UNIVERSIDADE FEDERAL DE PERNAMBUCO
GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
CENTRO DE INFORMÁTICA

Trabalho de Graduação

Uma interface de comunicação para o ambiente VEPersonal utilizando Xj3D e a API Scene Authoring Interface

Aluno: Domingos Rodrigues de Menezes Neto (drmn@cin.ufpe.br)
Orientador: Fernando da Fonseca de Souza (fdfd@cin.ufpe.br)
Co-orientador: Marcus Salerno de Aquino (msa@cin.ufpe.br)

Recife, Outubro de 2006

Resumo

Os Ambientes Virtuais (AV) vêm se tornando uma alternativa interessante para o desenvolvimento de interfaces mais realistas e intuitivas para os usuários, de modo que os referidos usuários podem navegar ou interagir mais proveitosamente com os AV. Com o crescente uso desse tipo de aplicação, por exemplo em AV de Ensino, sistemas tutores com uso de AV ou bibliotecas virtuais, torna-se interessante também a adaptabilidade dos AV. No entanto, ainda há problemas em aberto, como a adaptabilidade em tempo real. Este trabalho visa o desenvolvimento de uma interface para uma infra-estrutura de ambientes virtuais adaptativos, o VEPersonal, o qual deve usar X3D para visualização e um SGBD com suporte a XML para armazenamento, visando a reusabilidade de objetos virtuais. Esta interface terá recursos para comunicar ações do usuário ao sistema, além de dar a possibilidade de este último fazer alterações na cena em tempo real.

Abstract

Virtual Environments (VE) have become an interesting option for the development of more realistic and intuitive interfaces to the users, allowing them to browse or interact with VE in a more interesting way. With the increasing use of this kind of application, for instance in a Teaching VE or in virtual libraries, its adaptivity becomes interesting as well. However, there are still some open problems, and one of them is real-time adaptation. This work aims at the development of an interface for an Adaptive Virtual Environment infra-structure, VEPersonal, which will use X3D for viewing and a DBMS with XML support for object storage to allow virtual objects reuse. This interface will have features for communicating user actions to the system, alongside the possibility of being able to alter the environment in real-time.

Índice

Lista de Figuras e Quadros

1. Introdução

Com o crescente aumento do poder computacional nos dias de hoje, os Ambientes Virtuais (AV) vêm se tornando uma alternativa interessante para o desenvolvimento de interfaces mais realistas e intuitivas para os usuários. Esses ambientes têm sido utilizados em diversos tipos de aplicações, como por exemplo *e-commerce*, Ensino a Distância (EAD), entretenimento, simulações, entre outras.

Aliado a isso, há o crescimento da largura de banda da Web e o advento e evolução de tecnologias e aplicações multimídia sobre a mesma. Daí, tornou-se viável a utilização de AV na Web com uso de técnicas de Realidade Virtual. Atualmente, é comum utilizar objetos tridimensionais e outros recursos multimídia como imagens, vídeo e som, para aumentar seu realismo e facilitar a representação da informação. Através da simulação da realidade, o usuário pode frequentemente mover-se em um AV e interagir com os ambientes encontrados.

Com o crescente uso desse tipo de aplicação, torna-se interessante também a adaptabilidade dos Ambientes Virtuais. No entanto, há problemas em aberto em relação a esses AV, entre eles a adaptabilidade em tempo real (ou seja, alteração do ambiente durante a interação do usuário), e a apresentação com diferentes níveis de complexidade, armazenados e recuperados de acordo com as mudanças do perfil do usuário. Tudo isso se agrava com o uso da Web, já que as trocas de dados entre cliente e servidor tornam-se mais problemáticas por causa de restrições da própria rede.

Para solucionar alguns desses problemas, o ambiente VEPersonal [Aquino, 2005], em desenvolvimento no CIn/UFPE, propõe uma infra-estrutura para o gerenciamento de componentes de Ambientes Virtuais Adaptativos, cuja adaptatividade é realizada em tempo real, de acordo com o perfil do usuário e da sua capacidade cognitiva de aprendizagem. Os ambientes virtuais são construídos

com objetos 3D reutilizáveis, utilizando a linguagem X3D (http://www.web3d.org/x3d), e armazenados em um Sistema de Gerenciamento de Banco de Dados com suporte a XML. Cada um desses objetos pode conter vários níveis de informação.

O objetivo deste trabalho é desenvolver uma interface capaz de identificar as ações realizadas pelo usuário, comunicando-as ao restante do sistema, e realizar modificações no ambiente, já que o ambiente VEPersonal deverá ser utilizado via Web e, portanto, os usuários estarão conectados remotamente. Estas informações são enviadas para o servidor que será responsável pela atualização do ambiente e inclusão de novos objetos na aplicação do usuário, de acordo com a evolução do seu perfil. Esta interface deverá ser composta por um browser, que permite ao usuário navegar num ambiente tridimensional, e um *applet*, para oferecer um conjunto de ferramentas que possibilitam a manipulação dos objetos desse ambiente.

O trabalho está organizado em sete capítulos, incluindo esta introdução. O Capítulo 2, Fundamentação Conceitual, tem por objetivo detalhar os conceitos básicos de Realidade Virtual e Ambientes Virtuais, discorrendo também sobre Ambientes Virtuais Adaptativos, e detalhando algumas tecnologias usadas para a confecção desses mundos, como VRML e X3D, além de Xj3D e SAI (usadas neste trabalho). O Capítulo 3, O Ambiente VEPersonal, descreve o ambiente e arquitetura do VEPersonal, dando informações sobre o que ele se propõe a fazer, além de destacar o módulo correspondente à interface desenvolvida neste trabalho. O Capítulo 4, Projeto e Implementação da Interface de Comunicação do VEPersonal, é sobre a realização deste trabalho em si, apresentando inclusive dificuldades e resultados. Por fim, o Capítulo 5, Conclusões, irá mostrar os objetivos atingidos, destacando o que foi obtido com o trabalho e fazer sugestões para trabalhos futuros. Os capítulos 6 e 7 referem-se, respectivamente, às referências usadas neste trabalho e aos anexos.

2. Fundamentação Conceitual

Este capítulo tem por objetivo abordar conceitos que dão base a este trabalho: realidade virtual, ambientes virtuais e ambientes virtuais adaptativos. Além desses conceitos, também serão detalhadas as tecnologias VRML (que dá base a X3D, usada aqui), a própria X3D, além de Xj3D e SAI [XJ3D, 2006].

2.1. Ambientes Virtuais

O conceito de Realidade Virtual (RV) tem sido utilizado como um novo paradigma para a visualização de informações, descrevendo Ambientes Virtuais (AV) tridimensionais mais realistas, propiciando ao usuário "entrar" e interagir nesse ambiente através de simulação da realidade [Aquino, 2005].

Realidade Virtual é uma tecnologia de interface avançada entre um usuário e um sistema computacional, cujo objetivo é recriar ao máximo a sensação de realidade para um indivíduo, levando-o a adotar essa interação como uma de suas realidades temporais. Assim, o usuário deve poder imergir (sensação de absorção ou envolvimento profundo, que pode ser atingido tanto com dispositivos utilizados pela RV Imersiva, como com aplicações tridimensionais desenvolvidas com RV Desktop (RV não-imersiva), ou seja, a visualização do ambiente tridimensional interativo através de monitores planos), navegar (envolvimento, exploração de um ambiente virtual) e interagir em um ambiente sintético tridimensional. [Wikipedia, 2006b, Osório et al., 2004]

O conceito de imersão está relacionado à sensação de se estar em outro ambiente ("imersão física"). Essa sensação se dá por estímulos transmitidos aos sentidos dos usuários. No entanto, isso não significa necessariamente que todo o corpo esteja imerso, ou que todos os sentidos estejam sendo estimulados. Muitos

jogos, hoje em dia, já dão essa sensação mesmo em um PC desktop. Os estímulos mais importantes são os visuais, gerados por estereoscopia e configurações físicas de *displays* capazes de aumentar o sentimento de imersão, como sistemas CAVE ou capacetes (RV Imersiva). Porém, trabalha-se para que outros sentidos sejam estimulados, para aumentar a imersão: estímulos sonoros (sistemas de som 3D) e tácteis (por exemplo, *force feedback* para sentir colisões ou *haptics feedback*, para sentir textura de objetos).

Já o conceito de interatividade diz respeito às respostas do sistema de RV em função das ações do usuário. Isso envolve a navegação e a capacidade de afetar objetos do mundo virtual. Para tanto, é essencial a geração das imagens em tempo real, o que exige um sistema computacional robusto e o uso de técnicas de otimização na renderização das cenas. A interação pode envolver, em RV Imersiva, dispositivos de entrada não-convencionais, como luvas, dispositivos de reastreamento de corpo, etc. Para RV não-imersiva, são usados dispositivos convencionais como *mouse*, teclado ou *joysticks* [Raposo et al., 2004] .

As principais técnicas de modelagem de ambientes de RV, a qual se refere à representação dos objetos que formam o ambiente, são a geométrica, matricial e topológica [Osório et al., 2004]. O modelo geométrico consiste em uma representação espacial, formada por primitivas geométricas (retas, pontos, polígonos ou sólidos mais complexos). A vantagem dessa representação é a facilidade de manipulação dos ambientes e objetos por computadores, além de ser intuitiva para os usuários. Alguns exemplos de tecnologias usadas nesse tipo de representação são VRML [VRML, 1997], Java3D [Java3D, 2006] e X3D [X3D, 2004] (usada neste trabalho). Já a abordagem matricial consiste em grades de ocupação de tamanho fixo, geralmente representadas em 2D, sendo possível descrever a ocupação espacial por decomposição espacial de células (quadtrees[1],

1 Estrutura de dados de árvore, onde cada nó pode ter até 4 filhos, muito usada para particionar um espaço bidimensional recursivamente em termos de seus quadrantes [Wikipedia, 2006]

para 2D, ou octrees[2], para 3D). Essa representação é simples de se construir, representar e manter, porém consomem muita memória e podem levar muito tempo para serem atualizadas, a depender da resolução da matriz, a qual deve ser precisa o bastante para capturar os detalhes importantes do ambiente. A abordagem topológica baseia-se em grafos relacionando pontos de referência interconectados, onde os nós são os pontos e os arcos as relações entre eles (por exemplo, em um ambiente com 2 salas, cada uma com uma porta, e um corredor, os nós seriam as salas e o corredor, e as arestas as portas). Esse paradigma é apoiado pela ciência cognitiva, porém não se adapta muito bem em implementações com dados métricos. É importante notar que todas essas técnicas podem ser usadas para a construção de modelos 2D ou 3D.

As pesquisas em Ambientes Virtuais têm base no paradigma 3D, já que este possibilita a representação da informação de forma intuitiva para os usuários (organização espacial). Daí, duas características tornam-se mais importantes, a navegação e a interação [Osório et al., 2004]. A navegação diz respeito à movimentação propriamente dita no ambiente, respeitando quando necessário (por exemplo, em modelos de simulação) leis da Física e também características inerentes ao ambiente como detecção de colisão contra paredes, obstáculos, avatares ou outros objetos, além de simular o movimento de objetos de forma adequada (por exemplo, ao tentar mover uma caixa, deveria ser levado em consideração o peso desta ou o atrito com o chão). A interação, por outro lado, refere-se à possibilidade de selecionar ou mover objetos, trocar informações, comunicação com agentes, ou ainda interações multi-modais (por exemplo, síntese e reconhecimento de voz, exploração de gestos).

Atualmente, com o crescente aumento do poder computacional, as aplicações para os Ambientes Virtuais (AV) só tendem a crescer. Eles já são

2 Similar a quadtree, porém cada nó pode ter até 8 filhos, e é usada para decompor espaços tridimensionais em termos de seus octantes

utilizados em, por exemplo, e-commerce, Ensino a Distância (EAD), entretenimento, simulações, entre outras. Hoje, os Ambientes Virtuais também podem ser utilizados na Web, o que foi viabilizado pelo crescimento da largura de banda da Internet. Esses AV podem usar várias tecnologias como base, entre elas VRML (Virtual Reality Modeling Language), X3D (baseado em VRML, com sintaxe XML), Sistemas de Banco de Dados (para armazenar objetos VRML/X3D), Java3D, além das tecnologias necessárias para renderização, como OpenGL ou DirectX [VRML, 1997; X3D, 2004; Java3D, 2006; DirectX, 2006].

2.1.1. Ambientes Virtuais Adaptativos

Muitas aplicações em AV requerem adaptação do ambiente de acordo com o perfil do usuário (por exemplo, conforme as preferências e interesses dos usuários), porém ainda há muito pouco sendo feito em termos de pesquisa. Nesse processo, elementos dinâmicos e interativos são introduzidos de modo a melhorar a interação do usuário com o ambiente; além disso, deveria ser possível ao ambiente notificar as ações do usuário para possibilitar o processo de adaptação. Nesse processo, os ambientes usam informações e suposições, conhecidas como Modelo de Usuário, sobre usuários individuais ou grupos de usuários, necessárias para que o sistema tenha no que se basear na determinação de como o sistema irá se adaptar. Segundo Osório [Osório et al., 2004], esses sistemas são mais utilizáveis do que os não adaptativos. No entanto, a adaptação de conteúdo 3D ainda é pouco estudada, embora promissora.

Esse processo de modelagem de usuários envolve coleta de informações necessárias e a representação das informações. Vários métodos têm sido usados, porém eles podem ser agrupados em dois conjuntos, os explícitos (coletados diretamente, por exemplo, por formulários) e implícitos (por monitoramento do comportamento do usuário durante a interação com o sistema).

Os métodos explícitos coletam informações diretamente dos usuários. Por isso, o processo de adaptação torna-se mais simples, já que pode ser direcionado explicitamente a depender das escolhas dos usuários; por exemplo, em um ambiente de e-commerce pode-se disponibilizar apenas os itens que interessam ao usuário, perguntando a ele o que ele deseja, ou ainda mudar a disposição dos mesmos a depender dessas escolhas.

Já os métodos implícitos devem inferir essas informações, dependendo de como o usuário interage com o ambiente. Normalmente, esses métodos envolvem coleta de dados navegacionais e transacionais (por exemplo, histórico de navegação ou palavras-chave usadas em buscas). Em ambientes 3D, essa coleta também deve levar em consideração o que o usuário visualizou ou os objetos com que o mesmo interagiu; isso pode ser feito por verificações tais como aproximação (o usuário ficou a uma distância mínima de um determinado objeto), a posição e orientação da câmera (para determinar para onde o usuário está olhando), ou a própria interação do usuário (objetos com os quais o mesmo interagiu, por exemplo, "empurrar uma caixa") [Osório et al., 2004].

Outro aspecto importante para o processo de adaptação é a organização espacial de objetos. Esta pode usar algum critério semântico, o que poderia, além de tornar o ambiente mais intuitivo à navegação do usuário, facilitar a coleta de dados sobre ele e a elaboração ou refinamento dos modelos de usuários (no caso de se usar métodos implícitos), ou ainda realizar o processo de adaptação em tempo real. Por exemplo, em uma loja virtual, a organização dos produtos pode ser realizada de acordo com o tipo de produto. Em um processo de adaptação em tempo real, esse ambiente pode verificar a freqüência com que um certo usuário vai a determinada seção (eletrônicos, vestuário, etc) e determinar o interesse dele nesta seção. Daí, uma variedade maior de produtos ou de preços poderia ser oferecida nesta seção. Em um ambiente virtual de ensino, é necessário agrupar os conteúdos pela área a que pertencem; os objetos a serem apresentados e as formas

de interação possíveis poderiam ser alterados de acordo com o quanto o usuário aprendeu. Essas informações seriam passadas ao sistema dependendo das respostas do usuário a perguntas, por exemplo.

Dois exemplos de ambientes virtuais adaptativos que usam modelos de usuários são propostos por Chittaro e Ranon [Chittaro e Ranon, 2000] e Osório [Osório et al., 2004]. O primeiro consiste numa loja virtual, cujos layout e estrutura são gerados a partir de um modelo de usuário previamente coletado por formulários, onde são requisitadas informações como gênero, idade, profissão, além de dados mais específicos, como tamanho e estilo da loja ou categorias de produtos de interesse. Então, é estabelecido um *ranking* com os produtos de interesse, a partir desse modelo, e então o mesmo é atualizado de acordo com a interação do usuário com a loja (por exemplo, seções visitadas, produtos comprados) e com regras pré-definidas, como por exemplo "IF visitou(produto)=0 AND numeroVisitas>3 THEN decrementaInteresse(produto)". Daí, um novo ranking dos produtos é obtido. Esse sistema já apresenta a adaptação do ambiente de acordo com o comportamento do usuário, como mostrado acima, porém a mesma não se realiza em tempo real: o ambiente só é modificado a cada nova visita do usuário.

O segundo exemplo, o sistema AdapTIVE [Osório et al., 2004], tem a estrutura dos ambientes e apresentação dos conteúdos modificados tendo base nos interesses e nas preferências dos usuários (coletados explicitamente, por intermédio de formulários) e conforme a manipulação (inserção, remoção ou atualização) de conteúdos no ambiente. A criação de modelos de conteúdos é feita através de um processo de categorização automático destes, e estes modelos são usados na organização espacial desses conteúdos no mundo virtual. No processo de adaptação, os modelos de usuário e conteúdo são utilizados. O modelo de usuário também é utilizado por um agente virtual inteligente, cujo papel é auxiliar os usuários durante a navegação pelo ambiente e também na localização de

informações que o agente julgue relevantes para este usuário. Existe neste trabalho a preocupação de adaptação de conteúdo segundo o perfil do usuário, por meio de suas preferências. Porém, esta adaptação que ocorre no ambiente baseia-se apenas no rearranjo dos conteúdos a serem apresentados em sessões posteriores, não havendo inclusão de novos conteúdos [Aquino, 2005].

O VEPersonal [Aquino, 2005] se propõe a realizar essa adaptação em tempo real, alterando o modelo de usuário de acordo com seu perfil cognitivo (como e quanto o mesmo aprendeu). Para fornecer essas informações, a interface com o usuário deverá ser capaz de notificar o sistema das ações deste, além de carregar e mostrar novos objetos, alterando o ambiente em tempo real. Mais detalhes sobre esse sistema serão vistos no Capítulo 3.

2.2. Tecnologias

Atualmente existe uma grande variedade de ferramentas e tecnologias que possibilitam a construção de Ambientes Virtuais. Essas tecnologias foram viabilizadas pelo aumento do poder computacional, aliado à evolução das placas gráficas, e vão desde bibliotecas gráficas e API[3] como OpenGL, DirectX e Java3D [OpenGL, 2006; DirectX, 2006; Java3D, 2006] (esta última construída sobre uma das anteriores, ou ambas), que oferecem relativa transparência no desenvolvimento de aplicações, no que diz respeito a diferenças entre os *chips* gráficos, além de linguagens descritoras de grafos de cena, como VRML ou X3D (construída sobre XML, também descrita nesta seção).

2.2.1. Grafos de Cena

Grafos de cena são uma importante ferramenta conceitual para representação de ambientes virtuais, descrevendo seus objetos constituintes. Um

3 Acrônimo para Application Programming Interface

grafo é formado por nós, os quais podem ou não conter atributos que influenciam a si mesmo ou a outros nós, conectados por arestas. Esses nós são organizados de maneira hierárquica correspondendo semântica e espacialmente com o mundo modelado.

Os nós podem ser divididos em três categorias: nó raiz, nós internos e nós folha [Raposo et al., 2004]. O nó raiz é o primeiro nó do grafo, e normalmente corresponde à cena em si (embora possa representar um objeto externo que possa ser importado por outras cenas, por exemplo, PROTOs de VRML). Já os nós folhas correspondem geralmente aos objetos que compõem a cena (por exemplo, primitivas geométricas ou objetos importados de origens externas, ou nós de áudio quando o ambiente fornece esse recurso). Os nós internos, também chamados de nós de agrupamento, comumente correspondem a transformações geométricas (rotação, translação, escala, ou o próprio agrupamento de outros objetos).

Uma propriedade muito importante dos grafos de cena é a herança de estado, ou seja, os nós do grafo devem herdar todas as propriedades dos nós posicionados hierarquicamente acima. Isso é usado, por exemplo, para se definir transformações geométricas nos objetos da cena, como rotação ou translação. Ou seja, todos os nós definidos dentro de um nó de translação sofrerão esse efeito, sendo eles nós folhas ou não.

Por exemplo, em uma cena que represente uma casa, seria possível adicionar um nó de rotação, para dar uma certa orientação na casa. Além disso, pode-se adicionar, para cada quarto, uma translação que irá posicioná-lo corretamente em relação à casa. Então, cada objeto colocado num quarto sofrerá a mesma translação em relação à casa, além da rotação inicial. Isso se dá devido à herança de estado. O grafo de exemplo é ilustrado na Figura 2.1.

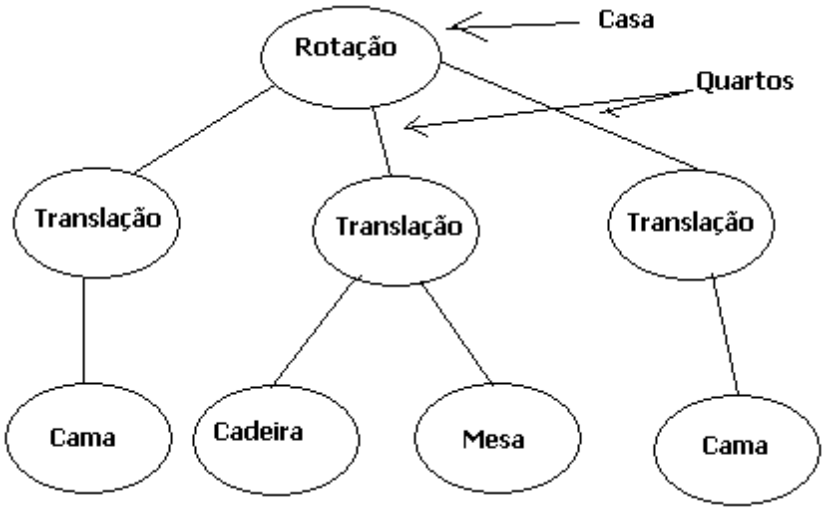

Figura 2.1. Exemplo de grafo de cena [Adaptado de Raposo et al., 2004]

Outra vantagem dos grafos de cena é a implementação de uma série de otimizações (por exemplo, redução do nível de detalhes, quantidade de polígonos/vértices enviados à placa gráfica a depender da distância do observador, eliminação de objetos além do plano de corte, *fogging*, oclusão, entre outros), necessárias para a renderização em tempo real. Estas se fazem especialmente necessárias para VRML ou X3D, tecnologias usadas para a construção de mundos 3D na Web. Daí, há uma melhora sensível do desempenho das aplicações. Outras vantagens do uso de grafos de cena são ganho de produtividade (já que o desenvolvimento se torna simplificado, pois o grafo de cena gerencia toda a parte gráfica), portabilidade (os grafos de cena encapsulam todas as tarefas de baixo nível para renderizar a cena e ler/escrever arquivos) e escalabilidade (grafos de cena podem funcionar tanto em PC desktop multimídia como em estações gráficas complexas).

2.2.2. VRML

A linguagem VRML (Virtual Reality Modeling Language) é atualmente o padrão usado para uso em aplicações 3D na Web. Essa linguagem se baseia nos princípios de grafos de cena, ou seja, conta com um conjunto de primitivas geométricas e de nós de agrupamento, usados para descrever mundos virtuais em 3D. Mundos construídos com essa linguagem são visualizados por meio de *plugins* usados em *browsers* Web, sendo que o mais usado é o Cortona VRML, da ParallelGraphics [ParallelGraphics, 2006].

Além dos recursos típicos de grafos de cena, a linguagem VRML possui recursos para criação da interação do usuário com o ambiente virtual (sensores e rotas de navegação) e oferece a possibilidade de associar comportamentos (*behaviors*) aos objetos modelados. Neste caso, os objetos podem se mover pelo mundo virtual e responder a eventos temporais ou disparados pelo usuário. Também é possível adicionar som espacial e filmes de vídeo à cena, tornando o ambiente virtual muito mais realista. Além disso, os objetos podem ser associados a arquivos de texto, arquivos HTML ou a outros sites HTML [Aquino, 2005].

Abaixo, o Quadro 2.1 mostra os principais nós de VRML:

TIPO DO NÓ	**NÓS**
Nós de agrupamento	Anchor, Billboard, Collision, Group, Transform
Grupos especiais	Inline, LOD, Switch
Nós comuns	AudioClip, DirectionalLight, PointLight, Script, Shape, Sound, SpotLight, WorldInfo
Sensores	CylinderSensor, PlaneSensor, ProximitySensor, SphereSensor, TimeSensor, TouchSensor, VisibilitySensor
Geometria	Box, Cone, Cylinder, ElevationGrid, Extrusion, IndexedFaceSet, IndexedLineSet, PointSet, Sphere, Text
Propriedades Geométricas	Color, Coordinate, Normal, TextureCoordinate,

	Appearance
Aparência	FontStyle, ImageTexture, Material, MovieTexture, PixelTexture, TextureTransform
Interpoladores	ColorInterpolator, CoordinateInterpolator, NormalInterpolator, OrientationInterpolator, PositionInterpolator, ScalarInterpolator
Nós "mutuamente inibidores" (somente uma instância destes nós pode estar ativa)	Background, Fog, NavigationInfo, Viewpoint

Quadro 2.1 – Tipos de nós em VRML [VRML, 1997]

Desses nós, merecem atenção especial os de sensores, os quais permitem interação do usuário com o mundo 3D. Esses nós podem disparar eventos ao serem ativados (por exemplo, um sensor de toque pode iniciar um *script* ao ser clicado, ou um de proximidade pode fazer o mesmo a depender da distância do usuário). Como pode ser visto acima, VRML apresenta implementações de todos os tipos de nós importantes para a definição de um grafo de cena, como os de agrupamento ou os de primitivas geométricas. Também são possíveis em VRML iluminações do tipo pontuais, direcionais ou ambiente, animações, som espacial e filmes (com arquivos do tipo MIDI ou WAV), entre outros. *Scripts*, escritos em Java ou JavaScript, podem ser elaborados para facilitar as animações, de forma a complementar a troca de informações entre os elementos do mundo virtual. Esta propriedade permite a possibilidade de animações e de dinamismo aos objetos inseridos no cenário. [Aquino, 2005]

Além de sensores, outro recurso de comportamento importante (introduzido em VRML97) são os chamados interpoladores, que permitem incorporar animações quadro-a-quadro à cena; estas são montadas a partir da descrição do objeto em determinados pontos. Então, o interpolador faz todos os cálculos necessários para as descrições intermediárias, garantindo uma maneira eficiente de criar as animações. Por exemplo, um interpolador pode ser configurado com três

posições, e na cena, pode ser associado ao nó de translação de um objeto, fazendo com que este seja alterado, começando pela primeira posição, passando pela segunda e terminando na terceira.

VRML também conta com os nós PROTO e EXTERNPROTO. Esses nós servem para instanciar objetos compostos previamente construídos, que podem estar definidos na mesma cena (PROTO) ou em um arquivo externo (EXTERNPROTO). Os PROTOs permitem a reutilização de código, podendo contar com uma interface para que se possa alterar propriedades de cada uma de suas instâncias.

Os nós podem ser definidos como uma abstração dos objetos e conceitos do mundo real, como por exemplo, primitivas geométricas como esfera, cubo, cilindro, ou luzes e descrição de materiais que formam essas primitivas (cor, brilho, textura, entre outros). Cada nó possui um tipo com campos e valores. Por exemplo, na Figura 2.2, há dois nós “geometry”, um do tipo “Box” e outro do tipo “Sphere”, cada um definido como nó filho de um nó “Transform” (ou seja, está definido dentro do campo “children” deste). O nó “Box” possui o campo “size” com valor 2.0 2.0 2.0, definindo o objeto cubo com duas unidades de largura, duas unidades de comprimento e duas unidades de altura. Um dos nós “Transform” (o da esfera) tem um campo translação, e outro (o da caixa) tem um translação e outro de rotação. Notar também que cada nó “Shape” tem também um campo “appearance”, que define a aparência do nó, neste caso apenas a cor de cada objeto. Também há, dentro do nó “Transform” da cena (que, pelo princípio da herança de estado, se aplica aos dois nós “Transform” filhos), um nó “DirectionalLight”. O resultado da renderização da cena da Figura 2.2 é mostrado na Figura 2.3.

```
#VRML V2.0 utf8
Transform {
  children [
    NavigationInfo {
      headlight FALSE
      avatarSize [ 0.25 1.6 0.75 ]
      type [ "EXAMINE" ]
    }
    DirectionalLight {
    }
    Transform {
      translation 3.0 0.0 1.0
      children [
        Shape {
          geometry Sphere { radius 2.3
          }
          appearance Appearance {
            material Material { diffuseColor 1.0 0.0 0.0
            }
          }
        }
      ]
    }
    Transform {
      translation -2.4 0.2 1.0
      rotation 0.0 0.707 0.707 0.9
      children [
        Shape {
          geometry Box {
              size 2.0 2.0 2.0
          }
          appearance Appearance {
            material Material {
                diffuseColor 0.0 0.0 1.0
            }
          }
        }
      ]
    }
  ]  }
```

Figura 2.2. Exemplo de cena em VRML

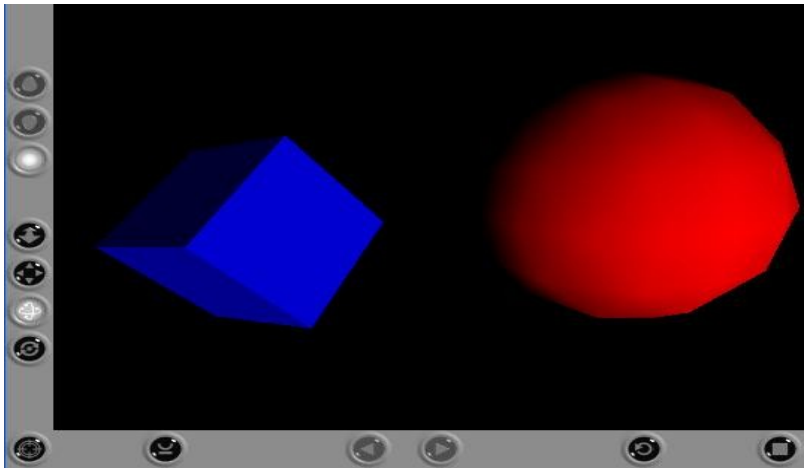

Figura 2.3. Exemplo da cena da Figura 2.2 visualizada no plug-in Cortona

Outro mecanismo introduzido na segunda versão de VRML é o nó *Script*,

que permite utilizar rotinas escritas em Java ou JavaScript para realizar desde simples decisões lógicas até análises complexas de todos os eventos do ambiente. Esse nó pode também ser usado para programar os sensores e interpoladores do ambiente, ou permitir que uma aplicação VRML possa executar a comunicação com computadores remotos na Internet ou identificar as atividades dos jogadores em ambientes multiusuários [Aquino, 2005].

Uma rota (ROUTE) consiste na conexão entre o nó que gera um evento (nó de saída) e o nó que o recebe (nó de chegada). Um evento de um nó pode ser roteado para um evento do mesmo tipo de outro nó [Pollo, 1997]. Através de uma rota, os eventos são utilizados para a troca de mensagens do nó de saída para o de chegada. As mudanças de estado nos objetos da cena são efetuadas dessa maneira, alterando os valores de seus campos quando isso se fizer necessário. Todos os recursos de animação e interação são baseados na adição de comportamentos aos objetos da cena, possuindo eventos que o nó pode receber (eventIn) e enviar (eventOut) [Aquino, 2005].

Como já mencionado, um nó “Shape” contém dois campos, “geometry” e “appearance”. O campo “geometry” pode ser formado por primitivas básicas, que são “Box”, “Cone”, “Cylinder” e “Sphere”, e também por formas avançadas: “Text”, “ElevationGrid”, “Extrusion”, “IndexedFaceSet”, “IndexedLineSet” e “PointSet”. Já o campo “appearance” pode ser formado por alguns tipos de nós que só podem ser usados como parâmetros para outros nós, como é o caso de “Material”, “ImageTexture” e “TextureTransform”, por exemplo. A Figura 2.4 mostra um diagrama com os campos de “Shape” e os nós possíveis de ser usados em cada um deles, além da hierarquia de utilização desta classe de nós.

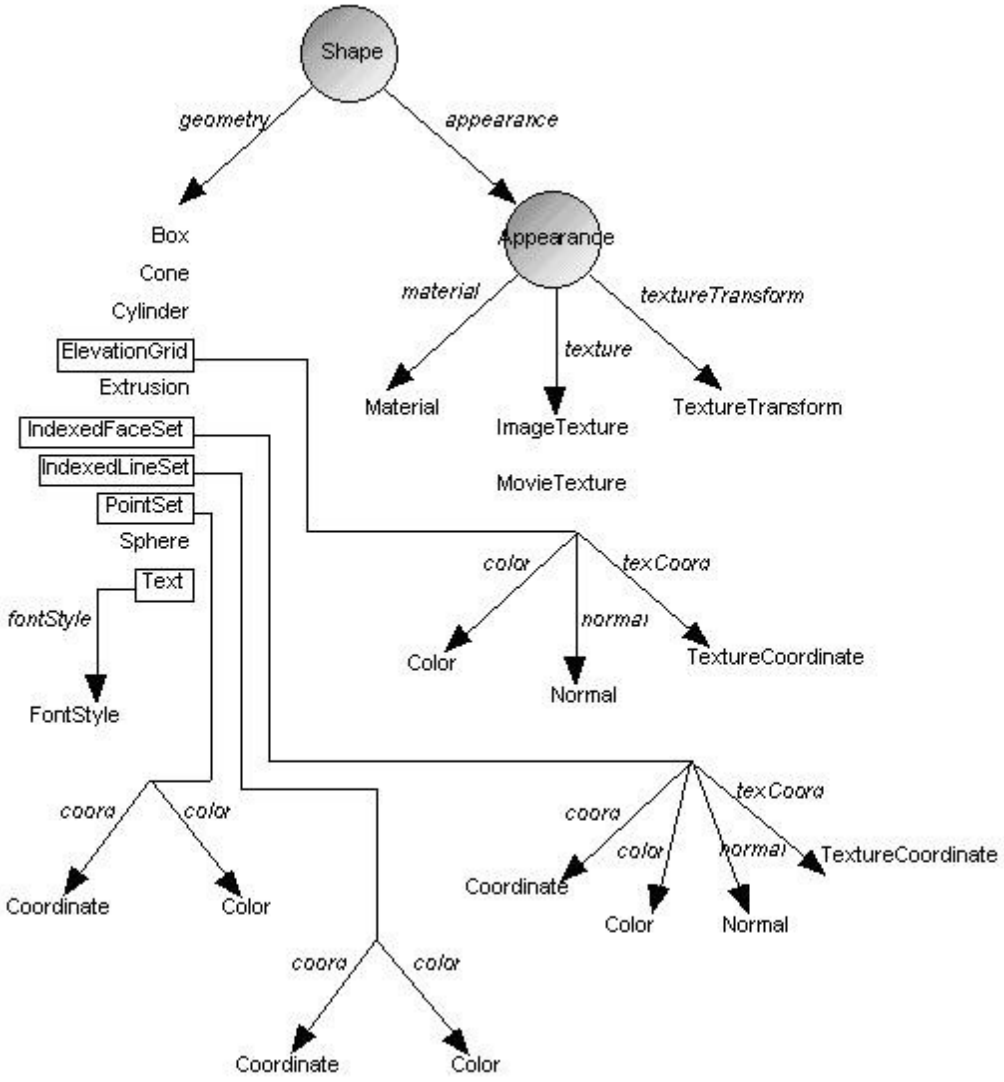

Figura 2.4 – Nós usados como parâmetros (campos) de outros nós [Pollo, 1997]

2.2.3. X3D

X3D (eXtensible 3D) é um padrão aberto ("*royalty-free*") de formato de arquivos e arquiteturas de execução (*run-time*), desenvolvido pelo Web3D Consortium [Web3D] para representação e comunicação de cenas e objetos 3D. É

um padrão ratificado pelo ISO[4] que provê um sistema para o armazzenamento, recuperação e reprodução de conteúdo 3D embutido em aplicações. É um padrão baseado em XML (eXtensible Markup Language)[5] e considerado a evolução de VRML.

O fato de ser uma linguagem baseada em XML permite a X3D vantagens como, entre outras, melhor interoperabilidade entre aplicações (um dos principais objetivos para a criação de X3D). Um dos grandes problemas de VRML é que nem sempre a mesma cena iria ser mostrada identicamente em todos os browsers. Muitas vezes havia diferenças entre implementações, às vezes inclusive quanto à especificação, que poderiam levar a diferenças significativas na representação das cenas), inclusive não gráficas e principalmente via rede, já que XML é, atualmente, o padrão de troca de dados mais usado. Outra vantagem é que a sintaxe também é baseada em XML (embora outros formatos de representação de dados possam ser utilizados), o que possibilita o uso de parsers e API XML já prontos (entre eles, DOM/JDOM). Além disso, também permite comunicação em tempo real de dados, inclusive via rede, e esta é talvez a característica mais importante de X3D que VRML não apresenta. Uma aplicação importante dessa característica poderia ser exatamente o uso em ambientes virtuais adaptativos, em que objetos poderiam ser adicionados ou retirados das cenas sem que todo o mundo seja recarregado, isso poupa largura de banda da rede, além de permitir mudanças mais rápidas das cenas e, mais importante, em tempo real.

Além dessas características, de acordo com Bullard [Bullard, 2003], há outras características importantes que diferenciam X3D de VRML: várias codificações diferentes (binária, XML padrão e a sintaxe VRML clássica), novos recursos gráficos (NURBs[6], animação humanóide, multi-textura, primitiva

4 ISO - International Organization for Standardization

5 Meta-linguagem de marcação desenvolvida pelo World Wide Web Consortium (W3C), sendo o padrão mais usado para publicação e transferência de dados de diferentes domínios de aplicação na Web.

6 **Non-Uniform, Rational B-spline**, é um modelo matemático usado comumente em Computação Gráfica para gerar e representar curvas e superfícies [http://en.wikipedia.org/wiki/Nurbs]

Triângulo), novos recursos de rede (Inline melhorado, LoadSensor – sensor de carregamento) e API melhoradas, com mais *bindings* linguagem-modelo de objetos.

O padrão X3D apresenta uma arquitetura modular, que permite perfis ("profiles") em camadas, os quais podem prover mais e melhores funcionalidades para ambientes imersivos e interatividade melhorada. Porém, a maioria dos domínios das aplicações não necessita de todos os recursos do padrão e tão pouco todas as plataformas dão suporte à variedade de funcionalidades definidas na especificação.

Daí, o padrão X3D agrupa os recursos em componentes (blocos modulares de funcionalidade), definindo uma coleção específica de nós que possui um conjunto de funcionalidades. Os componentes podem ser estendidos individualmente ou modificados através da adição de novos níveis, ou novos componentes podem ser adicionados para introduzir novos recursos. Um perfil é uma coleção de componentes para um específico nível de suporte. Assim, todos os arquivos X3D requerem a definição do perfil que está em uso [Web3D, 2006].

Os perfis atualmente definidos para X3D são Interchange (perfil básico para comunicação entre aplicações, que dá suporte às funcionalidades básicas, como geometria, texturas, iluminação básica e animação. Não há nesse perfil modelo de execução para renderização); Interactive (permite interação básica com um ambiente 3D por meio de nós de sensores, mais tipos de nós de iluminação (por exemplo, Spotlight ou PointLight)); Immersive (inclui, entre outros, os nós de áudio, fumaça, colisão e scripts) e Full (inclui todos os tipos de nós definidos, como NURBs ou H-Anim – animação humanóide). A vantagem deste tipo de especificação em camadas é que se pode dimensionar o perfil desejado a depender do tipo de aplicação que se pretende desenvolver. Assim, é possível reduzir o tamanho dos *plug-ins* ao estritamente necessário para executar uma determinada cena, evitando que, por exemplo, arquivos de cenas 3D simples necessitem da

instalação de um *plug-in* X3D muito grande [Aquino, 2005]. A Figura 2.5 mostra os perfis de X3D já definidos.

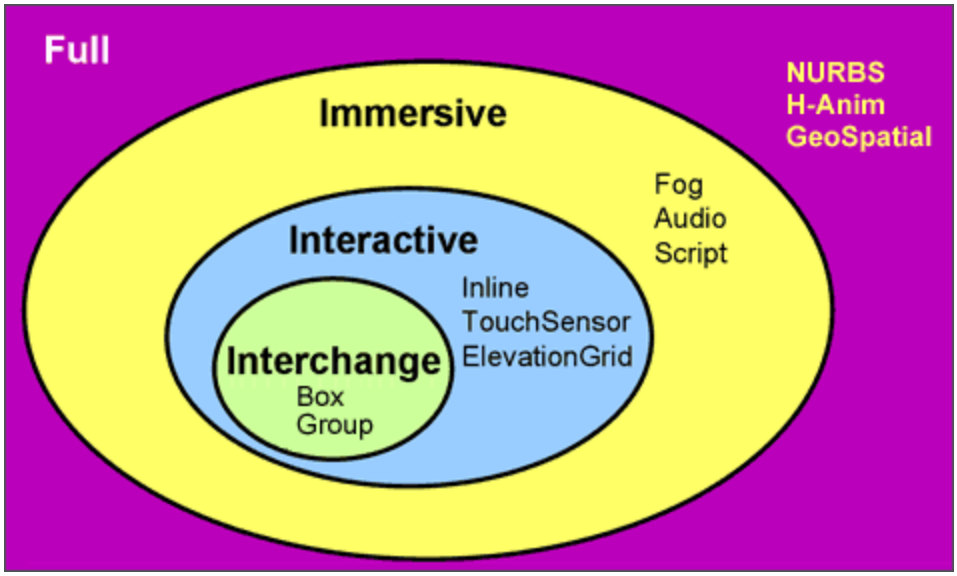

Figura 2.5. Perfis ("profiles") definidos para X3D [Web3D, 2006]

Sendo uma integração de VRML com XML, X3D conta com todos os tipos de nós de VRML descritos na seção anterior. Também conta com três representações de dados possíveis: binário (x3db – sua vantagem é de ter uma representação de menor tamanho, além de facilitar o processamento por máquinas), XML padrão (x3d – mais legível para humanos, e por estar codificado em XML, permite interoperabilidade com outras aplicações que usem esse padrão) e VRML clássico (x3dv – facilita a transição para usuários de VRML para X3D). Na Figura 2.6 é apresentado um exemplo de arquivo de cena (a mesma cena das Figuras 2.2 e 2.3) X3D, codificado em XML padrão (a codificação em VRML clássico ficaria similar à Figura 2.2). Na primeira *tag* (X3D) já se nota a definição do perfil a ser usado (Immersive). Também é possível notar a correspondência entre os nós de VRML e os de X3D.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.0//EN"
  "http://www.web3d.org/specifications/x3d-3.0.dtd">

<X3D profile="IMMERSIVE"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema-instance"
    xsd:noNamespaceSchemaLocation="http://www.web3d.org/specifications/x3d-3.0.xsd">
  <head>
    <meta name='filename' content='RedSphereBlueBox.x3d'/>
  </head>
   <Scene>
     <Transform>
       <NavigationInfo headlight='false'
         avatarSize='0.25 1.6 0.75' type='EXAMINE'/>
       <DirectionalLight/>
       <Transform translation='3.0 0.0 1.0'>
         <Shape>
           <Sphere radius='2.3'/>
           <Appearance>
             <Material diffuseColor='1.0 0.0 0.0'/>
           </Appearance>
         </Shape>
       </Transform>
       <Transform translation='-2.4 0.2 1.0' rotation='0.0 0.707 0.707 0.9'>
         <Shape>
           <Box/>
           <Appearance>
             <Material diffuseColor='0.0 0.0 1.0'/>
           </Appearance>
         </Shape>
       </Transform>
     </Transform>
   </Scene>
</X3D>
```

Figura 2.6. Exemplo da cena da Figura 2.1 em X3D, sintaxe XML

2.2.3.1. Xj3D/SAI

O Xj3D [XJ3D, 2006] é um projeto *open-source* do Web3D Consortium. Consiste em um *toolkit* feito em Java para conteúdos VRML97 e X3D e provê um *browser* que funciona tanto com VRML97 como com X3D, nos formatos XML padrão e VRML clássico (x3dv). Além do browser, o Xj3D apresenta um conjunto de APIs Java que permitem acesso a este. Dois projetos que usam Xj3D podem ser vistos, respectivamente, em [Bullard, 2003] (Digital Rare Book Library System - DiRBS) e em [Martins, 2006] (protótipo de jogo RPG).

O SAI (Scene Access Interface) é a interface da programação (API) usada para fazer a comunicação entre o X3D e o Java. Esta comunicação é feita com

base nas funções definidas por esta API. Através do SAI é possível uma comunicação com os nós de um mundo X3D, usando o modelo de execução do X3D [Martins, 2006].

Ainda segundo Martins [Martins, 2006], existem duas formas de se usar o SAI, são o acesso interno e o acesso externo. No primeiro tipo de acesso existe um nó de script dentro do arquivo X3D que vai fazer a ligação com uma classe Java, ou seja, é a própria cena X3D que, ao ser carregada num browser X3D, vai chamar a classe Java onde existe o código que vai efetuar as alterações pretendidas na cena. Já a segunda forma de acesso consiste em a classe Java carregar o arquivo X3D, e fazer qualquer tipo de alteração na cena correspondente. Neste tipo de acesso, a classe chama uma instância de um browser, e usa a API para receber informações deste ou efetuar alguma ação na cena (por exemplo, inserir ou remover objetos e alterar nós da mesma). Este tipo de acesso é útil quando se pretende uma aplicação composta por duas janelas, uma janela referente à cena 3D e outra referente à interface Java. As Figuras 2.7, 2.8 e 2.9 ilustram o funcionamento dos dois tipos de acesso. Exemplos de uso do SAI e tutoriais podem ser encontrados em [Martins, 2006].

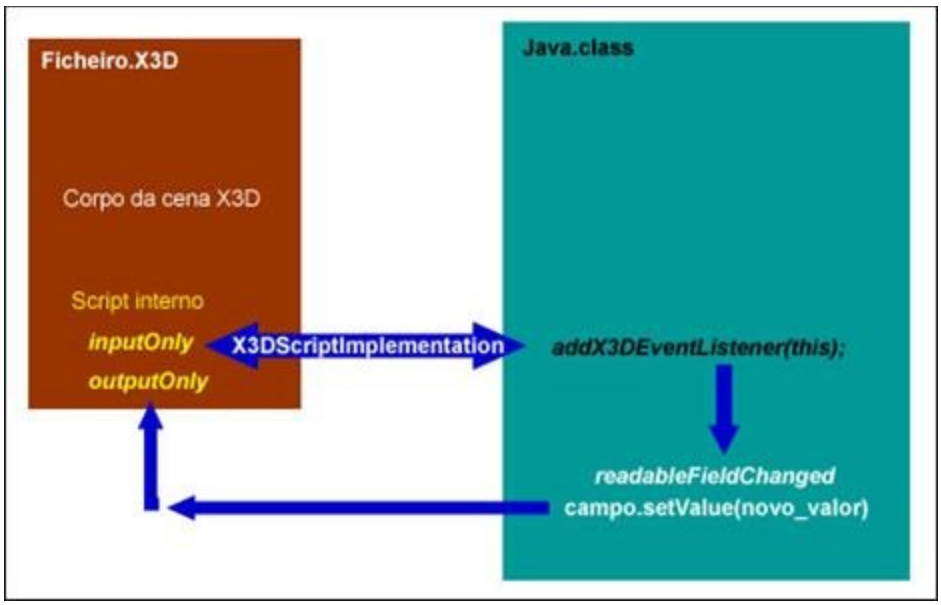

Figura 2.7. SAI - Acesso Interno [Martins, 2006]

```
<TouchSensor DEF="Sensor_Toque"/>
...

Defenição do interface do script

<Script DEF="Script_01" url="&quot;SAIExemplo1.class&quot;">
 <field accessType="inputOnly" name="isOver" type="SFBool"/>
 <field accessType="outputOnly" name="diffuseColor_changed" type="SFColor"/>
</Script>

Defenição das routes

<ROUTE fromField="isOver" fromNode="Sensor_Toque" toField="isOver" toNode="Script_01"/>
<ROUTE fromField="diffuseColor_changed" fromNode="Script_01" toField="set_diffuseColor" toNode="Objecto"/>
```

Figura 2.8. Definição de um nó Script e de rotas em um arquivo X3D para acesso interno [Martins, 2006]

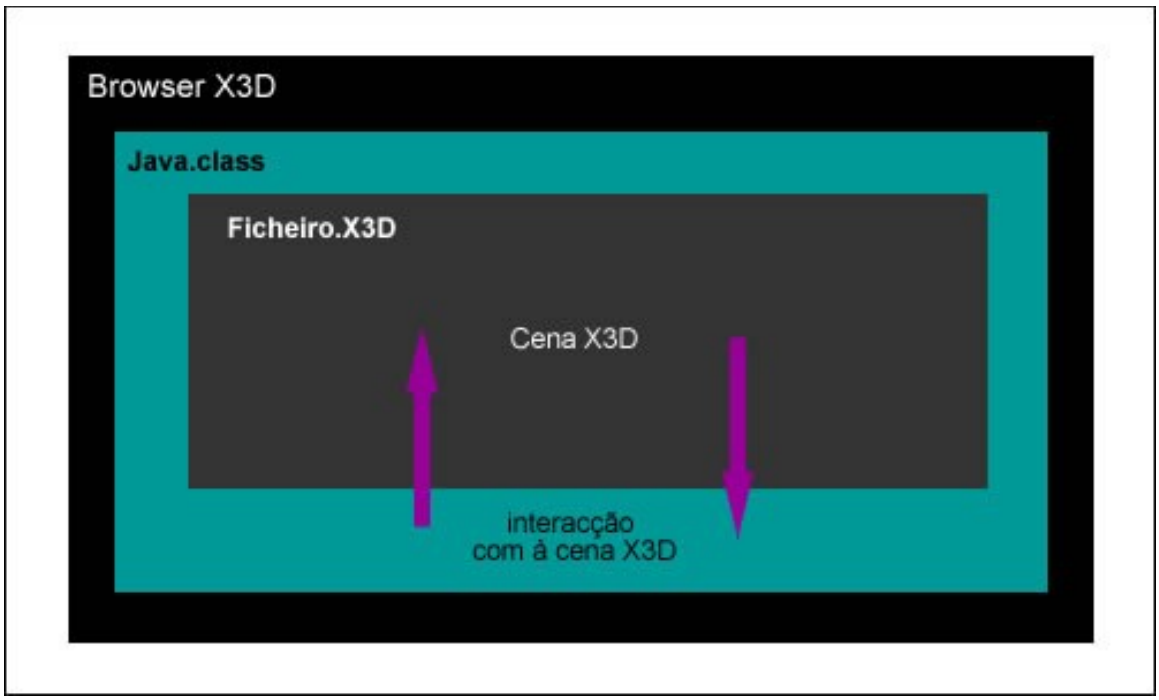

Figura 2.9. Acesso Externo do SAI ilustrado; a própria classe Java acessa a cena X3D [Martins, 2006]

3. O Sistema VEPersonal

O VEPersonal consiste numa proposta de uma infra-estrutura para gerenciamento de ambientes virtuais adaptativos construídos a partir de bases de objetos reutilizáveis, de autoria de Aquino [Aquino, 2005]. Este sistema deverá ser capaz de apresentar as informações relacionadas a um determinado conteúdo com maior ou menor nível de detalhes em função do perfil do usuário. A proposta parte do princípio de que cada usuário deve poder visualizar o mesmo mundo virtual, mas com diferentes níveis de detalhamento, para que um usuário iniciante não seja sobrecarregado com informações para o qual não esteja preparado, mas um usuário mais experiente deveria poder enxergar esses detalhes [Aquino, 2005].

Para esse fim, o sistema deverá contar com informações sobre o perfil de cada usuário e ser capaz de efetuar as alterações no ambiente a partir de decisões tomadas pelo sistema, sobre como apresentar cada informação a cada tipo de usuário. Além disso, o sistema, também, deve ser capaz de alterar dinamicamente o perfil dos usuários durante a interação com o ambiente, fazendo então com que ele possa ser apresentado em mais detalhes à medida em que o usuário vai evoluindo quanto ao uso do ambiente.

Por exemplo, para um Ambiente Virtual de Ensino, um aluno iniciante, que não tenha domínio sobre o que está sendo ensinado, não deverá ter acesso a assuntos que ele não tenha base para entender. Além disso, para um determinado assunto, este deverá ser tratado em um nível de detalhes menor. À medida que esse aluno vai aprendendo mais (esse nível de aprendizado também deve ser medido pelo sistema, para que o perfil desse usuário possa ser atualizado), este assunto vai sendo apresentado em mais detalhes ou novos assuntos, que poderiam usar este último como base, vão sendo disponibilizados. Caso isso não fosse feito, este aluno iniciante não poderia aprender, ou teria sua capacidade de aprendizado

limitada, devido à sobrecarga de informações, ou simplesmente não poderia evoluir por não poder entender o que estivesse sendo apresentado.

A adaptação no detalhamento do conteúdo a ser apresentado ao usuário ainda não é explorada em Ambientes Virtuais. Dois sistemas que se adaptam em função do usuário foram abordados na seção 2.1.1. Estes trabalhos propõem uma estruturação mais adequada para a organização espacial das informações (um rearranjo dos objetos do mundo para se adaptar aos interesses do usuário ou facilitar o acesso às informações que ele solicitar, ou que o sistema julgar necessário), ou seja, as informações fornecidas na construção do ambiente são sempre as mesmas. Além disso, não há adaptatividade em tempo real, ou seja, nesses trabalhos, o sistema não modifica as informações do mundo virtual durante uma mesma sessão. As modificações só são realizadas nas próximas sessões desse usuário.

Uma outra questão abordada nesse sistema é a reutilização de objetos existentes em um Banco de Dados (BD) para a construção de mundos virtuais tridimensionais. A maioria dos sistemas de Ambientes Virtuais atuais, por utilizarem arquivos VRML, armazenam os objetos do mundo em um único arquivo, não utilizando um SGBD para tal e, por conseguinte, não exploram a possibilidade de reuso de objetos [Aquino, 2005].

Daí, para solucionar os problemas de adaptatividade descritos acima, o VEPersonal utiliza um modelo de representação de objetos 3D que permite adaptação do mundo virtual de acordo com o perfil do usuário. Esta adaptação pode ocorrer tanto em função das preferências e interesses do usuário, definidas no início de uma sessão, como também em função da evolução cognitiva do usuário, ou do conhecimento adquirido pelo mesmo, durante a interação com o sistema. O VEPersonal é capaz de monitorar as ações do usuário e, em função do estado atual do ambiente, buscar no SGBD novos objetos a serem adicionados ao mundo virtual (processo realizado por SGBD/linguagens de consulta com suporte a XML,

obedecendo à especificação (realizada através da utilização de ontologias responsáveis pela geração de esquemas para dados semi-estruturados) do mundo a ser criado [Aquino, 2005].

3.1. Objetivos do sistema VEPersonal

Há vários problemas a se considerar na concepção de um sistema para AV Adaptativos. Eles serão discutidos brevemente aqui, juntamente com as soluções propostas.

O primeiro deles diz respeito à interação do usuário com o ambiente, o qual precisa se adaptar aos seus interesses e à sua evolução cognitiva. Para isso, este deve ser monitorado para que se possa identificar seu perfil e sua evolução durante a interação com o ambiente. Este processo permite conhecer as preferências do usuário, bem como sua capacidade cognitiva e seu conhecimento sobre o ambiente, propiciando ao sistema uma melhor adaptação às suas necessidades. Para solucionar esse problema, o VEPersonal contará com uma sociedade de agentes capaz de monitorar as ações do usuário e determinar as modificações (adaptações) necessárias ao ambiente. Também serão importantes um Modelo de Usuário e um Modelo de Ambiente capazes de armazenar os dados pertinentes ao perfil do usuário e as atualizações ocorridas no ambiente, respectivamente. Isto será importante para que o sistema possa decidir sobre as modificações a serem realizadas no ambiente. Não obstante, é importante que na próxima interação do usuário, o ambiente atualizado seja recuperado. Para tanto, é necessário que o Modelo do Ambiente realize um gerenciamento do estado atual do AV e o armazene adequadamente em um SGBD.

Outro problema se refere à representação e armazenamento dos ambientes e objetos que os compõem. Esses mundos virtuais, normalmente baseados em

tecnologia VRML (linguagem que não permite manipular um objeto, percorrendo os seus nós, por exemplo, para poder recuperar parte das informações contidas no mesmo), são armazenados em arquivos. Sendo assim, eles só podem ser visualizados como um todo, não podendo ser utilizada apenas uma parte deles. A proposta do VEPersonal contempla o gerenciamento e a recuperação dos objetos 3D (em X3D, linguagem baseada em XML) por intermédio de um SGBD XML, permitindo a reutilização dos componentes do ambiente em novos mundos, reduzindo o espaço de armazenamento desses mundos e aumentando a eficiência na geração de novos AV.

E por fim, a adaptação de ambientes ainda é realizada off-line. Mesmo que o usuário adquira conhecimento sobre o conteúdo do mundo, a adaptação só ocorre na sessão seguinte. Adaptação em tempo real ainda é um problema em aberto, mas que pode ser resolvido parcialmente com as novas técnicas de armazenamento e recuperação de objetos XML. A interface para o VEPersonal proposta neste trabalho resolverá esse problema em parte. Ao receber dos agentes as informações sobre o ambiente a ser mostrado, ela poderá substituir todo o ambiente, ou mesmo inserir e remover objetos sob demanda (a depender das decisões da sociedade de agentes). Isso também terá parte na solução do problema do detalhamento dos objetos a depender do usuário, já que a interface mostraria apenas os objetos que os agentes determinassem.

3.2. Arquitetura do VEPersonal

O VEPersonal usa um modelo de representação de objetos 3D que permite adaptação do mundo virtual de acordo com o perfil do usuário. Neste ambiente, é dado suporte ao usuário para explorar o mundo, visualizando, aproximando ou tocando nos objetos. Então, o sistema deve monitorar a ação do usuário e fornecer,

como resposta, as modificações do mundo em função do estado atual do ambiente e da evolução do perfil do usuário. Dessa forma, o VEPersonal deve poder procurar por novos objetos no SGBD XML e adicioná-los (ou substituir por outro equivalente) ao mundo.

Para isso, é proposta uma arquitetura a ser usada pelo sistema. Sua infra-estrutura permite a manipulação de objetos com vários níveis de informação e sua inserção de acordo com a evolução do usuário. Esta evolução pode ser caracterizada por um atributo denominado *User Level*. Esse atributo determina o nível de conhecimento do usuário sobre o mundo, e é calculado pelo sistema à medida que o usuário interage com o ambiente. Caso seja identificado uma evolução do seu nível de conhecimento, o *User Level* poderá ser modificado. Então, de acordo com o referido *User Level*, o sistema determina se novos objetos devem ser inseridos no ambiente e/ou se eles devem ser atualizados. Desta forma, o sistema pode se adaptar ao usuário durante a sua interação com o ambiente [Aquino, 2005].

A Figura 3.1. mostra a arquitetura do sistema VEPersonal. Nela, as características e comportamento dos usuários são armazenados no Modelo do Usuário, enquanto as modificações no ambiente são armazenadas no Modelo de Ambiente. Essas informações são obtidas pela interface do usuário. Esta deverá comunicar as ações do usuário à sociedade de agentes (mostrada em "Sistema multi-agente"), a qual é responsável pela análise tanto das informações armazenadas nos Modelos de Usuário e de Ambiente, quanto das ações do usuário durante a sessão. Além disso, essa sociedade é responsável pela atualização das informações nos dois modelos (ainda de acordo com as ações do usuário) e pelas mudanças necessárias no ambiente (por exemplo, inserção/remoção de objetos ou mudanças nos mesmos), que deverão ser comunicadas à interface. Então, essa sociedade deverá ser composta por quatro agentes, destacados a seguir:

O Agente Gerente, responsável pela comunicação entre a interface e os

outros agentes, além do acompanhamento das ações do usuário e do estado atual do ambiente. Este agente também delega tarefas aos outros agentes e coordena a comunicação com eles.

O Agente Pessoal é o responsável pela geração do perfil do usuário e pela atualização do Modelo de Usuário assim que alguma alteração nesse perfil for detectada.

O Agente de Ambiente deve verificar o estado do ambiente virtual e gravar as modificações no Modelo de Ambiente.

O Agente Atualizador deve gerenciar o processo de atualização do mundo virtual por intermédio das informações fornecidas pelos demais agentes.

Durante o processo de decisão, o Agente Atualizador consulta uma ontologia de Domínio que contém a representação do modelo do Ambiente Virtual, e então este gera consultas ao SGBD com suporte a XML. Este último deve armazenar todos os objetos possíveis de serem utilizados para o mundo virtual, definidos na ontologia. O suporte a XML é necessário, pois esses objetos serão definidos em linguagem X3D. De acordo com a representação, são então recuperados os objetos que deverão ser usados para atualizar o mundo virtual. O Agente Atualizador usa a biblioteca JDOM [JDOM, 2006] para extrair objetos (ou partes de objetos) definidos através do atributo userLevel. Então, o Agente Gerente recebe os objetos 3D do Agente Atualizador, gera a estrutura 3D do Ambiente Virtual e o manda à interface do usuário.

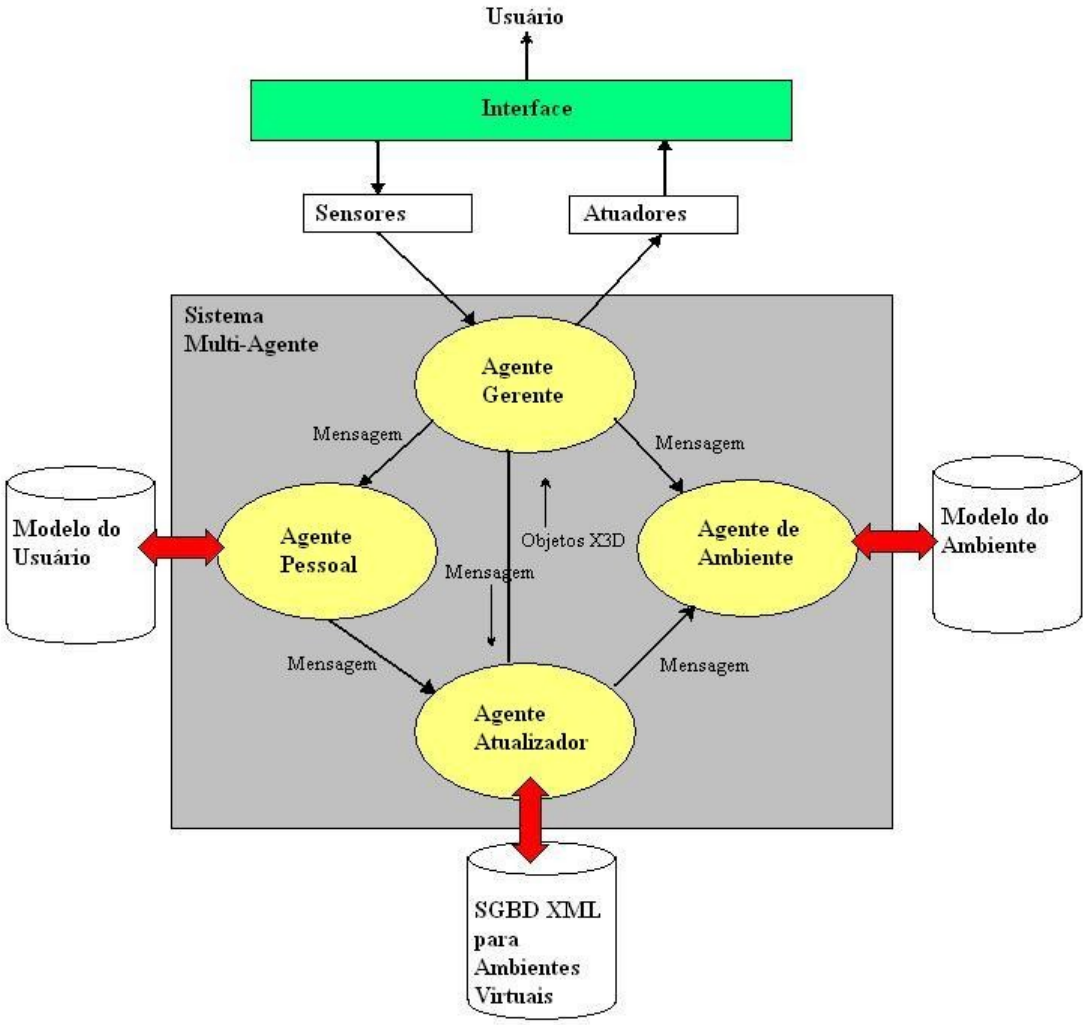

Figura 3.1. Arquitetura proposta para o sistema VEPersonal [Adaptado de Aquino et al., 2006]

A seguir serão mostrados os componentes da arquitetura do VEPersonal. Mais detalhes podem ser vistos em [Aquino, 2005] e [Aquino et al., 2006].

3.2.1. Modelo de Usuário

A função do Modelo de Usuário é armazenar quaisquer informações relevantes (sob o ponto de vista da identificação de seu perfil cognitivo, que implicará na forma com que o ambiente será apresentado) a respeito do perfil de

cada usuário do Ambiente Virtual. Isto possibilita que o ambiente visualizado possa se tornar mais próximo da sua capacidade cognitiva e, portanto, mais interessante para esse usuário. Essas informações são coletadas tanto por meio de coleta explícita (em que um formulário é apresentado ao usuário para que este possa fornecer informações sobre interesses, necessidades e nível de conhecimento), como por coleta implícita. A coleta explícita se faz necessária no sentido de o Agente Pessoal montar um perfil inicial para esse usuário, atualizando o MU e inferindo o userLevel a partir de um conjunto de regras. A coleta implícita é realizada pelo Agente Gerente, usando informações fornecidas pela interface (vide seção 2.1.1 para referências sobre esses tipos de coleta de informação).

Em relação ao processo de modelagem, o Modelo do Usuário deve ter as seguintes propriedades [Aquino, 2005]:

- especialização é individual;
- extensão temporal é de curto e longo prazos (as ações mais recentes e as realizadas em sessões anteriores são consideradas na atualização do modelo);
- modelo é dinâmico, podendo ser alterado durante a interação; e
- atualização do modelo do usuário através de modelo baseado em lógica (em que o MU considera o conjunto das crenças, preferências, interesses e desejos do usuário).

O modelo usado no VEPersonal para determinar o perfil do usuário é um conjunto de regras "IF Evidência THEN Hipótese" que associa um fator de certeza (Certainty Factor – CF). Este fator indica a crença ou descrença em uma hipótese, dada uma evidência, e seu valor pode varir de -1 a +1, onde um fator maior indica maior crença nessa hipótese. Essas regras serão usadas para definir o atributo userLevel do usuário. Mais detalhes sobre as regras usadas e o mecanismo de inferência baseado nestas, além da atuação do Agente Pessoal, podem ser vistos

em [Aquino, 2005; Aquino et al., 2006].

É importante lembrar que todas as informações do Modelo de Usuário devem ser armazenadas em uma base de dados. O acesso a esses dados é realizado pelo Agente Pessoal através de consultas SQL ao SGBD. Desta forma, o sistema poderá armazenar vários perfis de usuário e manter a integridade da informação.

3.2.2. Modelo de Ambiente

O Modelo de Ambiente [Aquino et al., 2006] contém a estrutura do ambiente que está sendo visualizado no momento, e permite que todas as modificações possam ser geradas de acordo com o modelo existente. Todas as modificações ocorridas no ambiente deverão ser gravadas nesse módulo.

Este modelo é constituído por um *framework* que contém a estrutura do ambiente, ou seja, todos os objetos que o compõem, a localização do usuário no ambiente e a descrição das modificações ocorridas durante a interação do usuário.

A presença do Modelo do Ambiente na arquitetura resulta em duas vantagens. Uma delas é a existência do histórico das modificações ocorridas no ambiente. Desta forma, considerando o estado atual do ambiente, inferências sobre adaptações poderão se tornar mais precisas. A outra vantagem consiste na possibilidade de salvar o estado atual ao final de uma sessão para que este possa ser recuperado na próxima interação do usuário.

3.2.3. Os Agentes do VEPersonal

Os agentes são responsáveis pelo acompanhamento das ações dos usuários e das modificações no ambiente. Além disso, também são responsáveis pelas inferências sobre novos objetos a serem incluídos no mundo virtual, assim como a troca de mensagens entre si e com o usuário. Esses agentes são organizados em um

sistema multi-agente que permite a organização e coordenação das atividades para a solução de um problema e facilita a comunicação na sociedade dos agentes.

O uso dos agentes no processo decisório permite a construção de sistemas dinâmicos e eficientes, e se fez necessário por duas razões. A primeira delas diz respeito à capacidade de comunicação entre eles, facilitada pelo uso de linguagens específicas, o que permite a troca de mensagens e informações necessárias ao processo de atualização do ambiente. A segunda se refere à determinação de objetivos, planos e ações, que permite a modelagem de um sistema de inferência com maior precisão. A seguir, são brevemente apresentados os agentes do sistema, retirados de [Aquino, 2005]. Na mesma referência também podem ser vistos mais detalhes sobre o funcionamento dos mesmos.

O Agente Pessoal é um agente reativo (só acionado quando ocorre um estímulo, por exemplo uma demanda do Agente Gerente) que monitora as ações do usuário para que se possa identificar suas características (ou pelo menos as relevantes para a montagem do seu perfil) e adicioná-las ao Modelo de Usuário. Sem isto, não seria possível a geração de ambientes adaptativos. Ele utiliza um sistema de inferência baseado em lógica considerando as crenças que o sistema possa ter sobre o usuário e então determina as ações a serem tomadas dependendo dos estímulos recebidos (por exemplo, gravar informações no Modelo ou enviar uma mensagem ao Agente Atualizador).

O Agente Ambiente também é um agente reativo, e serve como interface entre o Modelo do Ambiente e o Agente Atualizador. Seu papel é verificar as modificações ocorridas no ambiente e atualizar o Modelo do Ambiente. As mensagens necessárias sobre o estado atual do ambiente, para que o Agente Atualizador possa gerar suas inferências, são fornecidas pelo Agente Ambiente (desta forma, o Agente Atualizador não realiza consulta direta ao Modelo do Ambiente).

Já o Agente Atualizador é o responsável por determinar a atualização do

mundo e obter novos objetos do BD para serem incluídos no ambiente. Seu papel é definido em função dos objetivos a alcançar (atualizar o ambiente em função do Modelo du Usuário, por exemplo). Ele deve elaborar um plano (obter informações do Modelo do Usuário e do Modelo do Ambiente, solicitar objetos do SGBD, fazer com que os novos objetos cheguem ao Gerenciador de Ambientes) e determinar uma sequência de ações (identificar o *User Level* do usuário, gerar consultas XML, comunicar-se com os outros agentes) [Aquino, 2005].

E por fim, há o Agente Gerente, que, como já dito, é o responsável pela comunicação entre a interface e os outros agentes, além do acompanhamento das ações do usuário e do estado atual do ambiente. Eventualmente, ele também pode fornecer informações adicionais ao Agente Atualizador sobre o ambiente, e também receber informações sobre como se alterar este, e passar essas informações à interface. Este agente também delega tarefas aos outros agentes e coordena a comunicação com eles.

3.2.4. SGBD com suporte a XML

Um Sistema de Gerenciamento de Bancos de Dados (SGBD) deverá armazenar os objetos do ambiente virtual, os quais deverão ser capazes de serem administrados, recuperados de forma rápida e eficiente e reutilizados a depender da demanda dos agentes (que tomarão essa decisão dependendo do perfil do usuário). O SGBD deverá ter suporte a XML, pois esses objetos serão codificados em formato X3D, com sintaxe XML padrão (por isso, fez-se necessário o uso de X3D). O Agente Atualizador pode, então, gerar consultas ao SGBD para a recuperação de objetos 3D específicos sob demanda para a construção dos ambientes.

O armazenamento de ambientes virtuais no SGBD tem início na definição do esquema baseada na especificação de uma ontologia. Esta permite a

representação de modelos de ambientes virtuais (para padronizar esses ambientes), além da geração e armazenamento de novos ambientes virtuais no SGBD. Essa ontologia também servirá para padronização das informações trocadas entre os agentes, facilitando a comunicação entre eles durante o processo de atualização dos mundos. Com a geração de novos modelos de mundo, será possível um maior grau de reuso dos mundos, principalmente devido à possibilidade de se reusar os objetos 3D já existentes e armazenados no SGBD [Aquino, 2005]. A Figura 3.2. ilustra o processo de geração de modelos de mundo a partir de uma ontologia.

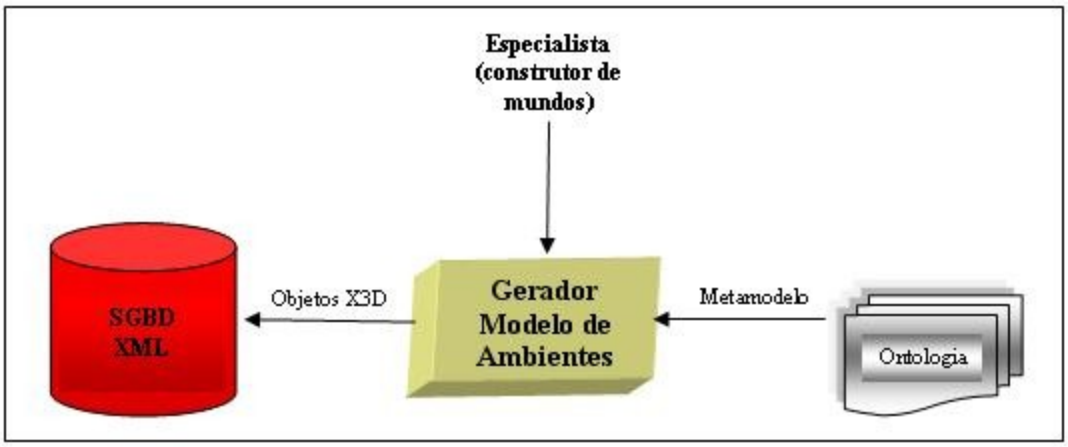

Figura 3.2. Geração de modelos de mundos para armazenamento no SGBD XML [Aquino, 2005]

3.3. A interface com o usuário

O VEPersonal utiliza um ambiente Web para a comunicação com usuários conectados remotamente. Isso se faz importante para que se possibilite a sincronização de informações em um ambiente dinâmico. Daí, torna-se mais efetivo o uso de uma arquitetura cliente-servidor, em que do lado do cliente tem-se a interface, e do lado do servidor tem-se o sistema em si. Isso reduz a carga de informação do lado do cliente, tornando o sistema mais leve e menos oneroso para PC desktop, conforme proposto.

A função da interface é comunicar as ações do usuário ao Agente Gerente,

além de receber informações sobre como se vai atualizar o mundo virtual. Essas decisões serão tomadas pela sociedade de agentes, como mostrado anteriormente. Portanto, a função da interface, na atualização dos mundos, será apenas de executar ordens enviadas pelo agente Gerente, o responsável pela comunicação com o usuário. Logo, a interface deverá disponibilizar a esse agente meios para que este envie um ambiente X3D inteiro para substituição, e também para inserção (por meio de *tags* X3D) ou remoção (por um nome) de objetos do ambiente.

A interface deverá ser composta por um *applet* Java e por um browser X3D. Esse browser usará o Xj3D [XJ3D, 2006], uma implementação *open-source* e feita em Java de um *toolkit* X3D, e permitirá aos usuários navegar no ambiente tridimensional. Já o *applet* deverá oferecer ferramentas para possibilitar a manipulação dos objetos dentro do browser, além de estender a comunicação com o usuário (por exemplo, comunicando ao mesmo a mudança de userLevel, ou solicitando a ele alguma confirmação). Para tornar possível a manipulação dos ambientes virtuais em tempo real (inserindo ou removendo objetos) será usada uma API chamada SAI (Scene Access Interface – vide seção 2.2.3.1), a qual permite a comunicação bi-direcional entre o *applet* e o *browser*. Então, o *applet* deverá conter uma instância do *browser*, além da representação em X3D de uma cena.

Na arquitetura do VEPersonal, esse *applet* deverá também ser responsável por capturar as ações relevantes do usuário dentro do mundo (por exemplo, cliques do mouse ou aproximação de um determinado objeto), mandar essas informações para o Agente Gerente e, a depender da resposta deste (enviada em concordância com as decisões da sociedade de agentes), alterar o mundo virtual de acordo com a resposta recebida. No entanto, a interface não conterá nenhum tipo de informação semântica sobre os mundos. Neste caso, seu papel se resume a passar informações sobre eventos disparados pelo próprio *browser*. Caberá à sociedade de agentes interpretar e tomar decisões a respeito desses eventos. A Figura 3.3 ilustra um

exemplo do funcionamento da interface e a interação com o Agente Gerente.

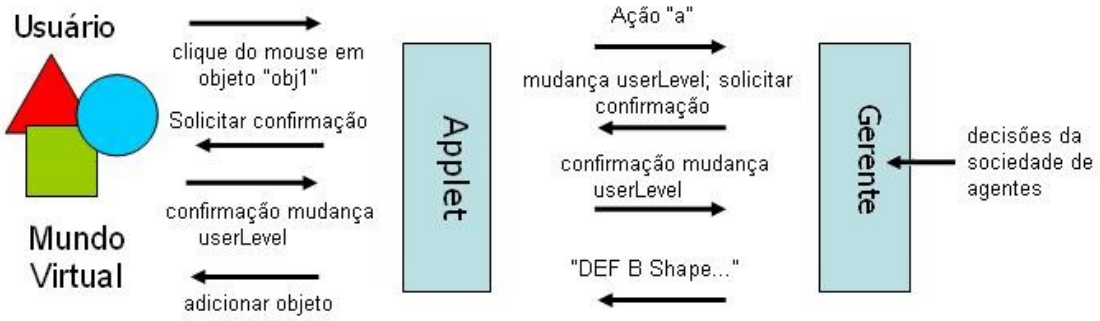

Figura 3.3. Funcionamento e interação da interface com o Gerente [adaptado de Aquino, 2005]

No entanto, no decorrer da implementação, surgiram alguns problemas que fizeram com que essa arquitetura fosse alterada para o protótipo. A seguir, é mostrada a Figura 3.4, que ilustra o funcionamento da interface implementada. Para mais detalhes, vide o Capítulo 4.

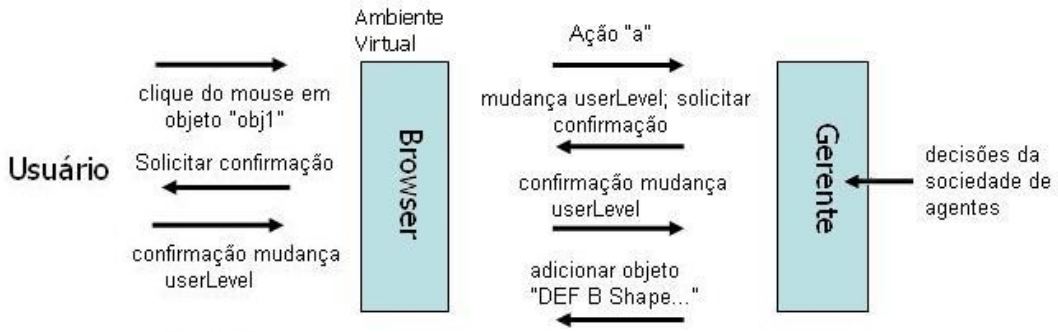

Figura 3.4. Diagrama da interface do protótipo implementado

4. Projeto e Implementação da Interface de Comunicação do VEPersonal

Este capítulo tem por objetivo mostrar as fases de projeto e de implementação do trabalho proposto, abordando as decisões de projeto e a modelagem do mesmo, além da implementação em si, passando pelas dificuldades e limitações, e resultados obtidos. Ele é dividido em quatro seções, a seguir:

A Seção 4.1 aborda as principais decisões de projeto tomadas anteriormente ao início do desenvolvimento propriamente dito. A Seção 4.2 detalha a modelagem do protótipo implementado, ilustrando os diagramas de Casos de Uso e de Classes. Já a Seção 4.3 diz respeito às dificuldades encontradas no decorrer do projeto, e eventuais limitações acarretadas no produto final. Por fim, a Seção 4.4 mostra os resultados obtidos, além de soluções para os problemas encontrados e testes preliminares de validação.

4.1. Decisões de Projeto

A principal decisão tomada refere-se ao uso de um applet como intermédio entre o *browser* Xj3D e o restante do sistema VEPersonal. Em seu lugar, para a implementação do protótipo, foi feita uma aplicação *stand-alone* Java padrão, a qual deve acessar o browser Xj3D e através dele, exibir os ambientes virtuais e realizar as modificações necessárias nestes. Isso se deu por conta de dois problemas descritos a seguir. O primeiro diz respeito à inicialização dos arquivos *jars* necessários para a execução de uma aplicação que use Xj3D. Em [Martins, 2006] há um tutorial mostrando como se fazer a configuração do Xj3D e de

programas que o utilizem, inclusive no que diz respeito à estrutura de pastas recomendada, e foram essas as instruções seguidas. Tentou-se configurar essa estrutura de forma diferente, inclusive por intermédio de IDE[7], sem resultados. A outra dificuldade encontrada foi o próprio Xj3D. Este é um projeto ainda em implementação, portanto ainda apresenta uma série de *bugs* e limitações. Um *bug* que levou a tomar a decisão é relacionado ao mecanismo de renderização. Às vezes o browser não renderizava as cenas, apesar de tê-las carregado. Em outras palavras, o *applet* e o *browser* da Figura 3.3 foram fundidos em uma única classe. Além disso, implementar um protótipo *stand-alone* facilitaria o processo de validação dos requisitos principais. O funcionamento da nova interface foi ilustrado na Seção 3.3.

Outra decisão tomada foi de não se usar estrutura de pacotes para o projeto. Embora pudesse ser utilizada, não o foi para que se facilitasse um eventual processo de integração posterior; sendo assim, bastaria inserir as classes numa pasta e rodar o protótipo. Vale lembrar que, apesar desta decisão, a modularidade da interface não foi comprometida; isso será mostrado com mais detalhes na seção 4.2.

4.2. Modelagem e Implementação do Protótipo

O diagrama de Casos de Uso é mostrado na Figura 4.1. É importante notar que há três atores definidos no mesmo, os quais são o Agente Gerente (mostrado no Capítulo 3), o usuário do sistema (que apenas interage com o ambiente em si) e a própria Interface (que deverá se comunicar com o Agente Gerente).

É importante lembrar que a função da interface é de comunicar as ações

7 Acrônimo para Integrated Development Environment, que significa Ambiente de Desenvolvimento Integrado

efetuadas no ambiente com as decisões tomadas pela sociedade de agentes do VEPersonal. Portanto, os Casos de Uso descritos deverão refletir essa finalidade. Por isso, há serviços descritos tanto para chamada do Agente Gerente para a Interface, quanto para o caminho inverso.

A seguir, serão detalhados os Casos de Uso mostrados na Figura 4.1, primeiramente os disponibilizados ao Gerente, depois os da Interface. Vale lembrar que os únicos Casos de Uso relacionados ao Usuário são triviais, os de navegar pelo ambiente e sair do sistema, portanto não serão detalhados. São eles:

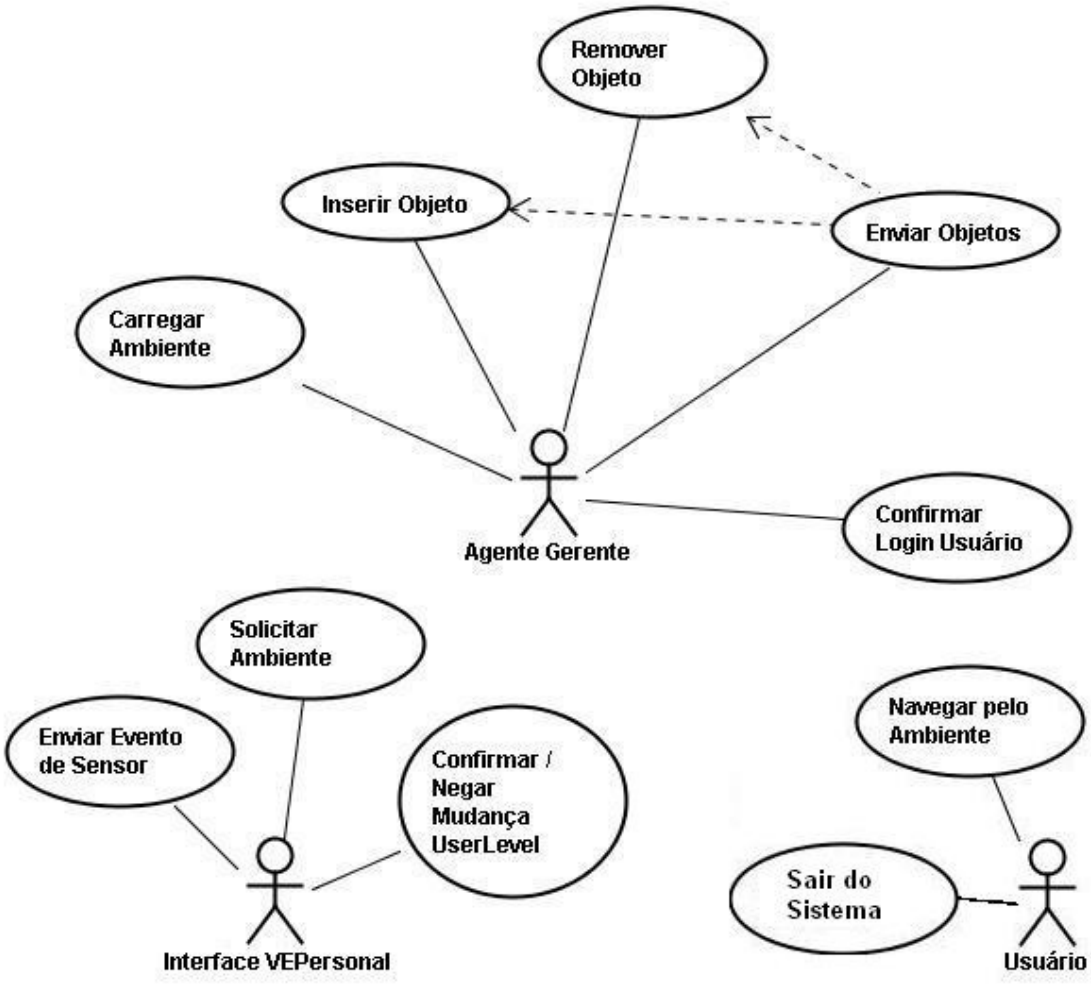

Figura 4.1 – Diagrama de Casos de Uso do projeto

- Carregar Ambiente: Esse caso de uso é chamado, por exemplo, quando da confirmação do login de um usuário, ou do cadastramento de um novo usuário. Esses processos não são ilustrados no diagrama pois deverão ser realizados por outros módulos do VEPersonal. Esse caso de uso também se faz importante já que, às vezes, se faz necessário trocar todo o ambiente no qual o usuário se encontra, por exemplo, entre outros casos, numa eventual "mudança de sala";
- Inserir Objeto: Um dos principais requisitos da interface é que se possa inserir e remover objetos sem que seja necessário se recarregar toda a cena (requisito fundamental da adaptação dos ambientes em tempo real). Portanto, a interface deverá fornecer esse serviço ao Gerente, único que se comunica com a interface, e responsável por comunicar as alterações no mundo virtual à interface;
- Remover Objeto: É um requisito fundamental para a adaptação em tempo real, assim como a inserção. É responsável por remover um objeto da cena, por seu nome;
- Enviar Objetos: Envia um conjunto de objetos à interface, associados a uma ação para cada um. Essa ação pode ser inserção ou remoção do respectivo objeto no ambiente, por isso esse caso de uso depende dos dois citados anteriormente. Esse caso de uso permite que se faça uma série de alterações no ambiente com apenas um comando; e
- Confirmar Login Usuário: Caso de Uso responsável por inicializar o ambiente para um usuário após este ter realizado login no sistema. Deverá fornecer ao sistema o *login* (identificador de acesso ao sistema) e o *userLevel* deste usuário.

A seguir, serão mostrados os casos de uso relativos às chamadas da Interface ao Gerente: É importante lembrar que esses casos de uso não foram implementados, porque a implementação deles pertence ao gerente. No entanto, esses casos de uso são ilustrados pois a implementação da interface depende dos mesmos.

- Solicitar Ambiente: Esse caso de uso é chamado após a confirmação de um *login*. Deverá solicitar ao gerente o ambiente correspondente ao usuário recém-admitido pelo sistema, que será então enviado por meio do caso de uso "Carregar Ambiente";
- Enviar Evento de Sensor: Sempre que o usuário interagir com algum tipo de sensor no ambiente (neste protótipo, restrito a sensores de toque e de proximidade), a interface deverá comunicar ao gerente o *login* do usuário, o *userLevel* deste e o nome do sensor ativado. Para qualquer sensor desses tipos, ativado no sistema, a interface deverá proceder desse modo, já que não há lógica adicional na mesma para lidar com eles; e
- Confirmar/Negar Mudança de UserLevel: Esse caso de uso se faz importante, pois nem sempre interessa ao usuário mudar de *userLevel*. A mudança só será feita diante da confirmação do mesmo.

A seguir, a Figura 4.2 mostra o diagrama de classes do protótipo implementado. Em seguida, as classes do sistema serão explicadas, além de serem mostrados alguns detalhes de implementação, quando for necessário. Para um detalhamento maior, vide o Anexo 1, o qual contém o código-fonte do protótipo.

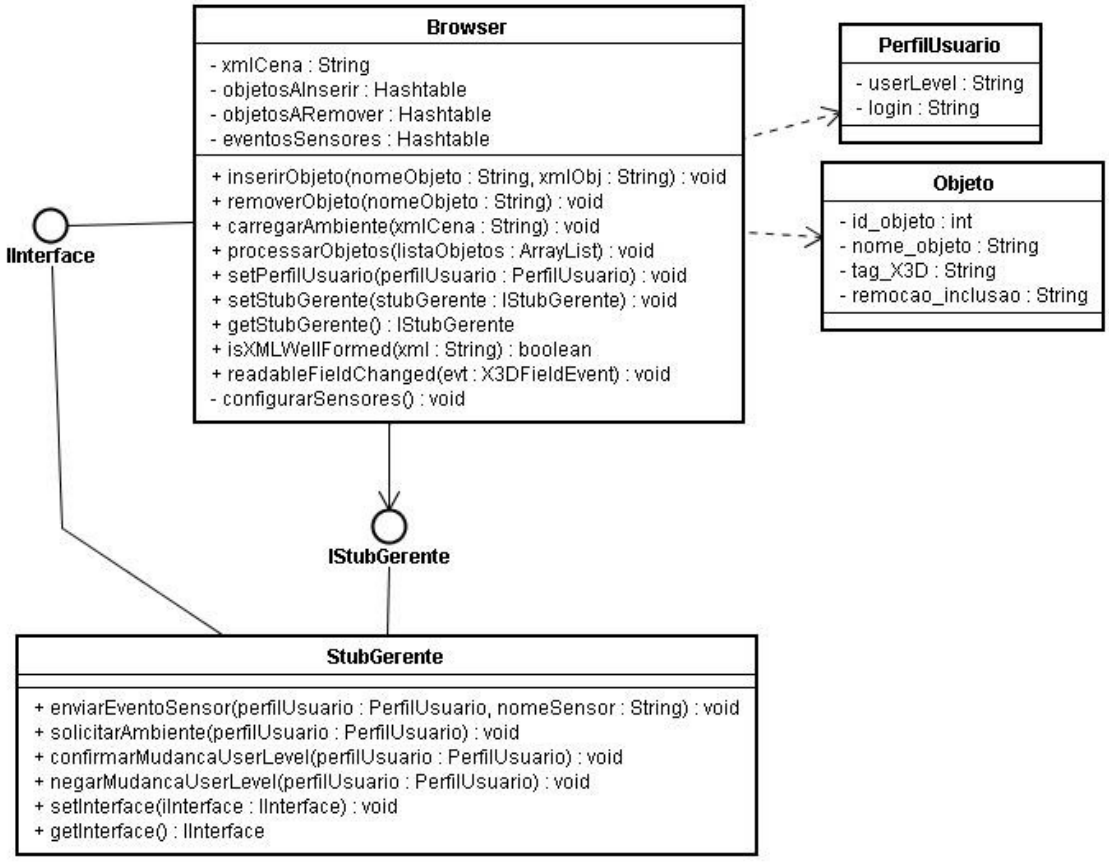

Figura 4.2 – Diagrama de classes do protótipo da interface

A interface Java IInterface e a classe Browser, realização da primeira, correspondem à implementação do Browser propriamente dito. Neste protótipo, essa classe é o resultado da fusão dos componentes "applet" e "browser" da arquitetura mostrada na Seção 3.3. A interface IInterface é a responsável pelo acesso do Agente Gerente aos métodos responsáveis pela inserção e remoção de objetos, além do carregamento de ambiente e confirmação de *login* (método setPerfilUsuario).

O método de carregamento se dá a partir de um *String* Java contendo todo o

código X3D da cena. Podem ser carregados ambientes em formato XML padrão ou em formato VRML clássico. A princípio, não foi implementado um método para carregamento por URL, embora isso seja possível diretamente por SAI.

A inserção de objetos é feita fornecendo um nome para o objeto e um código X3D (ou seja, um conjunto bem-formado de *tags*) que contenha no primeiro campo DEF o mesmo nome fornecido. Este método apresenta uma limitação quanto aos tipos de objetos a serem inseridos; esses objetos sempre deverão vir dentro de algum tipo de nó de agrupamento (normalmente, Group ou Transform, com campo DEF igual ao nome do objeto fornecido para o método) e todos os campos DEF deverão ter nomes diferentes uns dos outros e de quaisquer campos DEF presentes no grafo de cena do ambiente (uma característica da própria linguagem X3D, não da interface). Além disso, esse objeto não deverá conter nós ROUTE ou Script, nem campos USE (para reutilização de nós definidos por DEF). Isso se dá porque a inserção das tags é sempre feita no início do grafo de cena, logo abaixo da primeira tag "<scene>". Outra limitação desse método diz respeito ao formato dos ambientes: esse método não funciona se o ambiente tiver sido carregado por um código em formato VRML clássico, pois o método sempre verifica (através do método isXMLWellFormed) se os códigos da cena e do objeto a ser inserido são códigos XML bem-formados.

Já o método de remoção de objetos é menos limitado. Como só recebe um nome, ele irá procurar um nó na cena com esse nome e removê-lo via SAI. Se for um nó previamente inserido via interface, o método irá procurar o código numa tabela hash contendo pares [nome] [código XML] e removê-lo da string da cena. Se não, irá procurar, no XML da cena, pelo nó contendo o nome fornecido e removê-lo via JDOM.

Outro método importante é o método privado (ou seja, não pode ser chamado pelo gerente) configurarSensores(). Esse método serve para procurar

todos os nós de sensores de toque e proximidade do ambiente e inserir seus nomes na *hash* eventosSensores (mostrada na Figura 4.2). Isso se faz necessário para que se possa avisar ao gerente exatamente qual o sensor de toque ou proximidade ativado pelo usuário (através do método readableFieldChanged, o qual foi implementado devido à interface X3DFieldEventListener, não mostrada no diagrama por vir de SAI). Esse método também apresenta uma limitação, ele só irá configurar esses sensores se eles tiverem nomes pré-definidos. Mais detalhes podem ser vistos no Anexo 1.

A classe StubGerente (e a interface associada IStubGerente) não conta com nenhuma implementação nesse protótipo. No entanto, ela está presente no diagrama, pois seus métodos são chamados pelo *browser*. Ela serviria para a comunicação com o Gerente, diminuindo o acoplamento entre as duas classes e permitindo diferentes formas de comunicação entre elas, só bastando alterar os métodos definidos nessa classe. Além disso, outra razão para a confecção dessas classes é a necessidade de se resolver eventuais diferenças de implementação entre o Gerente e a Interface. Neste caso, uma implementação dessa classe poderia servir como tradutor entre o *browser* e o Gerente, possibilitando inclusive a passagem de alguma informação semântica para este último. No entanto isso restringiria ainda mais as características dos ambientes a serem carregados pelo *browser*, já que nesse caso estas informações seriam acrescidas ao próprio código da classe.

As classes PerfilUsuario e Objeto são classes auxiliares, que encapsulam dados do usuário (*login* e *userLevel*) e de um objeto a ser inserido ou removido. No entanto, essa última classe só é utilizada no método processarObjetos() do *browser*. Este método recebe um ArrayList (classe vinda do pacote java.util) contendo várias instâncias dessa classe, e a depender da ação contida em cada Objeto (atributo remocao_inclusao), vai realizando inserções ou remoções nos

respectivos objetos, representados um a um.

4.3. Dificuldades

No decorrer da realização do trabalho, surgiram alguns problemas que dificultaram o andamento do mesmo em maior ou menor grau. As dificuldades mais importantes serão descritas a seguir, além das eventuais limitações que possam ter acarretado no projeto.

A primeira delas refere-se à inexperiência do aluno em relação ao uso de X3D, o que acarretou em dificuldades no entendimento do funcionamento do Xj3D e do SAI. Aliado a esse problema, existem outros dois intimamente ligados: um deles é o fato de Xj3D ser um projeto ainda em implementação e relativamente recente; sendo assim, apresenta ainda muitas limitações e *bugs*, como já dito na seção 4.1. O outro é o fato de a literatura sobre esses sistemas ainda ser bastante escassa e, além disso, a documentação existente poderia ser melhorada.

Dentre as limitações do Xj3D que acarretaram em limitações no projeto, podem ser citadas: a impossibilidade de se inserir diretamente um conjunto de tags XML na cena como um objeto via SAI. Apenas a remoção é possível, por nome de nó. Por isso foi tomada a decisão de se fazer a inserção diretamente no código, então a classe Browser deve conter um atributo String que representa o código XML da cena. Assim que esse atributo é alterado por meio da inserção do novo objeto (o código é inserido na string contendo a cena, logo após a *tag* <scene>), a cena deve ser recarregada no *browser*, o que leva a outra limitação no protótipo: toda vez que um objeto (ou um conjunto de objetos, se for usado o método processarObjetos) é inserido na cena, o viewpoint volta ao início. Ou seja, se o usuário tiver se deslocado na cena, esse deslocamento é desconsiderado.

Outro problema é que, para se configurar via SAI um sensor de toque para que este possa responder a eventos (no caso da interface implementada, a resposta

é a chamada de um método no StubGerente), deve-se saber de antemão o nome do respectivo sensor (definido por um atributo DEF). Isso implicou em outra limitação no protótipo implementado: para que este reconheça os sensores, no ambiente seus nomes devem ser definidos como “Touch1”, “Touch2”, ...“TouchN”, para sensores de toque (“TouchSensors”) e como “Prox1”, “Prox2”, ...“ProxN”, para sensores de proximidade (“ProximitySensors”). Uma abordagem diferente foi tentada, que configuraria os sensores recursivamente por intermédio da navegação no grafo de cena, porém não foi possível encontrar um meio de recuperar via SAI o nome dos sensores de toque. A implementação desse método ainda encontra-se presente na classe Browser para fins de ilustração, embora o mesmo não seja usado.

Outra dificuldade encontrada, que também acarretou em uma limitação no projeto, diz respeito à inserção de nós ROUTE ou Script. Esses tipos de nós não deverão ser inseridos no sistema via interface, já que fariam referência a objetos que só seriam definidos posteriormente, o que X3D não permite (todos os nós inseridos via interface são incluídos no início do código X3D da cena). No entanto, nada impede a utilização dos mesmos no carregamento de uma cena inteira (via método carregarAmbiente).

4.4. Resultados

Para testar as funcionalidades do protótipo implementado, foi desenvolvida uma classe de teste (cujo código-fonte é descrito no Anexo 2) que faria o papel das chamadas do Agente Gerente. Além disso, foram colocadas, na classe StubGerente, chamadas a System.out.println() para mostrar as mensagens que deveriam ser enviadas ao Gerente, além de respostas a sensores de toques

definidos lá como “padrão” (por exemplo, “TouchSair” avisaria ao gerente e sairia do sistema).

Os resultados preliminares, dadas as limitações do sistema, foram satisfatórios. A interface consegue inserir e remover objetos com sucesso (desde que as limitações referentes a nomes e tipos de objetos a inserir sejam respeitadas), e também carregar ambientes inteiros válidos (neste último caso, tanto com formato XML padrão quanto com formato VRML clássico). Além disso, a classe associa um *userLevel* ao usuário conectado ao sistema (foi implementada também uma função de *login* não-funcional, apenas para atribuir à sessão um nome e *userLevel* de um usuário) e também simula a comunicação deste ao Gerente.

Um resultado importante conseguido com esta implementação foi a flexibilidade do sistema, devido ao baixo acoplamento conseguido com a inclusão da classe StubGerente. Por exemplo, é possível associar uma lógica a esta classe (ainda que não seja relacionada à comunicação com agentes) e fazê-la responder diferentemente a eventos variados dependendo do ambiente carregado.

A seguir são apresentadas, nas Figuras 4.3, 4.5 e 4.6, telas contendo testes preliminares do sistema, relativos à inserção e remoção de objetos, além dos códigos X3D correspondentes aos objetos inseridos na cena. O código X3D referente à cena original de exemplo encontra-se no Anexo 3.

A Figura 4.4 mostra o código X3D referente ao objeto de nome “dad_GROUND2” inserido na cena. Este objeto contém um sensor de toque, o qual é inicializado logo após a inserção. Então, a Figura 4.5 mostra a cena logo após a inserção do objeto. É importante reparar na reinicialização do viewpoint, já que a cena é recarregada. A Figura 4.6 mostra a cena logo após a remoção de dois objetos, e a Figura 4.7 mostra a primeira cena (a qual, nessa ocasião, foi carregada novamente) com um objeto “Pergunta” inserido (Figura 4.8). Também é mostrada

na Figura 4.7 a tela de saída do protótipo (abaixo da tela do *browser*), que mostra a comunicação do evento de clique em um sensor de toque, para um usuário conectado e um *userLevel* associado. Isto mostra a capacidade do sistema de comunicar eventos disparados por ações do usuário da interface para o gerente.

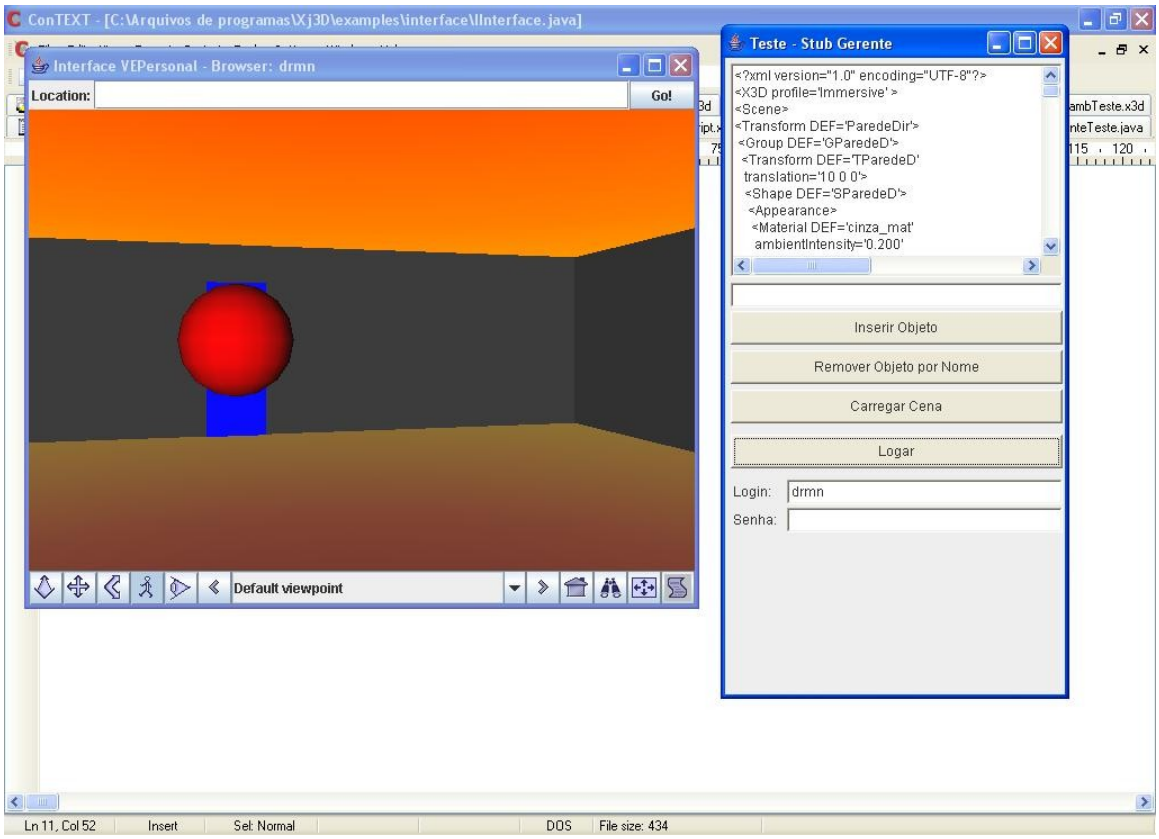

Figura 4.3 – Cena carregada a partir do código do Anexo 3, após a simulação do login no sistema

```
<Transform DEF='dad_GROUND2'>
 <Group DEF='GROUND2'>
  <Transform DEF='dad_Sphere2' translation='-2 0 0' rotation='0 1 0 0'>
   <Shape DEF='Sphere2'>
    <Appearance>
     <Material DEF='green_mat' ambientIntensity='0.200' shininess='0.200'
diffuseColor='0 1 0'/>
    </Appearance>
    <Sphere radius='1.000'/>
   </Shape>
  </Transform>
  <TouchSensor DEF='Touch2'/>
 </Group>
</Transform>
```

Figura 4.4 – Código XML referente a um objeto a ser inserido

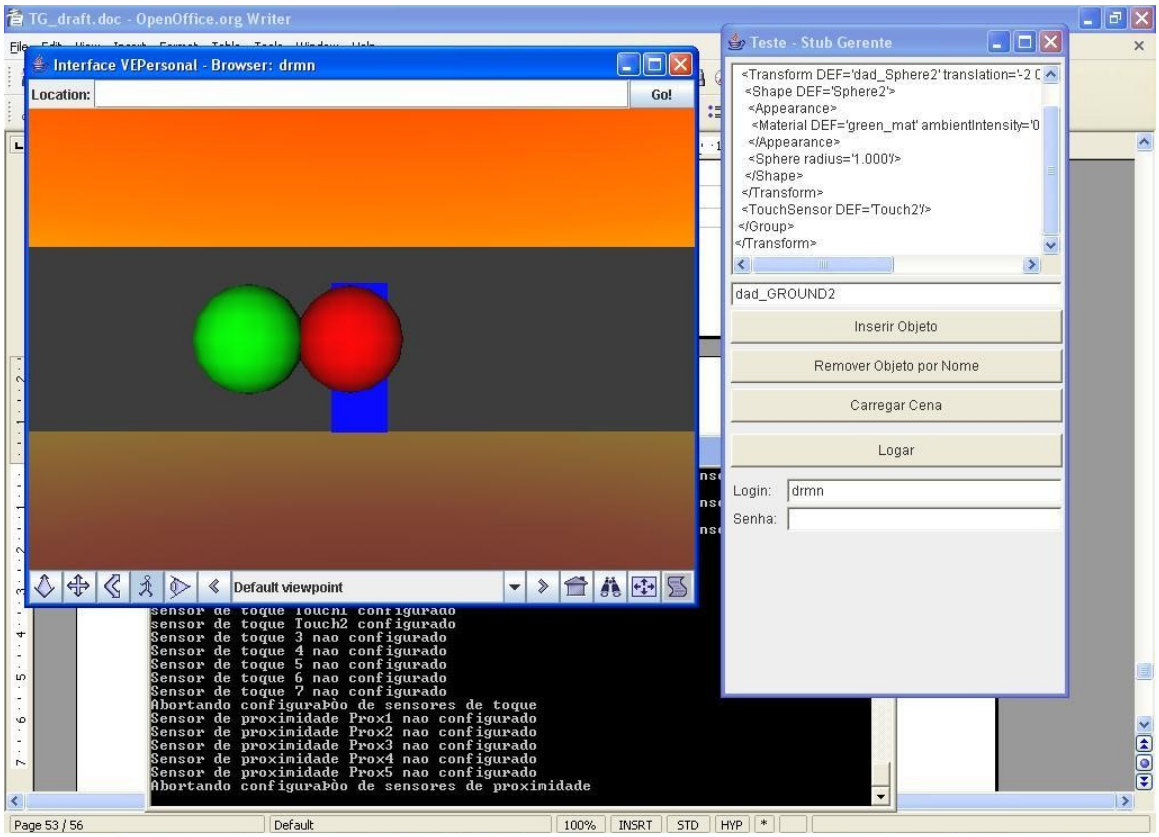

Figura 4.5 – Cena recarregada após a inserção do objeto da Figura 4.4

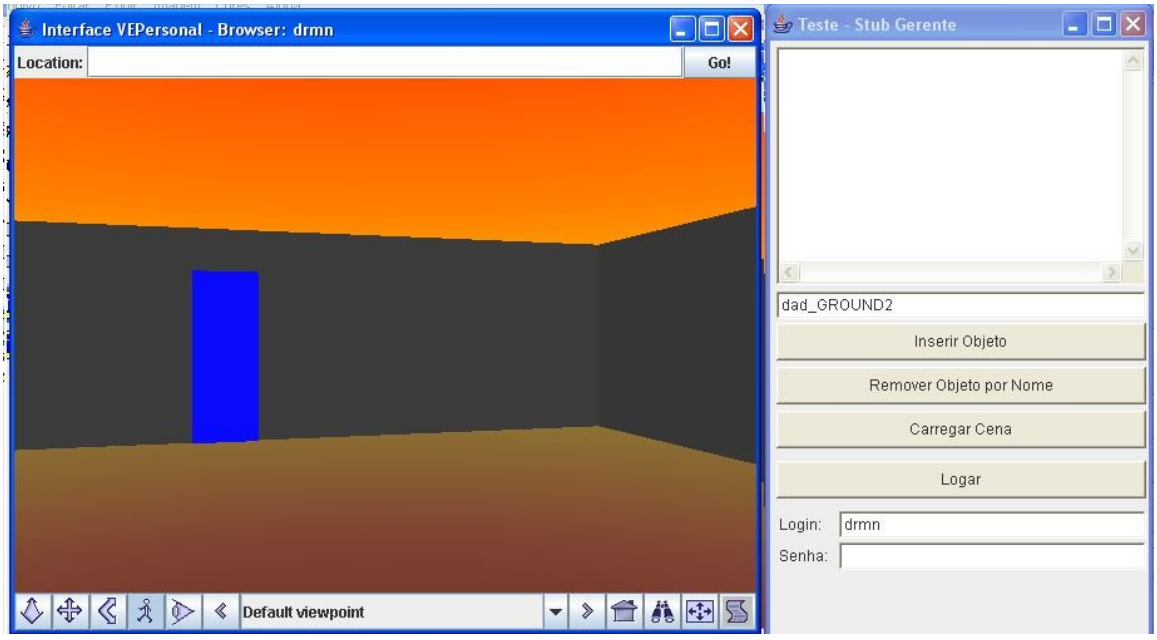

Figura 4.6 – Cena após a remoção dos objetos de nomes “Esfera” e “dad_GROUND2”

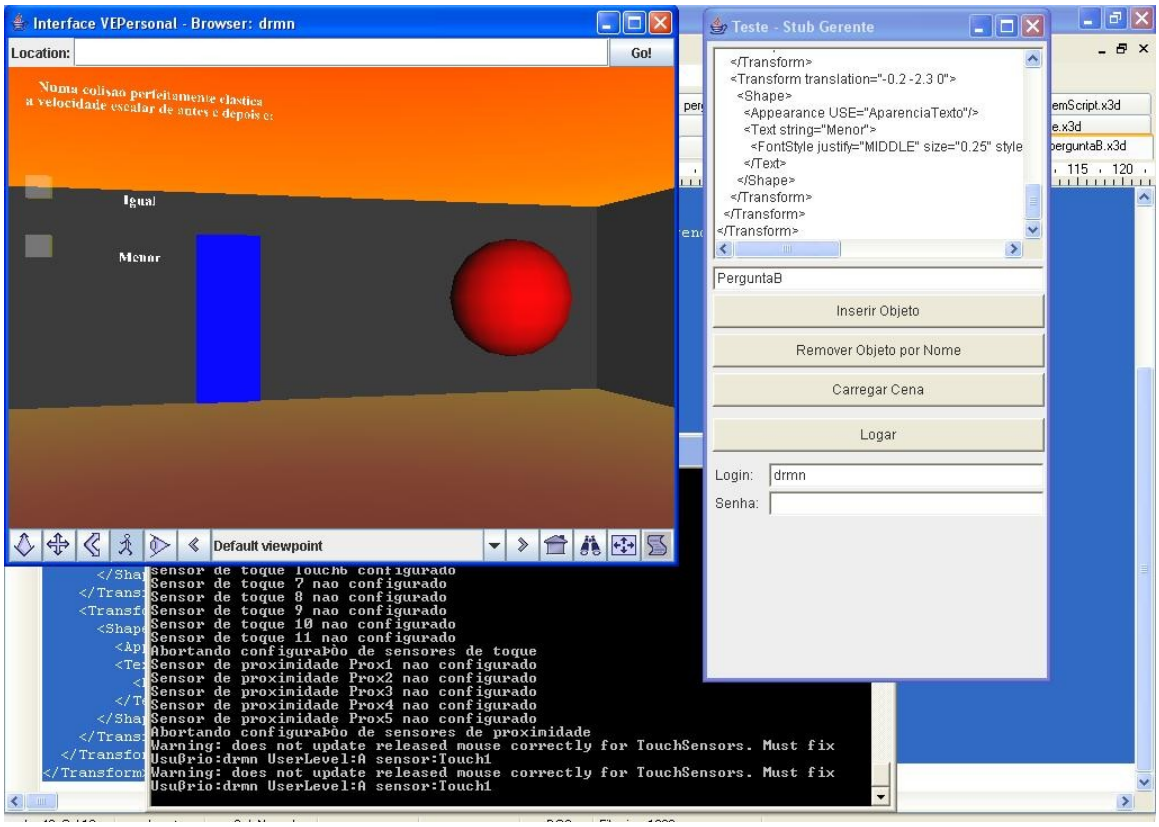

Figura 4.7 – Cena original da Figura 4.3 com a inserção de um objeto "PerguntaB" (Figura 4.8)

```
<Transform DEF='PerguntaB'>
  <Transform DEF="Pergunta1" translation="-1 3 0.5">
    <Transform>
      <Shape>
        <Appearance DEF="AparenciaTexto">
          <Material diffuseColor="0 0 0" emissiveColor="1 1 1"/>
        </Appearance>
        <Text string="&quot;Numa colisao perfeitamente elastica&quot;, &quot;a
velocidade escalar de antes e depois e:&quot;">
          <FontStyle horizontal="true" justify="MIDDLE" leftToRight="true" size="0.25"
style="BOLD" topToBottom="true"/>
        </Text>
      </Shape>
    </Transform>
    <Transform DEF="Botao1" translation="-1.5 -1.5 0">
      <Shape>
        <Appearance DEF="BotaoCor">
          <Material diffuseColor="0 0 1" emissiveColor="1 1 0" transparency="0.7"/>
        </Appearance>
        <Box size="0.3 0.3 0.1"/>
      </Shape>
      <TouchSensor DEF='Touch5'/>
    </Transform>
    <Transform DEF="Botao2" translation="-1.5 -2.3 0">
      <Shape>
        <Appearance USE="BotaoCor"/>
        <Box size="0.3 0.3 0.1"/>
      </Shape>
      <TouchSensor DEF='Touch6'/>
    </Transform>
    <Transform translation="-0.2 -1.5 0">
      <Shape>
        <Appearance USE="AparenciaTexto"/>
        <Text string="Igual">
          <FontStyle justify="MIDDLE" size="0.25" style="BOLD"/>
        </Text>
      </Shape>
    </Transform>
    <Transform translation="-0.2 -2.3 0">
      <Shape>
        <Appearance USE="AparenciaTexto"/>
        <Text string="Menor">
          <FontStyle justify="MIDDLE" size="0.25" style="BOLD"/>
        </Text>
      </Shape>
    </Transform>
  </Transform>
</Transform>
```

Figura 4.8 – Código X3D do objeto de nome “PerguntaB” inserido na cena da Figura 4.7

5. Conclusões

O crescimento e disseminação da internet têm viabilizado cada vez mais a utilização de Ambientes Virtuais para os mais diversos propósitos. Os ambientes que agregam adaptatividade propiciam a visualização de ambientes mais interessantes para os usuários, a depender das peculiaridades de cada um. No entanto, os ambientes atuais ainda apresentam algumas limitações, sendo as mais importantes a impossibilidade de apresentação de objetos em vários níveis de detalhamento e a impossibilidade de se modificar o ambiente em tempo real numa mesma sessão.

Este trabalho teve como objetivo a construção de uma interface para uma infra-estrutura de Ambientes Virtuais Adaptativos já em desenvolvimento denominada VEPersonal [Aquino, 2005], para que se pudesse superar os problemas supra mencionados. Neste contexto, foi construído o protótipo abordado, o qual agregou serviços de comunicação de ações dos usuários e de modificação dos objetos dos mundos virtuais. O caso de teste descrito na Seção 4.4 demonstra que esse módulo pode ser um elemento importante para a implementação de ambientes virtuais adaptativos.

Como trabalho futuro, pode ser citada uma integração do módulo com o restante do VEPersonal, quando este estiver pronto, testando inclusive diferentes métodos de comunicação com o mesmo e fazendo medições de performance para se avaliar a adequação do Xj3D ao projeto. Outra possibilidade seria a adoção do *applet* para que seja possível a visualização dos ambientes de dentro de uma página Web, assim que o Xj3D estivesse pronto e com os *bugs* resolvidos. Além disso, poderia ser dada ao usuário a possibilidade de fazer alguns tipos de alterações no ambiente por conta própria, o que abriria muitas possibilidades em

termos de interação com os ambientes.

Durante a implementação desta interface surgiram algumas dificuldades que impediram que mais recursos fossem adicionados à mesma. A primeira delas diz respeito à inexperiência do aluno em lidar com X3D. A segunda se refere a dificuldades na realização de testes, ocasionadas pela pouca disponibilidade de mundos virtuais descritos em X3D e o fato de que, não apenas o ambiente ao qual a interface deve ser integrada, mas também as ferramentas utilizadas na confecção da mesma, ainda estão em fase de construção e, sendo tecnologias recentes, a literatura sobre as mesmas ainda é bastante escassa.

Apesar dos testes preliminares realizados, somente quando o sistema VEPersonal estiver pronto é que se poderá avaliar se o protótipo implementado atende a todas as necessidades do sistema, abrindo a possibilidade de fazer eventuais ajustes que venham a ser necessários.

6. Referências

- [Aquino, 2005] AQUINO, M. S. Uso de técnicas de Gerenciamento de Objetos 3D e de Agentes Inteligentes para Melhora da Usabilidade de Ambientes Virtuais Adaptativos. Exame de Qualificação e Proposta de Tese. CIN/UFPE, Recife – PE, 2005.
- [Aquino et al., 2006] AQUINO, M. S.; SOUZA, F. F., FRERY, A. C. Adaptive Virtual Environments for Web3D: Contents Presentation in accordance with User Profile (a ser submetido ao periódico Interacting With Computers).
- [Bullard, 2003] BULLARD, LEN. Extensible 3D: XML Meets VRML. Disponível em < http://www.xml.com/pub/a/2003/08/06/x3d.html >. Acesso em 25 de Setembro de 2006.
- [Chittaro e Ranon, 2000] CHITTARO L.; RANON R. Adding Adaptive Features to Virtual Reality Interfaces for E-Commerce. Proceedings of AH-2000: International Conference on Adaptive Hypermedia and Adaptive Web-based Systems, Lecture Notes in Computer Science 1892, Springer-Verlag, Berlin, pp. 86-97. 2000.
- [DirectX, 2006] DirectX Microsoft Corporation. Disponível em < http://www.microsoft.com/windows/directx/default.aspx>. Acesso em: 3 de Outubro de 2006.
- [Java, 2006] Java Technology. Disponível em: < http://www.sun.com/java/about/ >. Acesso em: 3 de Outubro de 2006.
- [Java3D, 2006] Sun Microsystems Java 3D Engineering Team. Java 3D API Tutorial. Java3D API 1.3.1. Disponível em: < http://java.sun.com/developer/onlineTraining/java3d >. Acesso em: 3 de Outubro de 2006.
- [JDOM, 2006] JDOM. Disponível em <http://www.jdom.org>. Acesso em

3 de Outubro de 2006.

- [Martins, 2006] MARTINS, N. M. P. X3D. Disponível em < http://www.sal.ipg.pt/user/estg/martins/x3d.htm >. Acesso em 25 de Setembro de 2006.
- [OpenGL, 2006] OpenGL API. Silicon Graphics. Disponível em < http://www.sgi.com/products/software/opengl.> Acesso em: 3 de Outubro de 2006.
- [Osório et al., 2004] OSORIO, F.S.; MUSSE, S. R.; SANTOS, C. T.; HEINEN, F.; BRAUN, A.; SILVA, A. T. Ambientes Virtuais Interativos e Inteligentes: Fundamentos, Implementação e Aplicações Práticas. Tutorial in JAI – Jornada de Atualização em Informática/SBC. Salvador, Bahia, 2004.
- [ParallelGraphics, 2006] ParallelGraphics. Disponível em <http://www.parallelgraphics.com >. Acesso em 3 de Outubro de 2006.
- [Pollo, 1997] POLLO, L. F. Software para a Geração Automática de Modelos 3D em VRML. Disponível em < http://www-usr.inf.ufsm.br/~pollo/TG/ >. Acesso em 4 de Outubro de 2006.
- [Raposo et al., 2004] RAPOSO, A.B.; SZENBERG, F; GATTASS, M.; CELES, W. Visão Estereoscópica, Realidade Virtual, Realidade Aumentada e Colaboração. Tutorial in JAI – Jornada de Atualização em Informática/SBC. Salvador, Bahia, 2004.
- [VRML, 1997] VRML97 Functional specification - ISO/IEC 14772-1:1997. Disponível em <http://www.web3d.org/x3d/vrml >. Acesso em 4 de Outubro de 2006.
- [Wikipedia, 2006] Wikipedia. Disponível em <http://en.wikipedia.org/>. Acesso em 25 de Setembro de 2006.

- [Wikipedia, 2006b]. Realidade Virtual – Wikipédia. Disponível em <http://pt.wikipedia.org/wiki/Realidade_virtual>. Acesso em 25 de Setembro de 2006.
- [Web3D, 2006] What is X3D? Disponível em <http://www.web3d.org/about/overview/>. Acesso em 26 de Setembro de 2006.
- [X3D, 2004] X3D Overview. Disponível em < http://www.web3d.org/x3d/overview.html >. Acesso em 27 de Setembro de 2006.
- [XJ3D, 2006] The Xj3D Project. Disponível em < http://www.xj3d.org >. Acesso em 27 de Setembro de 2006.

7. Anexos

ANEXO 1: Código-Fonte do protótipo

```
import java.awt.*;
import java.util.*;
import javax.swing.*;
import org.web3d.x3d.sai.*;
import java.io.*;
import org.jdom.*;
import org.jdom.input.SAXBuilder;

public class Browser extends JFrame implements X3DFieldEventListener, IInterface {
    private X3DScene mainScene;
    private ExternalBrowser x3dBrowser;
    private X3DProtoInstance instanciaPergunta;

    //atributos que deverão ser inicializados no login
    private PerfilUsuario perfilUsuario;
    private IStubGerente stubGerente;

    //essa hashtable deve ser composta de [nome obj] [codigo xml]
    //será útil na remoção de objetos
    private Hashtable objetosAInserir;
    //adicionar objetos aqui sempre que
    //se inicializar os sensores - ver readableFieldChanged()
    private Hashtable eventosSensores;
    //essa hash se faz importante para que se possa garantir que
    //todas as vezes que se recarregar a cena pela string original,
    //se possa também remover esses objetos
    private Hashtable objetosARemover;

    private SFTime ttime;

    private String xmlCena;
    private String xmlCenaOriginal;
    //variável usada para configurar os sensores de toque recursivamente - ver
configurarSensorToque
    private int contadorSensoresToque;

    //deve inicializar todas as hashes e o perfil do usuário
    public Browser(String xmlCena, IStubGerente stubGerente) {
```

```
        System.setProperty("x3d.sai.factory.class",
"org.web3d.j3d.browser.X3DJ3DBrowserFactoryImpl");
        //resetar quantidade de sensores de toque
        contadorSensoresToque = 0;

        this.setDefaultCloseOperation(EXIT_ON_CLOSE);

        HashMap requestedParameters = new HashMap();
        X3DComponent x3dComp = BrowserFactory.createX3DComponent(requestedParameters);
        JComponent x3dPanel = (JComponent)x3dComp.getImplementation();
        x3dPanel.setSize(600, 500);
        getContentPane().add(x3dPanel, BorderLayout.CENTER);

        this.setTitle("Interface VEPersonal - Browser");
        this.stubGerente = stubGerente;

        objetosAInserir = new Hashtable();
        objetosARemover = new Hashtable();

        x3dBrowser = x3dComp.getBrowser();
        this.setSize(600,500);
        this.setVisible(true);

        //verifica se a cena é um XML bem-formado
        if (!isXMLWellFormed(xmlCena)) {
            //se não for, deve tentar carregar a cena em formato x3dv
            try {
                mainScene = x3dBrowser.createX3DFromString(xmlCena);
                x3dBrowser.replaceWorld(mainScene);
            } catch (Exception e) {
                e.printStackTrace();
            }
        } else {
            //só seta o atributo xmlCena se for em formato XML
            this.xmlCena = xmlCena;
            this.xmlCenaOriginal = xmlCena;
            //seta a string XMLCena no ambiente
            mainScene = x3dBrowser.createX3DFromString(this.xmlCena);
            x3dBrowser.replaceWorld(mainScene);
        }
```

```
        configurarSensores();
    }

    public IStubGerente getStubGerente() {
      return this.stubGerente;
    }

    public void setStubGerente(IStubGerente stubGerente) {
      this.stubGerente = stubGerente;
    }

    public boolean isXMLWellFormed(String xml) {
        boolean retorno = false;
        if (xml == null || xml.equals("")) return false;
        byte[] arrayBytes = xml.getBytes();
        ByteArrayInputStream is = new ByteArrayInputStream(arrayBytes);
        SAXBuilder builder = new SAXBuilder();
        try {
            Document doc = builder.build(is);
            retorno = true;
        } catch (Exception e) {
            System.out.println("xml nao eh bem-formado" + "\n" + e.getMessage());
            return false;
        }
        return retorno;
    }

    //método definido em X3DFieldEventListener, responsável pela
    //resposta a eventos gerados na cena
    //foi usada uma hashtable para não limitar o numero de respostas
    public void readableFieldChanged(X3DFieldEvent evt) {
        Object udata = evt.getData();
        if (eventosSensores.containsKey("Touch" + udata.toString())) {
            if (stubGerente != null && perfilUsuario != null)
stubGerente.enviarEventoSensor(perfilUsuario, udata.toString());
        } else if (eventosSensores.containsKey("Prox" + udata.toString())) {
            if (stubGerente != null && perfilUsuario != null)
stubGerente.enviarEventoSensor(perfilUsuario, udata.toString());
        } else if (eventosSensores.containsKey(udata.toString())) {
            if (stubGerente != null && perfilUsuario != null)
stubGerente.enviarEventoSensor(perfilUsuario, udata.toString());
        } else {
```

```
        System.out.println("Evento de sensor não configurado");
    }
}

//configura os sensores "TouchSim", "TouchNao" e "TouchSair"
private void configurarToquePadrao(String nomeSensor) {
  X3DNode noToque;
  SFTime ttime;
  try {
    noToque = mainScene.getNamedNode(nomeSensor);
  } catch (Exception e) {
    System.out.println("Sensor de toque " + nomeSensor + " nao configurado");
    noToque = null;
  }
  if (noToque != null) {
    ttime = (SFTime) noToque.getField("touchTime");
    ttime.addX3DEventListener(this);
    ttime.setUserData(nomeSensor);
    eventosSensores.put(nomeSensor, nomeSensor);
    System.out.println("sensor de toque " + nomeSensor + " configurado");
  }
}

private void configurarSensores() {
    this.contadorSensoresToque = 0;
    //sempre zerar a hash de eventos de sensores
    this.eventosSensores = new Hashtable();

    //antes de tudo, configurar sensores de toque padrão "sim"/"não"/"sair"
    //esses sensores devem enviar mensagens diferenciadas para o Gerente
    configurarToquePadrao("TouchSair");
    configurarToquePadrao("TouchSim");
    configurarToquePadrao("TouchNao");

    //configuração dos sensores de toque
    //para este método, os nomes dos sensores de toque
    //devem ser limitados a "Touch1", "Touch2", etc., em ordem
    X3DNode noToque;
    SFTime ttime;
    int contadorToques = 0;
    int contadorErros = 0;
```

```
         while(true) {
            contadorToques++;
            try {
              noToque = mainScene.getNamedNode("Touch" + contadorToques);
              contadorErros = 0;
            } catch (Exception e) {
              noToque = null;
              System.out.println("Sensor de toque " + contadorToques + " nao
configurado");
              contadorErros++;
              if (contadorErros > 5) {
                //essa folga foi implementada para lidar com eventuais remoções de objetos
                //contendo sensores de toque - depois de 5 falhas seguidas, abortar
configuração
                System.out.println("Abortando configuração de sensores de toque");
                break;
              }
              continue;
            }

            if (noToque != null) {
                ttime = (SFTime) noToque.getField("touchTime");
                ttime.addX3DEventListener(this);
                ttime.setUserData("Touch" + contadorToques); //ttime1->event1=0
                eventosSensores.put("Touch" + contadorToques, "Touch" + contadorToques);
                //pode ser que funcione ver se na hash, em readableFieldChanged,
                //procurar por "sensor" + evt.getData(), ou fazer uma busca por valor
                System.out.println("sensor de toque " + "Touch" + contadorToques + "
configurado");
            }
         }

         //configuração dos sensores de proximidade
         //para este método, os nomes desses sensores devem ser
         //limitados a "Prox1", "Prox2", etc., em ordem
         X3DNode noProx;
         SFBool proximidade;
         int contadorProx = 0;
         contadorErros = 0;
         while(true) {
            contadorProx++;
            try {
```

```
              noProx = mainScene.getNamedNode("Prox" + contadorProx);
              contadorErros = 0;
            } catch (Exception e) {
              noProx = null;
              System.out.println("Sensor de proximidade " + "Prox" + contadorProx + " nao
configurado");
              contadorErros++;
              if (contadorErros > 5) {
                //essa folga foi implementada para lidar com eventuais remoções de objetos
                //contendo sensores de proximidade - depois de 5 falhas seguidas, abortar
configuração
                System.out.println("Abortando configuração de sensores de proximidade");
                break;
              }
              continue;
            }

            if (noProx != null) {
                proximidade = (SFBool) noProx.getField("isActive");
                proximidade.addX3DEventListener(this);
                proximidade.setUserData("Prox" + contadorProx);
                eventosSensores.put("Prox" + contadorProx, "Prox" + contadorProx);
                //pode ser que funcione ver se na hash, em readableFieldChanged,
                //procurar por "sensor" + evt.getData(), ou fazer uma busca por valor
                System.out.println("sensor de proximidade " + "Prox" + contadorProx + "
configurado");
            }
         }

         //configuração de sensores de colisão,
         //os nomes dos sensores de colisão devem ser limitados
         //a "Colisao1", "Colisao2", etc., em ordem
         X3DNode noColisao;
         SFTime tcolisao;
         int contadorColisao = 0;
         contadorErros = 0;
         while(true) {
            contadorColisao++;
            try {
              noColisao = mainScene.getNamedNode("Colisao" + contadorColisao);
              contadorErros = 0;
            } catch (Exception e) {
```

```
              noColisao = null;
              System.out.println("Sensor de colisao " + "Colisao" + contadorColisao + "
nao configurado");
              contadorErros++;
              if (contadorErros > 5) {
                //essa folga foi implementada para lidar com eventuais remoções de objetos
                //contendo sensores de colisão - depois de 5 falhas seguidas, abortar
configuração
                System.out.println("Abortando configuracao de sensores de colisao");
                break;
              }
              continue;
            }

            if (noColisao != null) {
                tcolisao = (SFTime) noColisao.getField("collideTime");
                tcolisao.addX3DEventListener(this);
                tcolisao.setUserData("Colisao" + contadorColisao);
                eventosSensores.put("Colisao" + contadorColisao, "Colisao" +
contadorColisao);
                //pode ser que funcione ver se na hash, em readableFieldChanged,
                //procurar por "sensor" + evt.getData(), ou fazer uma busca por valor
                System.out.println("sensor de colisao " + "Colisao" + contadorColisao + "
configurado");
            }
        }

        //o método abaixo não é mais usado pois não foi possível se
        //detectar por ele os nomes dos sensores de toque na cena
        /*X3DNode[] nosRaiz = mainScene.getRootNodes();
        configurarSensorToque(nosRaiz);*/
    }

    public void inserirObjeto(String nomeObjeto, String xmlObj) {
      inserirObjeto(nomeObjeto, xmlObj, true);
    }

    public void inserirObjeto(String nomeObjeto, String xmlObj, boolean
inConfigurarSensores) {
      inserirObjeto(nomeObjeto, xmlObj, inConfigurarSensores, true, true);
    }

    //o atributo inVerificarInsercaoPrevia deve ser setado como false em
```

```
recarregarAmbiente()

    public void inserirObjeto(String nomeObjeto, String xmlObj, boolean
inConfigurarSensores, boolean inVerificarInsercaoPrevia, boolean inRecarregarAmbiente) {
        if (!isXMLWellFormed(this.xmlCena)) {
            System.out.println("Cena não é XML bem-formado");
            return;
        }
        if (!isXMLWellFormed(xmlObj)) {
            System.out.println("Objeto não é XML bem-formado");
            return;
        }

        //só inserir se o nome não vier nulo
        if (nomeObjeto == null || nomeObjeto.equals("")) {
          System.out.println("Objeto a ser inserido deve ter um nome associado valido");
          return;
        } else {
          int indiceDef = xmlObj.indexOf("DEF='" + nomeObjeto + "'");
          int indiceDefAlt = xmlObj.indexOf("DEF=\"" + nomeObjeto + "\"");
          if (indiceDef < 0 && indiceDefAlt < 0) {
            System.out.println("O nome do objeto associado deve ser igual ao do primeiro
DEF");
            return;
          }
        }

        //checar se esse objeto pertence à hash de remoção
        //se sim, tirá-lo de lá
        nomeObjeto = nomeObjeto.trim();
        String codigoObjeto = (String) objetosAInserir.get(nomeObjeto);
        if (objetosAInserir.containsKey(nomeObjeto) && (codigoObjeto != null &&
this.xmlCena.contains(codigoObjeto))) {
            objetosARemover.remove(nomeObjeto);
        }

        //checar se o objeto já existe na cena antes de tentar a inserção;
        //checar na hash se já foi inserido
        if (inVerificarInsercaoPrevia) {
          if (objetosAInserir.contains(nomeObjeto)) {
             System.out.println("Ja existe um objeto com o nome especificado na cena");
             return;
          } else if (objetosAInserir.containsValue(xmlObj)) {
```

```
            System.out.println("O codigo XML ja foi inserido na cena");
            return;
          }
        }
        if (this.xmlCena.toLowerCase().contains(xmlObj.toLowerCase())) {
            System.out.println("O codigo XML ja foi inserido na cena");
            return;
        }

        //sendo bem-formado, alterar o atributo de XML da cena
        String s;
        String cenaLowerCase = this.xmlCena.toLowerCase();
        int indice = cenaLowerCase.indexOf("<scene>");
        s = xmlCena.substring(0, indice + 7) + "\n" + xmlObj + "\n" +
xmlCena.substring(indice + 7);

        //não deverá recarregar o ambiente imediatamente após cada inserção em
'processarObjetos()'
        //vai fazer isso só no fim das inserções; se houver algum erro, nenhum dos
objetos será inserido
        if (inRecarregarAmbiente) {
          try {
            mainScene = x3dBrowser.createX3DFromString(s);
            // Replace the current world with the new one
            x3dBrowser.replaceWorld(mainScene);
          } catch (Exception e) {
            System.out.println("Erro na inserção do objeto " + nomeObjeto);
            return;
          }
          this.xmlCena = s;
        } else {
          this.xmlCena = s;
        }

        //adicionar o objeto à hash de inserções
        objetosAInserir.put(nomeObjeto, xmlObj);

        //após a inserção, deve remover todos os objetos da hash de remoção
        //para que eles não sejam incluídos novamente
        Iterator iteradorObjetosARemover = objetosARemover.keySet().iterator();
        String nomeObjRemocao;
        while(iteradorObjetosARemover.hasNext()) {
```

```
            nomeObjRemocao = (String) iteradorObjetosARemover.next();
            removerObjeto(nomeObjRemocao);
        }

        if (inConfigurarSensores) configurarSensores();
    }

    public void removerObjeto(String nomeObjeto) {
        nomeObjeto = nomeObjeto.trim();
        String codigoObjeto = (String) objetosAInserir.get(nomeObjeto);
        if (objetosAInserir.containsKey(nomeObjeto) && (codigoObjeto != null &&
this.xmlCena.contains(codigoObjeto))) {
            //se a hash de objetos a inserir contiver o objeto,
            //deverá remover o nome dessa hash, e remover via SAI
            objetosAInserir.remove(nomeObjeto);
            X3DNode no = mainScene.getNamedNode(nomeObjeto);
            if (no == null) {
                System.out.println("Não há objeto com nome especificado a remover");
            } else this.mainScene.removeRootNode(no);

            //remover também o código da string da cena
            int i = this.xmlCena.indexOf(codigoObjeto);

            System.out.println("" + i + ":" +  ((int) i + codigoObjeto.length() + 1));

            this.xmlCena = this.xmlCena.substring(0, i) + "\n" + this.xmlCena.substring(i
+ codigoObjeto.length() + 1);

        } else {
            //se a hash de objetos a inserir não contiver o objeto,
            //deverá inserir o nome na hash de objetos a remover
            //e remover o objeto via SAI
            X3DNode no;
            try {
              no = mainScene.getNamedNode(nomeObjeto);
            } catch (Exception e) {
              System.out.println("Não há objeto com nome especificado a remover");
              return;
            }
            if (no == null) {
                System.out.println("Não há objeto com nome especificado a remover");
                return;
```

```
            } else {
                //deve garantir que o objeto estará na raiz da cena
                this.mainScene.removeRootNode(no);
                objetosARemover.put(nomeObjeto, nomeObjeto);
            }
        }
    }

    //sempre que for chamado, esse método deverá limpar as hashes de objetos a inserir
    //recarregar a cena via SAI e reinicializar os sensores
    public void carregarAmbiente(String xmlCena) {
        try {
          //seta a string XMLCena no ambiente
          mainScene = x3dBrowser.createX3DFromString(xmlCena);
          x3dBrowser.replaceWorld(mainScene);
        } catch (Exception e) {
          System.out.println("Erro no carregamento da cena");
          return;
        }

        this.contadorSensoresToque = 0;
        this.objetosAInserir = new Hashtable();
        this.objetosARemover = new Hashtable();
        this.xmlCena = xmlCena;
        this.xmlCenaOriginal = xmlCena;

        configurarSensores();
    }

    //insere ou remove uma coleção de objetos
    public void processarObjetos(ArrayList objetos) {
      String xmlOriginal = this.xmlCena;

      //não deverá recarregar o ambiente imediatamente após cada inserção em
'processarObjetos()'
      //vai fazer isso só no fim das inserções; se houver algum erro, nenhum dos objetos
será inserido
      if (objetos != null) {
         Iterator iteradorObjetos = objetos.iterator();
         ArrayList objetosRemocao = new ArrayList();
         Objeto objetoAProcessar = null;
         String acao = null;
```

```
          while(iteradorObjetos.hasNext()) {
            objetoAProcessar = (Objeto) iteradorObjetos.next();
            acao = objetoAProcessar.getAcaoRemocaoInclusao();
            if (acao.equalsIgnoreCase(Objeto.COD_INCLUSAO)) {
              inserirObjeto(objetoAProcessar.getNomeObjeto(),
objetoAProcessar.getTagX3D(), false, false, false);
            } else if (acao.equalsIgnoreCase(Objeto.COD_REMOCAO)) {
              //não deve remover os objetos imediatamente; deverá fazer isso após as
inserções;
              //para isso, deve garantir que nunca se vai tentar inserir e remover objetos
com mesmo nome
              //mesmo em caso de substituição de objetos na cena, deverão ter nomes
diferentes
              objetosRemocao.add(objetoAProcessar.getNomeObjeto());
            } else {
              System.out.println("Acao para objeto " + objetoAProcessar.getNomeObjeto() +
" nao cadastrada");
              continue;
            }
          }

          //recarregar o ambiente após inserções
          try {
            mainScene = x3dBrowser.createX3DFromString(this.xmlCena);
            // Replace the current world with the new one
            x3dBrowser.replaceWorld(mainScene);
          } catch (Exception e) {
            System.out.println("Erro na insercao dos objetos; verifique o codigo dos
mesmos");
            this.xmlCena = xmlOriginal;
            return;
          }

          //remover os objetos do ArrayList de remoção
          Iterator iteradorObjetosRemocao = objetosRemocao.iterator();
          while(iteradorObjetosRemocao.hasNext()) {
            removerObjeto((String) iteradorObjetosRemocao.next());
          }

          //então, configurar os sensores
          configurarSensores();

        } else return;
```

```
    }

    //método chamado quando se confirma o login de um usuário;
    public void setPerfilUsuario(PerfilUsuario perfilUsuario) {
        this.perfilUsuario = perfilUsuario;
        this.setTitle("Interface VEPersonal - Browser" + ": " + perfilUsuario.getLogin());
    }

    //método que configuraria os sensores de toque recursivamente, navegando pelo
    //grafo de cena do ambiente. porém, não foi possível ler os nomes dos sensores,
    //o que inviabilizou o uso deste método.
    private void configurarSensorToque(X3DNode[] nodes) {
        int len = nodes.length;
        int types[];

        X3DFieldDefinition[] decls;
        int dlen;
        X3DNode node;

        for(int i=0; i < len; i++) {
            node = nodes[i];
            decls = node.getFieldDefinitions();
            dlen = decls.length;
            int ftype;
            int atype;

            if (node.getNodeName().equals("TouchSensor")) {
                contadorSensoresToque++;
                //configurar esse sensor
                SFTime ttime = (SFTime) node.getField("touchTime");
                ttime.addX3DEventListener(this);
                ttime.setUserData(new Integer(contadorSensoresToque));
                //o problema é que o nome aqui não vai ser sempre igual,
                //se forem inseridos objetos com novos sensores de toque
                eventosSensores.put("toque" + contadorSensoresToque, new
Integer(contadorSensoresToque));
                System.out.println("sensor de toque " + contadorSensoresToque + "
configurado");
            }

            MFNode mfnode;
            SFNode sfnode;
```

```
            X3DNode[] snodes;

            for(int j=0; j < dlen; j++) {
                ftype = decls[j].getFieldType();
                atype = decls[j].getAccessType();

                if (atype == X3DFieldTypes.INPUT_OUTPUT || atype ==
X3DFieldTypes.INITIALIZE_ONLY) {
                    if (ftype == X3DFieldTypes.MFNODE) {
                        mfnode = (MFNode) node.getField(decls[j].getName());
                        snodes = new X3DNode[mfnode.size()];

                        mfnode.getValue(snodes);
                        configurarSensorToque(snodes);
                    } else if (ftype == X3DFieldTypes.SFNODE) {
                        sfnode = (SFNode) node.getField(decls[j].getName());
                        snodes = new X3DNode[1];

                        try {
                          snodes[0] = sfnode.getValue();
                        } catch (Exception e) {
                          System.out.println("não leu valor de nó");
                        }

                        if (snodes[0] != null) {
                            configurarSensorToque(snodes);
                        }   }    }   }   }   }
}

import java.util.*;
public interface IInterface {
  public void carregarAmbiente(String xmlCena);
  public void processarObjetos(ArrayList listaObjetos);
  public void inserirObjeto(String nomeObjeto, String xmlObj);
  public void removerObjeto(String nomeObjeto);
  public void setPerfilUsuario(PerfilUsuario perfilUsuario);
  public IStubGerente getStubGerente();
  public void setStubGerente(IStubGerente stubGerente);
}
```

```
public interface IStubGerente {
  public void enviarEventoSensor(PerfilUsuario perfilUsuario, String nomeSensor);
  public void solicitarAmbiente(PerfilUsuario perfilUsuario);
  public void confirmarMudancaUserLevel(PerfilUsuario perfilUsuario);
  public void negarMudancaUserLevel(PerfilUsuario perfilUsuario);
  public void setInterface(IInterface iInterface);
  public IInterface getInterface();
}

public class StubGerente implements IStubGerente {
  private IInterface iInterface;

  public StubGerente(IInterface iInterface) {
    this.setInterface(iInterface);
  }

  public void enviarEventoSensor(PerfilUsuario perfilUsuario, String nomeSensor) {
    System.out.println("Usuário:" + perfilUsuario.getLogin() + " UserLevel:" +
perfilUsuario.getUserLevel() + " sensor:" + nomeSensor);
    if (nomeSensor.equalsIgnoreCase("touchsair")) System.exit(0);
    if (nomeSensor.equalsIgnoreCase("proxsair")) System.exit(0);
    //nomes de sensores padrão dos botões usados para solicitar
    //ao usuário a mudança de userLevel
    //isso pode ser feito com uma pergunta no próprio ambiente,
    //com dois sensores denominados "TouchSim" e "TouchNao"
    else if (nomeSensor.equalsIgnoreCase("touchsim"))
this.confirmarMudancaUserLevel(perfilUsuario);
    else if (nomeSensor.equalsIgnoreCase("touchnao"))
this.negarMudancaUserLevel(perfilUsuario);
  }

  public void solicitarAmbiente(PerfilUsuario perfilUsuario) {
    //método que deverá ser chamado quando o usuário logar; assim que entrar na interface
    System.out.println("Usuário:" + perfilUsuario.getLogin() + " UserLevel:" +
perfilUsuario.getUserLevel() + " solicitou a exibicao do seu ambiente");
  }

  public void confirmarMudancaUserLevel(PerfilUsuario perfilUsuario) {
    //esse método não faz nada, mas deveria mandar ao gerente a
    //informação de que o usuário deseja a mudança de seu userLevel
    System.out.println("Usuário:" + perfilUsuario.getLogin() + " UserLevel:" +
perfilUsuario.getUserLevel() + " confirmou mudanca de userLevel");
  }
```

```
  public void negarMudancaUserLevel(PerfilUsuario perfilUsuario) {
    //esse método não faz nada, mas deveria mandar ao gerente a
    //informação de que o usuário não deseja a mudança de seu userLevel
    System.out.println("Usuário:" + perfilUsuario.getLogin() + " UserLevel:" +
perfilUsuario.getUserLevel() + " negou mudanca de userLevel");
  }

  public void setInterface(IInterface iInterface) {
    this.iInterface = iInterface;
  }

  public IInterface getInterface() {
    return iInterface;
  }
}

public class PerfilUsuario {
  private String login;
  private String userLevel;

  public PerfilUsuario(String login, String userLevel) {
    this.login = login;
    this.userLevel = userLevel;
  }

  public String getUserLevel() {
    return this.userLevel;
  }

  public String getLogin() {
    return this.login;
  }

  public void setUserLevel(String userLevel) {
    this.userLevel = userLevel;
  }

  public void setLogin(String login) {
    this.login = login;
  }
```

```
}

public class Objeto {
  private int id_objeto;
  private String nome_objeto;
  private String tag_X3D;
  private String remocao_inclusao;

  public static String COD_REMOCAO = "R";
  public static String COD_INCLUSAO = "I";

  public Objeto(int id_objeto, String nome_objeto, String tag_X3D, String
remocao_inclusao) {
    this.id_objeto = id_objeto;
    this.nome_objeto = nome_objeto;
    this.tag_X3D = tag_X3D;
    this.remocao_inclusao = remocao_inclusao;
  }

  public int getIdObjeto() {
    return this.id_objeto;
  }

  public String getNomeObjeto() {
    return this.nome_objeto;
  }

  public String getTagX3D() {
    return this.tag_X3D;
  }

  public String getAcaoRemocaoInclusao() {
    return this.remocao_inclusao;
  }

  public void setIdObjeto(int id) {
    this.id_objeto = id;
  }

  public void setNomeObjeto(String nome_objeto) {
    this.nome_objeto = nome_objeto;
```

```
  }

  public void setTagX3D(String tag_X3D) {
    this.tag_X3D = tag_X3D;
  }

  public void setAcaoRemocaoInclusao(String acao) {
    this.remocao_inclusao = acao;
  }
}
```

Anexo 2: Classe de teste usada para validar o sistema

```
import java.awt.*;
import java.util.*;
import javax.swing.*;
import org.web3d.x3d.sai.*;
import java.io.*;

public class GerenteTeste extends JFrame {
    private java.awt.Button button1;
    private java.awt.Button button2;
    private java.awt.Button button3;
    private java.awt.TextArea textArea1;
    private java.awt.TextField textField1;
    private java.awt.TextField textFieldLogin;
    private java.awt.TextField textFieldSenha;
    private java.awt.Button buttonLogar;
    private java.awt.Label labelLogin;
    private java.awt.Label labelSenha;
    private Browser browser;
    private IStubGerente stubGerente;
    private String xmlCena;

    public GerenteTeste() {
        initComponents();

        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setTitle("Teste - Stub Gerente");
        setSize(310, 600);
        this.setVisible(true);
    }

    private void initComponents() {
        button1 = new java.awt.Button();
        button2 = new java.awt.Button();
        button3 = new java.awt.Button();
        textArea1 = new java.awt.TextArea();
        textField1 = new java.awt.TextField();
        textFieldLogin = new java.awt.TextField();
        textFieldSenha = new java.awt.TextField();
        buttonLogar = new java.awt.Button();
```

```
labelLogin = new java.awt.Label("Login:");
labelSenha = new java.awt.Label("Senha:");


addWindowListener(new java.awt.event.WindowAdapter() {
    public void windowClosing(java.awt.event.WindowEvent evt) {
        exitForm(evt);
    }
});

button1.setLabel("Inserir Objeto");
button1.setBounds(new Rectangle(5, 225, 295, 30));
button1.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mouseReleased(java.awt.event.MouseEvent evt) {
        Clicar(evt);
    }
});

button2.setLabel("Remover Objeto por Nome");
button2.setBounds(new Rectangle(5, 260, 295, 30));
button2.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mouseReleased(java.awt.event.MouseEvent evt) {
        Remover(evt);
    }
});
button3.setLabel("Carregar Cena");
button3.setBounds(new Rectangle(5, 295, 295, 30));
button3.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mouseReleased(java.awt.event.MouseEvent evt) {
        Carregar(evt);
    }
});
buttonLogar.setLabel("Logar");
buttonLogar.setBounds(new Rectangle(5, 335, 295, 30));
buttonLogar.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mouseReleased(java.awt.event.MouseEvent evt) {
        logar(evt);
    }
});

getContentPane().setLayout(null);
```

```
        getContentPane().add(button1);
        getContentPane().add(button2);
        getContentPane().add(button3);
        getContentPane().add(buttonLogar);

        textField1.setText("");
        textField1.setBounds(new Rectangle(5, 200, 295, 22));
        getContentPane().add(textField1);
        textArea1.setText("");
        textArea1.setBounds(new Rectangle(5, 5, 295, 190));
        getContentPane().add(textArea1);
        textFieldLogin.setText("");
        textFieldLogin.setBounds(new Rectangle(55, 375, 245, 22));
        getContentPane().add(textFieldLogin);
        textFieldSenha.setText("");
        textFieldSenha.setBounds(new Rectangle(55, 400, 245, 22));
        getContentPane().add(textFieldSenha);
        labelLogin.setBounds(new Rectangle(5, 375, 45, 22));
        labelSenha.setBounds(new Rectangle(5, 400, 45, 22));
        getContentPane().add(labelLogin);
        getContentPane().add(labelSenha);

        desabilitarBotoes();

        stubGerente = new StubGerente(browser);
        browser = new Browser("<?xml version=\"1.0\" encoding=\"UTF-8\"?><X3D
profile='Immersive'><Scene></Scene></X3D>", stubGerente);
        stubGerente.setInterface(browser);
        browser.setStubGerente(stubGerente);
    }

    private void desabilitarBotoes(){
      button1.setVisible(false);
      button2.setVisible(false);
      button3.setVisible(false);
      textArea1.setVisible(false);
      textField1.setVisible(false);
    }

    private void habilitarBotoes(){
```

```
      button1.setVisible(true);
      button2.setVisible(true);
      button3.setVisible(true);
      textArea1.setVisible(true);
      textField1.setVisible(true);
    }

    private void Clicar(java.awt.event.MouseEvent evt) {
        browser.inserirObjeto(textField1.getText(), textArea1.getText());
    }

    private void Remover(java.awt.event.MouseEvent evt) {
        browser.removerObjeto(textField1.getText());
    }

    private void Carregar(java.awt.event.MouseEvent evt) {
        browser.carregarAmbiente(textArea1.getText());
    }

    private void exitForm(java.awt.event.WindowEvent evt) {
        System.exit(0);
    }

    private void logar(java.awt.event.MouseEvent evt) {
      String login = textFieldLogin.getText();
      String senha = textFieldSenha.getText();
      //para teste, no campo de senha virá o userLevel
      String userLevel = senha;
      if (userLevel == null || userLevel.equals("")) userLevel = "A";
      PerfilUsuario perfil = new PerfilUsuario(login, userLevel);
      browser.setPerfilUsuario(perfil);
      stubGerente.solicitarAmbiente(perfil);
      habilitarBotoes();
    }

    public static void main(String[] args) {
        GerenteTeste demo = new GerenteTeste();
    }
}
```

Anexo 3: Código X3D do ambiente usado no teste (Seção 4.4)

```
<?xml version="1.0" encoding="UTF-8"?>
<X3D profile='Immersive' >
<Scene>
<Transform DEF='ParedeDir'>
 <Group DEF='GParedeD'>
  <Transform DEF='TParedeD'
   translation='10 0 0'>
   <Shape DEF='SParedeD'>
    <Appearance>
     <Material DEF='cinza_mat'
      ambientIntensity='0.200'
      shininess='0.200'
      diffuseColor='0.7 0.7 0.7'
     />
    </Appearance>
    <Box
     size='0.1 5 10'
    />
   </Shape>
  </Transform>
 </Group>
</Transform>

<Transform DEF='ParedeEsq'>
 <Group DEF='GParedeE'>
  <Transform DEF='TParedeE'
   translation='-10 0 0'>
   <Shape DEF='SParedeE'>
    <Appearance>
     <Material DEF='cinza_mat2'
      ambientIntensity='0.200'
      shininess='0.200'
      diffuseColor='0.7 0.7 0.7'
     />
    </Appearance>
    <Box
     size='0.1 5 10'
    />
   </Shape>
```

```
  </Transform>
 </Group>
</Transform>

<Transform DEF='ParedeNorte'>
 <Group DEF='GParedeN'>
  <Transform DEF='TParedeN'
   translation='0 0 -5'>
   <Shape DEF='SParedeN'>
    <Appearance>
     <Material
      ambientIntensity='0.200'
      shininess='0.200'
      diffuseColor='0.2 0.2 0.2'
     />
    </Appearance>
    <Box
     size='20 5 0.1'
    />
   </Shape>
  </Transform>
 </Group>
</Transform>

<Transform DEF='PortaNorte'>
 <Group DEF='GPortaN'>
  <Transform DEF='TPortaN'
   translation='0 -0.5 -4.9'>
   <Shape DEF='SPortaN'>
    <Appearance>
     <Material
      ambientIntensity='0.200'
      shininess='0.200'
      diffuseColor='0 0 1'
     />
    </Appearance>
    <Box
     size='1.5 4 0.1'
    />
   </Shape>
  </Transform>
```

```
  <TouchSensor DEF='TouchSair'/>
 </Group>
</Transform>

<Transform DEF='BotaoPergunta' translation='4 2.9 0'>
  <Shape>
    <Text string='Pergunta'>
      <FontStyle justify='MIDDLE' size='0.35'/>
    </Text>
  </Shape>
  <Shape>
    <Box size='1.3 0.4 0.3'/>
    <Appearance>
      <Material diffuseColor='0 0 1' emissiveColor='1 1 0' transparency='0.3'/>
    </Appearance>
  </Shape>
  <TouchSensor DEF='Touch3'/>
  <TimeSensor DEF='Time3'/>
</Transform>


<Transform DEF='Esfera'>
 <Group DEF='GROUND'>
  <Transform DEF='dad_Sphere1'
   translation='-.18444 0 -.25032'
   rotation='0 1 0 0'>
   <Shape DEF='Sphere1'>
    <Appearance>
     <Material DEF='Red_mat'
      ambientIntensity='0.200'
      shininess='0.200'
      diffuseColor='1 0 0'
     />
    </Appearance>
    <Sphere
     radius='1.000'
    />
   </Shape>
  </Transform>
  <TouchSensor DEF='Touch1'/>
 </Group>
```

```
</Transform>
<Background DEF="PordoSol" groundAngle="1.309, 1.571"
      groundColor="0.1 0.1 0.0,  0.5 0.25 0.2,  0.6 0.6 0.2"
      skyAngle="1.309, 1.571" skyColor="1 0 0, 1 0.4 0, 1 0.8 0"/>

<TimeSensor DEF='Wizard'
 cycleInterval='1.000'
 loop='false'
 startTime='-1.000'
/>
<PositionInterpolator DEF='Wizard_pos0'
 key='
  0 .5 1
 '
 keyValue='
  0 0 0
  10 0 0
  0 0 0
 '
/>
<TimeSensor DEF='vizx_init'
 cycleInterval='0.100'
 loop='false'
/>
<ROUTE fromNode='vizx_init' fromField='cycleTime' toNode='Wizard' toField='startTime'/>

<ROUTE fromNode='Wizard' fromField='fraction_changed' toNode='Wizard_pos0'
toField='set_fraction'/>
<ROUTE fromNode='Wizard_pos0' fromField='value_changed' toNode='dad_Sphere1'
toField='set_translation'/>
<ROUTE fromNode='Touch1' fromField='touchTime' toNode='Wizard' toField='startTime'/>

</Scene>
</X3D>
```