dead-reckoning, etc) and the enconding of information must be standardized so different implementations can still exchange information.

The manner on which NetNodes and NetNodeManagers exchange events, is implementation dependent.

[X].2.4 Protocol Implementations

Every NetNodeManager implementation can be linked to a specific protocol set, that synchronizes messages in a particular way. Examples of such

implementations are a soft-consistency, best effort protocol and dead-reckoned protocol.

Each of these protocols are separately defined, along with their specific NetNodeManager.

[X].2.5 Dynamic Construction of VEs

Dynamic construction of virtual environments means the hability to add, update and remove X3D nodes from the files that represent a multiuser virtual environment during its normal operation, allowing an online, collaborative creation of the scenery.

[X].3 Abstract Types

[X].3.1 X3DNetNode

This is the abstract type from which all other NetNodes derive.

```
X3DNetNode : X3DNode {

  SFNode    [in,out] metadata NULL [X3DMetadataObject]
  SFBool    [in,out] active TRUE
  MFNode    [in,out] netNodeManagers []
  SFString  []       identification ""

  # One of each:
  fieldType [in]     set_value
  fieldType [Out]    value_changed

}
```

The active field indicates whether this particular NetNode is sending messages to the specified NetNodeManagers (if its value is TRUE) or if the events are ignored (if its value is FALSE).

The NetNodeManager field holds a list of NetNodeManagers that this NetNode should use when a set_value event is fired. If the value for this field is an empty list, the NetNode behaves as if its active field was FALSE. If the value identifies an invalid (non-existent) manager, then the behaviour is undefined.

The identication field holds a NetNode-wise, unique identifier that a NetNodeManager uses to determine the sender and intended reciever of messages. If this field has an empty string when a set_value event is fired, it behaves as if the active field was FALSE. If two NetNodes on the same X3D virtual world receive the same identifier, behaviour is undefined. When using the X3D Update Messages encoding, the size of this field should not exceed 255 characters. The responsibility of maintaining a unique identifier for each NetNode lies with the author of

the files that compose the virtual environment. Behaviour of the NetNode system is undefined if two NetNodes share the same identifier.

All nodes derived from X3DNetNode must define the set_value and value_changed fields with the specific field type the NetNode is implementing. Therefore, the SFBoolNetNode node, implementing a NetNode that carries a boolean type should define both as SFBool fields.

Whenever an event is dispatched into set_value, the NetNode should check its active field and direct the event to all NetNodeManagers registered on the netNodeManagers field. When a value arrives at the NetNodeManager and it is intendent for this node (which should be determined through the identification field), the value_changed event will be fired. The way this process (sending and receiving events to and from a NetNodeManager) is accoplished is implementation dependent.

A NetNode may contain a number of fields that specify additional options for a particular implementation of a NetNodeManager. Their presence and semantics are dependent upon the selected NetNodeManagers.

[X].3.2 X3DNetNodeManager

This is an abstract type that defines the base properties of a NetNodeManager.

```
X3DNetNodeManager : X3DNode {

  SFNode    [in,out] metadata NULL [X3DMetadataObject]
  SFBool    [in,out] active TRUE
  SFString  []       identification ""
  SFString  []       protocol ""

}
```

The active field indicates whether this NetNodeManager is sending and recieving events from the network. If this value is FALSE then no messages are synchronized and the NetNodeManager behaves as if all NetNodes connected to it have their active field set to FALSE. If this value is TRUE, messages are sent and recieved as defined on the protocol specification for this NetNodeManager.

The identification field specifies an ID that is used by a *browser* to link NetNodes and NetNodeManagers.

The protocol field should be initialized with a string containing a rough description or an identification of the protocol this NetNodeManager implements.

[X].3.3 X3DDynamicConstructionManager

This is the an abstract type for all managers that implement a dynamic construction protocol for virtual environments.

```
X3DDynamicConstructionManager : X3DNode {

  SFNode    [in,out] metadata NULL [X3DMetadataObject]
  SFBool    [in,out] active TRUE
  SFString  [in,out] encoding ".x3d"
  SFString  [in]     locallyCreateObject ""

}
```

The active field indicates whether this node is active (i.e. sending and receiving operations to and from the network).

The encoding field indicates which X3D encoding should be used when sending or receiving events. This corresponds to the file extension used to identify a particular X3D encoding, such as ".x3d" for a XML-encoded X3D file.

The locallyCreateObject field is an inputOnly field that can be used to create nodes on the current X3D scenegraph (i.e. create and insert a set of X3D nodes in the current *browser*'s scene). The content of the event sent to this field should a set of valid X3D nodes, encoded with

[X].4.1 LoopbackNetNodeManager

```
LoopbackNetNodeManager : X3DNetNodeManager {

  SFNode    [in,out] metadata NULL [X3DMetadataObject]
  SFBool    [in,out] active TRUE
  SFString  []       identification ""
  SFString  []       protocol "loopback"

}
```

The LoopbackNetNodeManager does not synchronize any events accross the network. Whenever a NetNode sends an event to an instance of this node, the event is instantly returned to the sender NetNode as if it was received from the network. In essence, this NetNodeManager allows an event graph that contains a NetNode to act as if it was not present. The following file:

```
...
<LoopbackNetNodeManager DEF="loop_nnm"/>
<TimeSensor enabled="false" DEF="TimeS"/>
<TouchSensor DEF="TouchS"/>
<SFBoolNetNode identification="boolNet" DEF="bool_nn">
  <NetNodeManagers>
     <LoopbackNetNodeManager USE="loop_nnm"/>
  </NetNodeManagers>
</SFBoolNetNode>
```

```
<ROUTE fromNode="TouchS" fromField="isActive" noNode="bool_nn"
toField="set_value"/>
<ROUTE fromNode="bool_nn" fromField="value_changed" toNode="TimeS"
toField="enabled"/>
...
```

will execute as if there was a ROUTE directly defined between the TimeSensor and the TouchSensor.

APÊNDICE B

ESPECIFICAÇÃO DO CONJUNTO INICIAL DE PROTOCOLOS IMPLEMENTADOS SOBRE O SISTEMA DE NETNODES

1 X3D Update Messages

The enconding of update (also refered as synchronization) messages is dependent upon each particular protocol. However, this specification defines a basic binary encoding that shall be used on the protocols here defined. This enconding technique is called X3D Update Messages (or XUMS).

When using XUMS, each event sent to a NetNodeManager by a NetNode is enconded on a structure called an Update Packet. Update Packets can be directly sent to remote hosts or an arbitrary number may bundled on an Update Message, that is then synchronized.

The following is an outline of an Update Packet:

```
 ------------------------------------------------
| Id Size (1 byte)   | Identification (up to 255.) |
| Pld Type (1 byte)  | PayLoad (var.)              |
 ------------------------------------------------
```

The Id Size field is a byte containing the size (number of bytes) of the Identification field.

The Identification field contains the identification of the NetNode that originated this event.

The Pld Type field indicates the type of the payload. This roughly corresponds to the type of event (if it is an SFBool, SFVec3f, etc) being transfered.

The Payload field carries the actual information of the event. This content depends on the type of event being transfered (as indicated by the Pld Type field).

1.1 XUMS Binary Encoding of X3D Fields

SFBool (type 1): The value is encoded in one byte. A false value is encoded with a payload value of 0 and a true value is encoded with a payload value of 1.

SFVec3f (type 2): The value is encoded as a sequence (3 items) of 4-byte floating point numbers. Each floating point number is IEEE-754 compatible.

SFRotation (type 3): The value is encoded as a sequence (4 items) of 4-byte floating point numbers. Each floating point number is IEEE-754 compatible.

SFString (type 4): The value is encoded as a sequence of bytes that represent the characters of the string. The characters should be encoded with the UTF-8 character encoding.

2 HTTP Occlusion Table Protocol

In order for a host to participate in a virtual environment that uses a Peer-To-Peer approach to state synchronization, it must learn the address of other hosts, either through a centralized or from a previously known hosts.

Furthermore, the locality of interest that is a common feature in multiuser virtual environments that simulate some form of spatial ditribution, mean that not all nodes are interested in all other nodes of the simulation (for instance, nodes may only be interested in knowing the position and orientation of users that are within an arbitrarily defined viewing frustum). This processes of discarding a subset of all users from simulation will be henceforth called occlusion.

This section defines the HTTP Occlusion Table Protocol (HOTP), whose purpose is to both provide an initial point of contact for nodes participaing in Peer-To-Peer virtual environmentss and as an initial occlusion mechanism, possibly reducing the load and increasing the performance of the overall system.

Version 1.0 of this protocol does not include an occlusion algorithm per se, but serves as the default proof-of-concept algorithm.

The basic operation of this protocol is the following: Each Occlusion Table is represented by a Uniform Resource Locator (URL) accessible using the Hypertext Transfer Protocol (HTTP) protocol.

This resource represents a text (ASCII) document, in which each line is either a comment (if it starts with a sharp '#' character) or the address of a remote host (blank lines are ignored). Each line is delimited by a "\n" (ASCII 0x10) character.

Each address is represented as either an IP or host address, followed by the pipe ('|') character and the port number of the host. This represents an endpoint of the host that is open for external communications (as generally required by P2P protocols).

All implementation should return at least an ID comment (of the form "#id=<specID>") where "implID" is a specification-unique (meaning each different specification of the HOTP protocol contains a different) text identification.

When accessing directly the resource of a HOTP table, all available (and applicable) addesses should be returned. Optionally, different versions and implementations of the basic HOTP protocol may require an arbitrary set of query parameters, that a node may specify by GET or POST HTTP parameters. Such information should be provided with the specification of each new version of this protocol.

2.1 netnode/simple HOTP

This is the default, proof-of-concept implementation of the HTTP Occlusion Table Protocol. The only information used by this occlusion algorithm is the time a host has registered itself with the table. Hosts older than a given timeout constant are automatically removed from the table.

The ID used in this implementation is "netnode/simple".

A host that wishes to allow itself to be contacted by other remote hosts, should register itself with the occlusion table through a GET query parameter named "host". The contents of this parameter follow the basic information of a remote host line defined in section 5 (i.e. it is of the form "<host or IP>|<port number>").

When a host registers itself with the occlusion table, the time the request is received should be saved along with the host information. After a given time has passed (which can be individually configured in each implementation) this host is automatically removed from the occlusion table (unless is sends a new registration request before its time expires).

Registration requests also return the contents of the occlusion table.

The timeout interval defined for the implementation should be returned on a timeout comment (of the type "#timeout=<timeout interval in seconds>"). This serves as a hint to the application as to when to request a new registration.

3 Simple Peer Consistency Protocol

```
SPCPNetNodeManager : X3DNetNodeManager {

  SFNode    [in,out] metadata NULL [X3DMetadataObject]
  SFBool    [in,out] active TRUE
  SFString  []       identification ""
  SFString  []       protocol "SPCP"
  MFString  [in,out] tableURLs ""

}
```

The Simple Peer Consistency protocol (SPCP) is used to synchronize users of a X3D world by spreading update messages on a best-effort basis. No provision is made for whether the messages actually arrive at their intended destionation and in the right order. Messages of this protocol use the X3D Update Message Schema version 0.1 (XUMS 0.1) for delivering the set_value events.

All messages are sent as User Datagram Protocol (UDP) datagrams. Each host participating in a SPCP-synchronized world must be able to maitain bidirectional connections.

No flow control algorithms are specified for this protocol. Three options are provided for implementations. Either all route events are directly sent through the network, or at most one event per timestamp is sent, or events are sent at a rate of either 30 or 60 events per second. An implementation must not rely on other nodes implementing a similar synchronization strategy than itself.

The discovery of remote peers of the network is done through the HTTP Occlustion Table Protocol (HOTP) using version d3net/simple. Any number of occlusion tables may be used, and the timeout for address refreshing is the lowest timeout returned by a table (minus an implementation-dependant safety interval). The tableURLs field is used to indicate the address of the occlusion tables to use with the HOTP protocol.

4 Simple Late-Comer Synchronization Protocol

```
SLCSPNetNodeManager : X3DNetNodeManager {

  SFNode    [in,out] metadata NULL [X3DMetadataObject]
  SFBool    [in,out] active TRUE
  SFString  []       identification ""
  SFString  []       protocol "SLCSP"
```

```
}
```

The Simple Late-Comer Synchronization Protocol (SLCSP) is designed to solve the problem of the synchronization of world events after an (unknown) number of nodes have connected to a virtual environment and have performed changes on the various properties of the environment's objects. However, it does not deal with dynamic addition, update and removal of objects.

When entering a virtual environment that has been running for an arbitrary amount of time, the state of the objects and their properties will (more likely than not) be different because of the interaction of the users with the virtual environment. Therefore, some method of synchronizating the current state of the world (provided that it is not saved with the world description) for the late comer users of the virtual environment is needed.

This is achieved by utilizing NetNodeManager, called SLCSPNetNodeManager, which is responsible for sending events to and receiving events from a server using the HTTP protocol. Such events correspond to properties of the virtual environment that are to be saved for future users of the system.

An SLCP server is identified by an URL and corresponds to what will be refered to as an SLCP table (which holds values for each identified NetNode of a virtual environment). This table is a text (UTF-8 encoded) file, containing one entry per line (with each line terminated by a new-line U+0010 character). Each entry may be a comment (if the first character is '#'), a blank line (which should be ignored) or a key/value pair, holding the last value of an identified NetNode. The identification field of the NetNode is used as key, and is succeeded by a '=' character, after which the numerical type of the field is encoded as a decimal number, followed by a'|' character, after which the value of the NetNode follows, encoded as a string.

The SLCPNetNodeManager requests the contents of the SLCP table by accessing the URL that specifies an SLCP server and using a "getTable" query (such as www.a-server-address.com/slcp.php?getTable=true). Just the presence of the getTable parameter triggers an instantenous return of the table, and no further processing is done. Upon receiving the results, each line of the table should be processed, and the appropriate NetNode should receive the notification of the event. Behaviour is undefined if an entry of the table references an inexistent NetNode.

The sending of values to be stored in the SLCP table is done by enconding the identification and the value of a NetNode to standard key/value pairs in an HTTP POST request to the URL that represents the SLCP server. The

server shall process the request and replace the value of each key with the provided one (or add the pair if does not yet exist). Any number of key/value pairs may be sent with each request.

4.1 SLCP String Enconding of X3D Field Values

SFBool (type 1): The true value is encoded as the literal 'true', the false value is encoded as the literal 'false'.

SFVec3f (type 2): The value is encoded as a sequence (3 items) of floating point numbers converted to a string, with an arbitrary number of numbers after the decimal separator (which should always be a '.'). Each item should be separated by a single whitespace ' ' character.

SFRotation (type 3): The value is encoded as a sequence (4 items) of floating point numbers converted to a string, with an arbitrary number of numbers after the decimal separator (which should always be a '.'). Each item should be separated by a single whitespace ' ' character.

SFString (type 4): The value is encoded as a sequence of bytes that represent the characters of the string. The characters should be encoded with the UTF-8 character encoding.

5 Simple Persistency Protocol

The Simple Persistency Protocol (SPP) is a complement to the SLCP protocol, and is designed to address the need to dynamically add, update and remove nodes from an X3D scene graph accross a network. This protocol is defined in this specification, however it does not follow the regular NetNode system.

The SPP protocol uses a client-server communication model, and the HTTP protocol for message delivery.

The SPP protocol represents a virtual environment in terms of a Master World File (MWF) and several children Object Files, that are added (or rather, instanced) on the MWF, through the means of inline and/or prototyped nodes. Both the Master World File and the Object Files should be valid, XML-encoded X3D files.

The SPP protocol defines a series of operations that can be performed in the virtual environment, which will be further specified.

5.1 The UploadFile operation

This operation allows the adition of new files to an SPP server, with the intent of using them inside a virtual environment as Object Files. The files

are uploaded inside a POST request of multipart/form-data content type, and with the value "uploadFile" for the name attribute of the content-disposition element that carries the binary-encoded file. This technique of file upload is more thoroughly described in [RFC1867].

Upon detecting this operation, the SPP server should decode the file from the request, and add it to the server's filesystem, allowing the file to be accessible from any host that might participate in the virtual environment.

The SPP server is free to copy the uploaded file to any suitable directory of the filesystem, with the condition that this directory is also added to the path of the file, when a CreateObject operation is specified.

If the filename of the newly uploaded file is the same as an existing one, the existing file is overwritten with the contents of the new file.

5.2 The CreateObject operation

This operation is used to dynamically add new content to an accessible virtual environment, by means of adding a new X3D node inside the Master World File that references an external file (either through an inline or prototype node) that was previously uploaded to the SPP server (either through a set of UploadFile operations or by other external means).

The only parameter required by this operation is the filename of the file that contains the object that is to be added to the virtual environment. This parameter is sent as a GET query, with the "createObject" key (for example: www.a-server.com/spp.php?createObject=example.x3d). This filename should not include any server specific path that is added to a filename, when the corresponding file is uploaded through an UploadFile operation. Such server specific path (if defined) should be automatically added by the server when a createObject operation is received (therefore, even if the server copies all uploaded files to the "x3dobjects" folder, the request to add a file named example.x3d that was previously uploaded through an uploadFile operation would be the one given previously).

The specific node configuration that is added to the Master World File as a result of the createObject operation is server dependent. This can range from a simple inline node that references the object filename provided, to a complex array of nodes, with an eventual prototype or inline node. The content of the node(s) added to the MWF should be returned as the result of the query (i.e. as the response of the HTTP request that originated the createObject operation). The client that requested the createObject operation (i.e. an X3D *Browser* that implements the D3net component specification) should add the returned data to the internal representation of the MWF, so that dynamic instantiation of object is readily visible to the user of the *browser*.

Unique identifiers (such as a node's DEF) should be evaluated by the SPP server and replaced by a (when no other form of previously-decided format is available) random string.

The createNode operation of the SPP protocol works only with XML-encoded X3D files.

5.3 SPPManager

```
SPPManager : X3DDynamicConstructionManager {

  SFNode    [in,out] metadata NULL [X3DMetadataObject]
  SFBool    [in,out] active TRUE
  SFString  [in,out] encoding ".x3d"
  MFString  [in,out] urls ""
  SFString  [in]     uploadFile ""
  SFString  [in]     createObject ""
  SFString  [out]    newObjectReceived ""
  SFString  [in]     locallyCreateObject ""
  SFBool    [in,out] autoUpdateObject TRUE

}
```

This node specifies an SPPManager object that can communicate with an SPP server, by means of the Simple Persistency Protocol, as previously specified.

The active field indicates whether this node is active (i.e. sending and receiving SPP operations).

The encoding field indicates which X3D encoding should be used when sending or receiving events. This corresponds to the file extension used to identify a particular X3D encoding, such as ".x3d" for a XML-encoded X3D file.

The urls field indicates the list of server that the manager should contact when performing an SPP operation.

The uploadFile field is an inputOnly field that initiates an uploadFile operation on all remote SPP servers that are currently registered on the urls field. This field should receive the local (i.e. the *browser*'s system) filename of the file that is to be uploaded. When this field is set, the SPPManager implementation should get the contents of the file and send it to all registered SPP servers, as detailed on previous sections.

The createObject field is an inputOnly field that initiates a createObject operation on all remote SPP servers that are currently registered on the urls field. The contents of this field indicate the filename of the Object File that is to be used during the createObject operation.

The newObjectReceived field is an outputOnly field that sends events when the results of a createObject operation were received. This is the set of nodes that were added to the Master World File by the (first reachable, if more that one is available) SPP server.

The locallyCreateObject field is an inputOnly field that can be used to simulate the effects of a createObject operation without actually sending one to the registered SPP servers. In other words, it is used to locally create an object (i.e. create and insert a set of X3D nodes in the current *browser*'s scene) received by means other than an SPP server (for example, a remote host that has itself created the object on remote SPP servers). The content of the event sent to this field should a set of valid, XML-encoded X3D nodes, such as those returned by a createOperation on an SPP server.

The autoUpdateObject field is an inputOutput field that indicates whether the manager should automatically add the results returned from a createObject operation (initiated when a value is sent to the createObject field of an SPP manager) into the current *browser*'s scene. If this field has a FALSE value, then upon receiving the reply of the createObject operation, a newObjectReceived event is sent but no nodes are locally created.