FUNDAÇÃO DE ENSINO “EURÍPIDES SOARES DA ROCHA”
CENTRO UNIVERSITÁRIO “EURÍPIDES DE MARÍLIA” – UNIVEM
PROGRAMA DE MESTRADO EM COMPUTAÇÃO

ELVIS FUSCO

X-LIBRAS: UM AMBIENTE VIRTUAL PARA A LÍNGUA BRASILEIRA DE SINAIS

MARÍLIA
2004

ELVIS FUSCO

X-LIBRAS: UM AMBIENTE VIRTUAL PARA A LÍNGUA BRASILEIRA DE SINAIS

Dissertação apresentada ao Programa de Mestrado do Centro Universitário Eurípides de Marília, mantido pela Fundação de Ensino "Eurípides Soares da Rocha", para obtenção do Título de Mestre em Computação.

Orientador:
Prof. Dr. José Remo Ferreira Brega

MARÍLIA
2004

ELVIS FUSCO

X-LIBRAS: UM AMBIENTE VIRTUAL PARA A LÍNGUA BRASILEIRA DE SINAIS

Banca examinadora da dissertação apresentada ao Programa de Mestrado do Centro Universitário Eurípides de Marília, mantido pela Fundação de Ensino "Eurípides Soares da Rocha", para obtenção do Título de Mestre em Computação.

Resultado: _____

Orientador: Prof. Dr. José Remo Ferreira Brega

1º Examinador: _____

2º Examinador: _____

Marília, ____ de _____ de 2004.

Dedico este trabalho aos meus pais por terem me apoiado em todos os momentos da minha vida e especialmente à minha esposa (Alecsandra) pela sua infinita paciência e incentivo, sem os quais este trabalho não teria sido completado, e também ao meu amado filho (João Pedro).

AGRADECIMENTOS

A Deus que nos dá a vida por meio de Nosso Senhor Jesus Cristo.

À Fundação de Ensino "Eurípides Soares da Rocha", mantenedora do Centro Universitário Eurípides de Marília, em nome do Magnífico Reitor Dr. Luiz Carlos de Macedo Soares, pelo apoio e incentivo no decorrer deste trabalho.

Ao meu orientador e amigo Prof. Dr. José Remo Ferreira Brega pelo constante incentivo, sempre indicando a direção a ser tomada nos momentos de maior dificuldade, sem o qual seria impossível a realização deste trabalho.

Aos professores do Programa de Mestrado em Computação da Fundação de Ensino "Eurípides Soares da Rocha".

À Virtock Technologies, Inc. que concedeu uma licença gratuita do software Vizx3D para o desenvolvimento deste projeto.

Ao Profº Stéfano Fioravanti Netto, pela assessoria prestada.

A todos que de uma forma direta ou indireta contribuíram para a produção deste trabalho.

FUSCO, Elvis. **X-LIBRAS: Um Ambiente Virtual para Língua Brasileira de Sinais**. 2004. 156 f. Dissertação (Mestrado em Computação) – Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2004.

RESUMO

Técnicas de Realidade Virtual estão sendo utilizadas nas mais diversas áreas de pesquisa e de aplicação. Suas tecnologias possibilitam a criação de interfaces avançadas, permitindo uma interação mais natural na visualização de ambientes tridimensionais. Estas técnicas podem ser utilizadas para facilitar o entendimento, por parte de pesquisadores e deficientes auditivos, da estrutura dos sinais da Língua Brasileira de Sinais (LIBRAS). Este trabalho apresenta o projeto e o desenvolvimento do vocabulário X-LIBRAS (LIBRAS Extensível), arquitetura baseada na tecnologia XML para a representação dos sinais da LIBRAS. A partir dessa plataforma, pesquisadores e desenvolvedores de tecnologias podem utilizar esse mecanismo para a representação e intercâmbio das informações dos sinais da LIBRAS de uma maneira estruturada e auto-descritiva. A fim de validar a arquitetura proposta, foi desenvolvido um Ambiente Virtual utilizando avatares humanóides do padrão H-Anim por meio das tecnologias *Extensible* 3D (X3D) e *Virtual Reality Modeling Language* (VRML). Este Ambiente Virtual permite a visualização tridimensional dos sinais LIBRAS, de modo que o usuário possa ter uma visão ampliada e detalhada do sinal. Para deste desenvolvimento, foi proposta uma metodologia de engenharia de *software* para ambientes virtuais incluindo as fases de especificação, análise, projeto e implementação com o propósito de documentar todas as etapas do projeto.

Palavras-Chave: LIBRAS, Realidade Virtual, XML, X3D, VRML, X-LIBRAS

FUSCO, Elvis. **X-LIBRAS: Um Ambiente Virtual para Língua Brasileira de Sinais**. 2004. 156 f. Dissertação (Mestrado em Computação) – Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2004.

ABSTRACT

Virtual Reality techniques are being used in the most different researches and application areas. Their technologies make it possible to create advanced interfaces, allowing a more natural interaction int the visualization of tridimensional environment. These techniques can be used to make it easier to understand the structure of the Brazilian Sign Language (LIBRAS) by researchers and the deaf users. This work presents the project and development of the X-LIBRAS (eXtensible LIBRAS) vocabulary, a type of architecture based on XML technology for the LIBRAS signs representation. From this basis, researches and technology developers are able to use this mechanism to represent and interchange informations about LIBRAS signs in a structured and self-descriptive way. In order to validate the proposed architecture, a Virtual Environment has been developed by using humanoid avatars from H-Anim standard through Extensible 3D (X3D) and Virtual Reality Modeling Language (VRML) technology. This Virtual Environment allows the tridimensional visualization from LIBRAS signs, so that the user can have an extended and detailed view of the sign. For this development, a software engineering methodology has been proposed for the virtual environments including the especification, analysis, design and implementation phases in order to document all stages of the project.

Keywords: LIBRAS, Virtual Reality, XML, X3D, VRML, X-LIBRAS

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

2D:	Bidimensional
3D:	Tridimensional
API:	Application Programming Interface
APM:	Ações Perceptivas Mínimas
ASL:	American Sign Language
AV:	Ambiente Virtual
CAD:	Computer Aided Design
CASE:	Computer Aided Software Engineering
CLG:	Command Language Grammar
COM:	Component Object Model
CORBA:	Common Object Request Broker Architecture
DOM:	Document Object Model
DTD:	Definition Type Document
EAI:	External Authoring Interface
GUI:	Graphics User Interface
IDL:	Interface Definition Language
LIBRAS:	Língua Brasileira de Sinais
OAI:	Object Action Interface
OMT:	Object Modeling Technique
RV:	Realidade Virtual
SAI:	Scene Access Interface
SAX:	Simple API for XML
UML:	Unified Modeling Language

VRML:	Virtual Reality Markup Language
VUI:	Voice User Interface
X3D:	Extensible 3D
XJL:	Extensible Juggler Language
XML:	Extensible Markup Language
XSL:	Extensible Style Language

SUMÁRIO

INTRODUÇÃO

Recursos tecnológicos têm sido utilizados para melhorar a comunicação das pessoas deficientes, isto acontece porque a flexibilidade de sistemas computacionais torna possível o desenvolvimento de serviços especiais para usuários que tenham alguma debilidade (SHNEIDERMAN, 1998).

No âmbito da deficiência auditiva, as línguas de sinais aparecem como o meio mais natural de comunicação, e nesta área muitos esforços tecnológicos têm sido feitos para facilitar o aprendizado e a divulgação das línguas de sinais como os trabalhos de CAPOVILLA (2003) para a Língua Brasileira de Sinais (LIBRAS).

A Realidade Virtual (RV), com os seus recursos avançados de interface, trouxe para esse campo de estudo, muitas possibilidades, devido o seu realismo visual e interação dos sentidos humanos.

Segundo Kirner e Pinho (1996):

> "realidade virtual pode ser definida de uma maneira simplificada como sendo a forma mais avançada de interface do usuário de computador até agora disponível. Com aplicação na maioria das áreas do conhecimento, senão em todas, e com um grande investimento das indústrias na produção de hardware, *software* e dispositivos de E/S especiais, a realidade virtual vem experimentando um desenvolvimento acelerado nos últimos anos e indicando perspectivas bastante promissoras para os diversos segmentos vinculados com a área".

Uma vantagem importante de Ambientes Virtuais (AV) sobre outras formas de interação homem-computador é que o ambiente pode ser visualizado a partir de qualquer ângulo, à medida que vão sendo feitas alterações em tempo real.

Devido a estrutura tridimensional dos sinais, AVs podem facilitar o entendimento, aprendizado e a comunicação de deficientes auditivos na língua de sinais.

Durante uma avaliação na *Florida School* para deficientes auditivos e visuais, a utilização de um sistema computacional com recursos de RV na representação dos sinais da

ASL (Língua Americana de Sinais) aumentou a compreensão quando da troca do "somente texto" para o texto acompanhado da língua de sinais utilizando a tecnologia SigningAvatar de 17% para 67%; houve, ainda, uma melhora notável da atenção, no nível de comprometimento e no desempenho dos alunos (SIGNINGAVATAR, 2002).

Por que utilizar animações 3D com recursos de RV ao invés de vídeo? Animações 3D providenciam inúmeras vantagens sobre o vídeo, incluindo:

- O tamanho da animação é menor que o vídeo, tornando mais rápida as aplicações.
- Enquanto a geração de sentenças podem ser criadas pela montagem de animações de sinais, seqüências de vídeo devem já estar gravadas, fazendo com que a criação de novos sinais ou sentenças deva ser realizada fora do sistema.
- Num sistema de RV, o usuário pode rotacionar, navegar, diminuir a velocidade e interagir com o avatar que realiza o sinal, permitindo uma compreensão maior dos sinais manuais e das expressões faciais.

Objetivos e detalhamento da pesquisa

Este trabalho objetiva desenvolver um ambiente estruturado de informações utilizando a tecnologia XML (*eXtensible Markup Language*) chamado vocabulário X-LIBRAS que permita criar uma arquitetura padronizada de armazenamento e intercâmbio de informações de sinais da LIBRAS. A proposta deste vocabulário é criar um mecanismo unificado, padronizado e flexível que permita facilitar o desenvolvimento de ferramentas computacionais utilizando a LIBRAS.

A partir deste vocabulário, implementar um sistema de RV que permita a visualização dos sinais da LIBRAS num ambiente tridimensional facilitando o entendimento da estrutura das unidades que formam um sinal.

A fim de alcançar os objetivos propostos desta dissertação de mestrado, foram realizadas as seguintes atividades:

- Revisão bibliográfica dos temas envolvidos no projeto.
- Estudo do XML e suas tecnologias.
- Estudo das tecnologias X3D e VRML para modelagem do avatar e implementação do ambiente virtual.
- Estudo da linguagem Java para manipulação do XML e do ambiente virtual.
- Estudo da natureza da LIBRAS e seus sinais para especificação dos requisitos do sistema.
- Definição da metodologia de projeto de ambientes virtuais
- Desenvolvimento do vocabulário X-LIBRAS, contendo o DTD, *Schema* e documentos XML para a representação dos sinais LIBRAS.
- Implementação do ambiente virtual utilizando o X-LIBRAS

Organização da dissertação

Esta dissertação está dividida em 6 capítulos, descriminados a seguir:

Capítulo 1 – XML: descreve os principais conceitos da tecnologia XML, da especificação X3D e aborda a utilização dessas tecnologias em sistemas de realidade virtual.

Capítulo 2 – Língua Brasileira de Sinais - LIBRAS: aborda os conceitos e a estrutura da Língua Brasileira de Sinais, bem como, as unidades mínimas distintivas da formação dos sinais.

Capítulo 3 – Especificação H-Anim: descreve a a estrutura e o mecanismo de desenvolvimento de avatares humanóides utilizando esta tecnologia.

Capítulo 4 – Aspectos de Modelagem de Ambientes Virtuais: aborda os problemas da análise e projeto de ambientes virtuais, assim como as etapas da modelagem e metodologias existentes na área.

Capítulo 5 – Metodologia Proposta para o Projeto de Ambientes Virtuais: apresenta e propõe uma metodologia para o desenvolvimento de ambientes virtuais, destacando suas fases e detalhando os processos.

Capítulo 6 – Implementação do Projeto Proposto: Desenvolvimento do vocabulário X-LIBRAS e implementação do ambiente virtual descrevendo e discutindo os principais aspectos da especificação, projeto e implementação.

CAPÍTULO 1 – XML

Neste capítulo estarão sendo mostrados os principais conceitos do padrão XML utilizados para o desenvolvimento do trabalho, bem como o estado da arte na área de implementação de sistemas de RV baseados na plataforma XML.

1.1 O Padrão XML

XML é um acrônimo para *Extensible Markup Language* – Linguagem de Marcação Extensível – linguagem que consiste de uma série de regras que divide um documento em partes lógicas e hierárquicas. O XML é uma maneira simples e padrão de delimitar os dados do texto. (ANDERSON et al., 2001). Este mecanismo de descrição de dados é um ótimo modo de compartilhar informação via Internet porque:

- É aberto; o XML pode ser usado para trocar dados com outros usuários e programas em uma plataforma de maneira independente.
- Sua natureza auto-descritiva o torna uma opção eficaz para soluções entre desenvolvedores.
- É possível compartilhar dados entre programar sem coordenação prévia.

Aplicações de RV por sua natureza são aplicações complexas devido à natureza do conteúdo de um grafo de cena e à grande quantidade de informações que são geradas pelo AV. Se a Internet for esse ambiente de execução da aplicação de RV, a arquitetura de armazenamento e a transferência dessas informações devem ser as mais simples e seguras possíveis, além de garantir acesso a plataformas abertas. O formato de um documento XML (W3C CONSORTIUM, 2003) possibilita essa característica, pois expressa de uma maneira simples e padrão a delimitação das informações do documento, facilitando, assim, a

transmissão e o processamento dos dados nele inseridos e ainda propondo a integração com tecnologias não proprietárias.

O XML permite agregar semântica ao conteúdo dos documentos, deixando por conta de cada aplicação a interpretação da marcação atribuída a este conteúdo. Esta abordagem amplia significativamente as possibilidades do uso das linguagens de marcação, entre elas a capacidade de definir metadados – dados que descrevem dados.

Além da maneira simples de representar as informações do ambiente, o XML ainda tem um mecanismo prático de descrever os dados no documento, isto é, um documento XML além de carregar os dados em si, leva em conjunto a descrição desses dados. Esta característica faz de uma aplicação XML um ótimo meio de compartilhar as informações com outras aplicações via Internet, pois os dados de uma aplicação de RV podem ser trocados entre usuários e outras aplicações em uma plataforma de maneira independente.

Através da natureza autodescritiva do XML, aplicações que rodem na ponta de um ambiente distribuído podem fazer a renderização visual de dados marcados em XML sem um conhecimento prévio das informações contidas nesse documento.

Segundo Anderson et al. (2001), a natureza de uma aplicação de RV sugere dois tipos de informações: as informações estáticas ou persistentes do ambiente, que são carregadas no visualizador do ambiente ao executar a aplicação, e as informações dinâmicas que são geradas através da interação com o ambiente; estas informações dinâmicas transformam-se em mensagens quando a aplicação trabalha sobre um ambiente distribuído como a Web; nesse contexto o padrão XML também se encaixa muito bem, pois, logicamente, pode representar ambos tipos de dados – documentos e mensagens – em um sistema.

1.2 Arquitetura do XML

As regras de sintaxe do XML são fundamentais para qualquer sistema baseado nessa tecnologia. Essas regras foram estabelecidas pela Recomendação XML 1.0 do W3C CONSORTIUM, e um documento que obedece a essas regras é chamado de XML bem-formado. Existem ferramentas analisadoras (*parser*) que são responsáveis por verificar se um documento obedece às regras de sintaxe do XML e identificar se um documento é bem-formado (ANDERSON et al. 2001).

Para criar um vocabulário XML específico para uma aplicação, devemos especificar as regras através das quais os documentos XML são escritos. Como qualquer desenvolvedor pode utilizar o XML para criar seu próprio vocabulário de marcação, devemos criar um mecanismo de conferência que defina as regras de um documento XML. Este mecanismo é chamado de DTD (*Definition Type Document*) e contém as regras que um desenvolvedor definiu para descrever um vocabulário de um sistema específico.

1.3 Definição de tipos de documentos

Ao projetar um documento XML para uma aplicação específica, o desenvolvedor deve definir no DTD o que é ou não permitido num documento XML. Com isso, uma aplicação deve consultar um DTD ao gerar um documento XML para outra aplicação. Da mesma maneira, a aplicação cliente, ao receber esse documento, antes de processá-lo, deve consultar o DTD para verificar se o documento é legítimo em relação às regras pré-definidas no sistema. Essa validação também é efetuada pelo *parser*; portanto, um *parser* pode tanto verificar um documento em relação às regras centrais do XML, são os chamados *parsers* de

não validação, quanto verificar um documento quanto às regras de um DTD, são chamados *parsers* de validação (REITMAYR; SCHMALSTIEG, 2001).

A Figura 1 mostra como este mecanismo, numa aplicação distribuída que gera e/ou recebe documentos XML, pode garantir a integridade das informações recebidas.

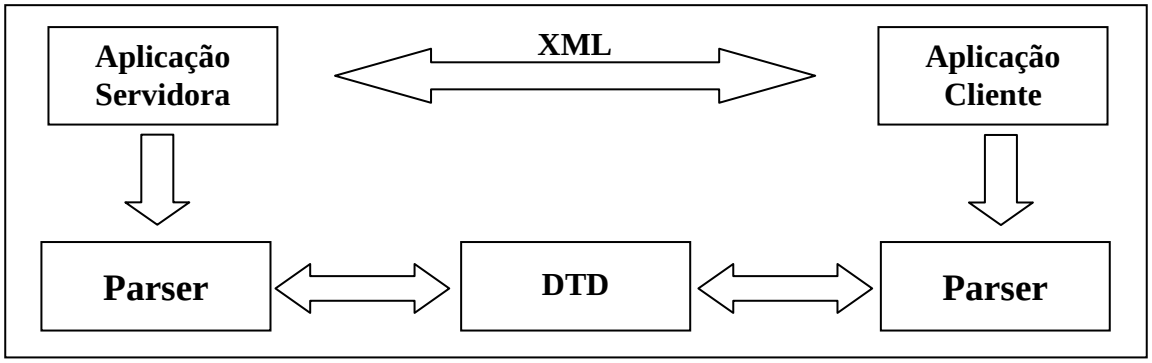

Figura 1: Arquitetura de validação de documentos XML

A criação de um DTD também segue algumas regras sintáticas que definem como escrever as regras de uma determinada classe de documentos XML.

Basicamente, um DTD precisa poder definir todos os elementos em um documento, os atributos que lhe podem ser designados e os relacionamentos possíveis entre os elementos.

```
<!ELEMENT Humanoide (Cabeca, Tronco?, Membro*)>
<!ELEMENT Cabeca (Cabelo, Olho+, Nariz, Boca)>
<!ATTLIST Cabelo cor   CDATA #REQUIRED
                 forma (LISO | ENCARACOLADO #REQUIRED>
...
```

Figura 2: Exemplo de DTD

A Figura 2 mostra a definição de um tipo de documento para expressar a forma de um humanóide; qualquer documento que se baseie neste DTD, deve ter um elemento humanóide como elemento raiz e, seguindo a hierarquia, um elemento cabeca, opcionalmente um elemento tronco e, opcionalmente, vários elementos membros. Foi definido também que o

elemento cabeca tem que ter um elemento cabelo, um ou mais elementos olhos, um elemento nariz e um elemento boca. Para o elemento cabelo foram definidos dois atributos: o atributo cor que deve estar com algum valor, e um elemento forma que deve ter o valor liso ou encaracolado. A representação gráfica deste DTD pode ser expressa como mostra a Figura 3:

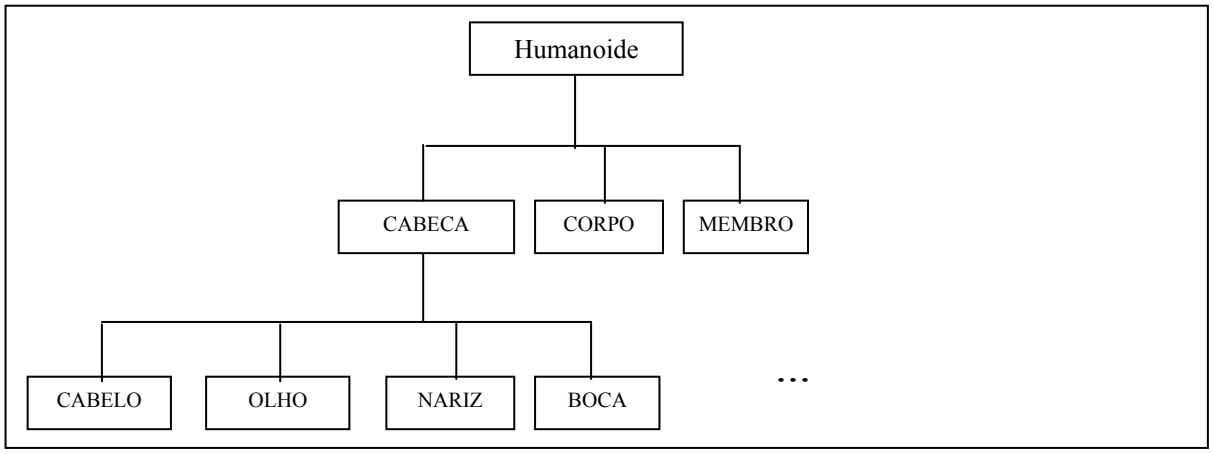

Figura 3: Hierarquia dos elementos XML

Ao criar um documento XML, o DTD é associado a esse documento; então, quando um *parser* de validação lê a instrução através da qual os documentos são associados a um DTD, isto informa ao *parser* que carregue o DTD e valide o documento de acordo com as regras ali contidas. A Figura 4 mostra como vincular um DTD a um documento XML.

Caso o *parser* não encontre no documento XML estas características, um erro acontecerá neste processo, e o documento não será validado.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Humanoide SYSTEM "humanoide.dtd">
<Humanoide>
...
```

Figura 4: Vínculo de um documento XML a um DTD

Os DTDs tem algumas limitações como sua capacidade de restrição, pois fornece pouco controle sobre os elementos e seus relacionamentos. Para resolver estas limitações o

W3C CONSORTIUM tem proposto o XML *SCHEMA* que pretende fornecer um mecanismo mais aprimorado de definir e validar documentos XML.

1.4 Documentos XML

O XML é inerentemente hierárquico, ou seja, um documento XML é definido como uma forma de uma árvore hierárquica simples, em que cada documento tem um nó raiz chamado de entidade do documento ou raiz do documento. Este nó é pai de todos os outros nós na árvore. Os nós são chamados de *elementos*, isto é, um documento XML é formado por um elemento raiz e por outros elementos secundários (ANDERSON et al. 2001).

Os elementos podem conter dados de caractere (*CDATA*) que são conteúdos de texto associados ao elemento ou a um valor de atributo. Cada elemento de um documento XML pode ser caracterizado por *atributos* que são informações que queremos anexar ao elemento. Estes atributos são formados por um conjunto de pares nome-valor.

A Figura 5 mostra um documento XML que contém entidades com dados de caractere e também com atributos para caracterizar essas entidades.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Humanoide SYSTEM "humanoide.dtd">
<Humanoide>
<Cabeca>
<Cabelo Cor="Preta" Forma="LISO"/>
</Cabeca>
...
```

Figura 5: Exemplo de um documento XML

Esta forma hierárquica de representar um documento no XML pode facilmente ser aplicada a um grafo de cena que segue as mesmas características, mostrando a compatibilidade entre o formato XML e os formatos de documentos de sistemas de RV.

1.5 Modelagem de documentos XML

Como já mencionado anteriormente, todo processo de desenvolvimento de aplicações de RV utilizando uma especificação XML, depende da utilização de um vocabulário que defina as informações e as regras que conduzem estas informações dentro do ambiente sintético.

A criação de um vocabulário XML depende do conhecimento que se tem da área específica na qual se deseja modelar, para a criação correta de documentos estruturados que representem o AV adequado.

Esse processo de modelagem inclui a compreensão da estrutura e do significado das informações presentes nos documentos, a especificação dos documentos, que é a tradução da análise de requisitos do assunto de pesquisa em um conjunto de regras ou esquema para criar os documentos, e a notação dos esquemas, que são técnicas para documentar sua especificação do documento de modo que ele seja acessível tanto para o *software* de processamento quanto para outros desenvolvedores.

Técnicas de modelagem de dados e engenharia de *software* podem ser utilizadas nesse processo, pois o resultado final do projeto estará totalmente relacionado e dependente de uma modelagem correta das necessidades e características do problema inicial.

A importância da modelagem de informações no projeto de um vocabulário XML surge da necessidade de alcançar definições absolutamente precisas sobre os dados que farão

parte dos documentos processados pelo sistema e a maneira eficaz de comunicação entre os usuários; o modelo de informações define o significado dos dados.

Como a modelagem de informações é independente de tecnologia, metodologias como o UML (*Unified Modeling Language*) (BOOCH, RUMBAUGH, JACOBSON, 2000) podem ser utilizadas para se alcançar uma precisão técnica maior do que modelagens informais.

Diferentemente de aplicações tradicionais, aplicações de RV não dividem documentos e dados. Diante deste fato, o XML vem ao encontro dessas características, pois o projetista tem a sua disposição a flexibilidade do projeto de documentos e a modelagem dos dados desses documentos dentro de uma mesma tecnologia.

Como colocado anteriormente, as informações dentro de um sistema de RV se dividem em informações estáticas e informações dinâmicas; por isso é necessária a modelagem separada desses dois tipos de informações.

As informações estáticas são aquelas que definem o grafo de cena na renderização ou carregamento inicial do AV. Para definir um modelo de informações estáticas de um sistema de RV é necessário identificar, definir e nomear entidades do grafo de cena; organizar essas entidades em uma hierarquia de classes; definir relacionamentos, cardinalidades e restrições dessas entidades e definir propriedades para identificar os detalhes dos valores associados com essas entidades (ANDERSON et al., 2001).

Uma questão a se resolver ao modelar documentos com informações persistentes é a quantidade de documentos que deve existir para um problema específico, já que, às vezes, em um único documento podem existir muitas informações referentes ao grafo de cena. Isto deve-se ao fato do XML não estar preparado para acesso direto à informação, ou seja, para renderizar uma parte de um documento XML, todo o documento deve ser carregado.

Outro problema, que aparece na modelagem de dados estáticos, é a quantidade de DTD's que representará as informações do sistema, já que não é necessário que um DTD seja uma entidade monolítica, contendo as regras de todos os objetos do sistema. Ele pode incorporar definições de outros DTD's ao utilizar entidades de parâmetros externos.

Além dessas questões, conforme Anderson et al. (2001), outros problemas têm que ser resolvidos no projeto de documentos XML, como:

- A representação de tipos de entidades;
- A representação de relacionamentos;
- A representação de atributos como elementos ou como atributos;
- A codificação de valores de propriedades.

A utilização do XML, para as mensagens que serão enviadas pelo AV numa arquitetura distribuída, oferece menos problemas de projeto do que utilizá-las para dados persistentes. Isso ocorre porque cada mensagem geralmente vem isolada de dados externos e a questão do que incluir em cada mensagem normalmente sai do modelo de processos.

1.6 Vocabulário XML

Um vocabulário XML é uma descrição de dados XML usados como meio de armazenamento e troca de informações, freqüentemente dentro de um domínio específico de atividade humana (negócios, química, matemática, computação, por exemplo).

Vocabulários XML compartilhados oferecem documentos e bases de dados mais fáceis de serem pesquisados, e uma maneira de se trocar informações entre muitas organizações e aplicações.

Um exemplo de vocabulário XML é a especificação X3D, formato padrão para representação de documentos tridimensionais na Web, este padrão é detalhado no tópico 1.9 A especificação X3D.

1.7 Manipulação de documentos XML

Depois de modelar as informações que podem compor os documentos XML, ou seja, existindo o vocabulário projetado, outros aspectos da aplicação têm que ser analisados. A definição da arquitetura do sistema pode gerar um modelo cliente/servidor, no qual o cliente faz requisições ao servidor que a partir de uma base de dados, gera um documento XML. Este é remetido à aplicação cliente interpretar, ou distribuído onde a aplicação cliente como servidora podem tanto gerar como ler documentos XML.

Estas rotinas de acesso aos documentos XML, como acessar uma parte do documento e editar suas informações, podem ser utilizadas por ferramentas de manipulação de XML que farão uso de métodos oferecidos pela API[1] do XML, cuja função é oferecer um conjunto de objetos e interfaces para a manipulação de documentos XML, o DOM (*Document Object Mode*l) (W3C DOM WORKING GROUP, 2002). Além do DOM, o W3C mantém também uma API mais simples para XML, o SAX (*Simple API for XML*) (MEGGINSON, 2002).

O DOM oferece uma visão do documento XML estruturado em árvore.

Uma aplicação *parser* desenvolvida para acessar um documento XML através do DOM, carrega todo o documento para a memória e tem à disposição uma visão de todos os objetos na memória como numa árvore.

[1] API (*Application Programming Interface*, Interface de Programação de Aplicativo): Um conjunto de rotinas que um aplicativo utiliza para solicitar serviços de baixo nível.

As principais estruturas das entidades do documento são nós na árvore objeto. Acessar esses nós e manipulá-los é uma questão de navegar na árvore usando as interfaces DOM.

Quando o documento a ser analisado é muito extenso, carregá-lo todo na memória pode sobrecarregar os recursos do computador; seria mais interessante ir carregando partes do documento conforme a necessidade. Para este cenário utilizar o SAX seria mais interessante, pois fornece este tipo de acesso a documentos XML.

O SAX é um estilo de interface bem diferente do DOM. Com o DOM, a aplicação pergunta o que está no documento ao seguir a referência do objeto na memória; com o SAX o analisador diz à aplicação o que está no documento ao notificar a aplicação de um fluxo de eventos analisadores.

De qualquer maneira, aplicações tanto servidoras quanto clientes, vão fazer uso das interfaces do DOM ou do SAX para manipular e gerar documentos XML. A Figura 6 mostra a integração do DOM dentro de um ambiente sintético utilizando a plataforma X3D.

1.8 Transformando XML

As transformações são uma das maiores facilidades introduzidas pelo XML. Permitem criar um novo documento XML a partir de um original, o que é fundamental pra converter dados de um formato para outro. Essa funcionalidade do XML é utilizada, atualmente, em interfaces de sistemas de realidade virtual que utilizam formatos não reconhecidos pelo *browser*.

Como parte de um sistema distribuído de RV, a visualização e a interação no ambiente é uma das partes mais importantes. Devido ao fato do VRML (AMES,

MORELAND e NADEAU, 1997) já estar a mais tempo no mercado, existem vários *plug-ins*[1] que funcionam em conjunto com *browsers* para executar documentos VRML na Web. Por esse motivo, muitos desenvolvedores que desenvolvem aplicações que não utilizam o VRML como plataforma, mas sim o XML, precisam converter os documentos XML para um formato legível para um *plug-in* específico, como o VRML, antes da renderização.

Esta transformação é feita através de uma linguagem de estilo para XML chamada XSL (*Extensible Style Language*) (CLARK, 1999).

O interessante sobre estes mecanismos é que o mapeamento da transformação é especificado em um documento separado ao invés de um código.

A maioria dos sistemas que utilizam o padrão X3D (WEB3D CONSORTIUM, 2002) utiliza-se destes mecanismos, já que os *plug-ins* para X3D estão ainda num estágio inicial e, muitos recursos do padrão, ainda não são reconhecidos pelos *browsers*. Aplicações que utilizam X3D devem fazer a conversão para o formato VRML antes da visualização do ambiente (CAMPOS et al. 1999).

A Figura 6 mostra este mecanismo de conversão. Neste caso, as regras de transformação estão num arquivo chamado X3dToVrml97.xsl do padrão X3D.

[1] *Plug-ins* são programas que normalmente "adicionam" novas características a um outro determinado programa.

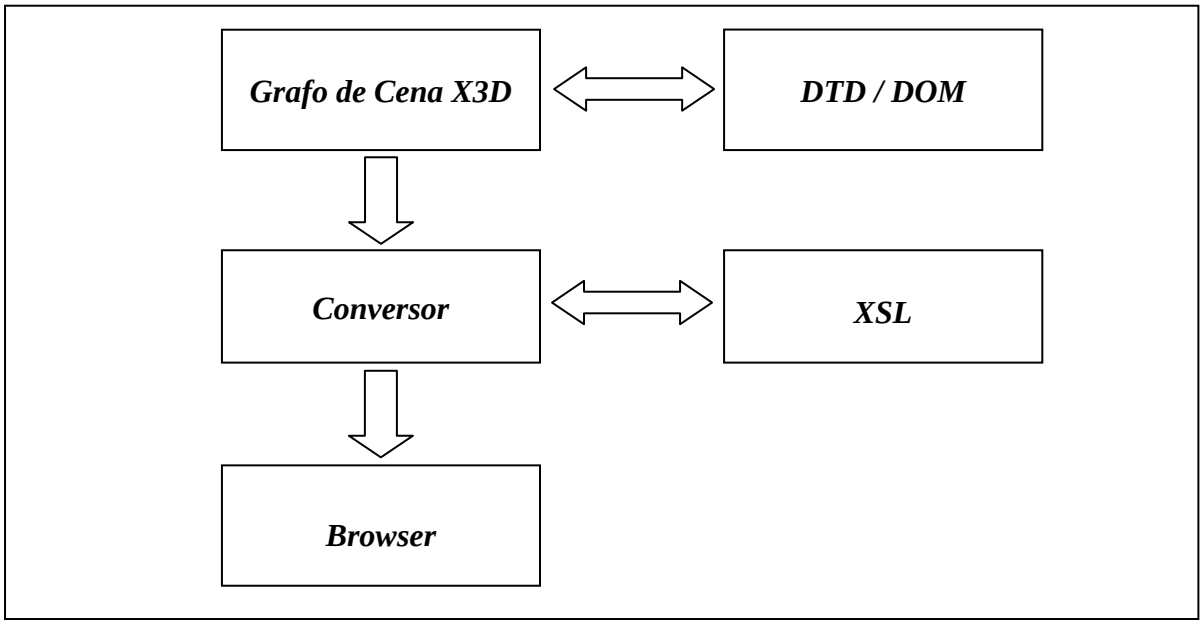

Figura 6: Visualização do grafo de cena X3D no *browser* VRML

A figura anterior mostra o grafo de cena associado a um ou mais arquivos DTD que serão utilizados pelo conversor para validar o arquivo X3D através de uma API DOM, antes de converter o arquivo X3D em um arquivo VRML através de diretivas de associação de comandos de cada linguagem contidas num arquivo XSL, o arquivo resultante, finalmente, pode ser interpretado pelo *browser*.

1.9 A especificação X3D

O formato X3D (WEB3D CONSORTIUM, 2002) tem-se mostrado uma excelente especificação na representação de capacidades de geometria e comportamento na Web, pois permite a utilização do padrão XML para codificação de sistemas de RV. O próximo passo é o desenvolvimento de ferramentas de autoria de grafos de cena e de aplicativos de visualização e navegação destes arquivos X3D. Atualmente, os *browsers* VRML não suportam o padrão X3D, e esse tem sido o desafio deste padrão. Um exemplo seria a especificação Xj3D (GOLDBERG, 1999), projeto que visa criar uma plataforma de

desenvolvimento baseada em Java3D que permita desenvolvedores carregar ambientes desenvolvidos em X3D ou ainda criar um *browser* que possibilite a navegação e a interação num mundo X3D.

O projeto Xj3D é uma iniciativa do Consortium Web3D[1] de criar uma plataforma onde desenvolvedores possam carregar seus ambientes criados na especificação X3D e também possam desenvolver novos recursos para o padrão além de proporcionar o desenvolvimento de *browsers* que suportem o formato X3D para ambientes virtuais na WEB.

Outra possibilidade é a utilização de *plug-ins* com suporte X3D nos principais *browsers* do mercado, porém, atualmente, esses *plug-ins* não conseguem representar todos os recursos da especificação X3D.

Atualmente, existem os *plug-ins* Flux Player[2], Octagon Player[3] e BS Contact[4] que permitem a visualização de ambientes desenvolvidos em X3D.

O X3D é uma especificação de grafos de cena 3D. Esta especificação pode ser implementada em inúmeras linguagens, incluindo C, C++ e Java. As principais implementações têm sido desenvolvidas em Java utilizando Java3D (BARRILEAUX, 2001). Isto não significa que o X3D necessite ser escrito em Java ou C++, ou que o X3D é restrito às implementações como um Applet Java.

[1] http://www.web3d.org
[2] http://www.mediamachines.com
[3] http://www.octaga.com
[4] http://www.bitmanagement.com

1.9.1 Arquitetura X3D

A arquitetura X3D descreve aplicações interativas e conteúdo multimídia 2D e 3D. Ela contém elementos declarativos (objetos e relacionamentos estruturados entre estes objetos) e elementos procedurais (código executável). O X3D estabelece uma hierarquia de objetos representando a cena que pode ser modificada através de uma variedade de mecanismos. O X3D inclui suporte a um grande conjunto de recursos para gráficos 2D e 3D, animações, áudio e vídeo, e navegação dentro de um espaço 3D.

1.9.2 Desenvolvendo em X3D

Tipicamente, o desenvolvedor do conteúdo X3D inicia escrevendo um arquivo X3D ou um fluxo contendo o estado inicial da aplicação, utilizando-se de comandos procedurais ou declarativos que descrevem o comportamento em todo o tempo. Um arquivo X3D pode ser puramente declarativo, com os comportamentos descritos em termos de objetos ou conexões de fluxo de dados entre estas propriedades de objetos, ou pode ser puramente procedural, com o visual e elementos comportamentais construídos dinamicamente como parte da execução da aplicação. Freqüentemente uma cena é a combinação de ambos. Um programa que implementa o ambiente de execução X3D é chamado de *browser*; um programa que gera o conteúdo X3D é chamado de gerador ou ferramente de autoria.

1.9.3 O Grafo de Cena

A estrutura fundamental do X3D é o grafo de cena, que descreve os elementos comportamentais e visíveis (Nós) e seus relacionamentos a outros nós. Um grafo de cena X3D

é uma grafo acíclico dirigido. Nós podem conter um ou mais nós filhos que também podem conter seus próprios filhos. Um nó filho pode ter mais que um nó pai.

Nós cíclicos não são permitidos; por exemplo, um nó não pode ter ele mesmo como um nó ancestral no grafo de cena. Entretanto, para este relacionamento hierárquico, nós podem referenciar cada outro nó do grafo de cena, por exemplo, um nó pode conter como um dos seus campos, um ponteiro para outro nó.

Nós podem ser também relacionados com outro nó pela criação de conexões entre seus campos (*routes*), suportando um modelo de programação de fluxo de dados. Estes relacionamentos, combinados com a habilidade de inserir código de programação no grafo de cena via *scripts*, providencia um modelo extremamente poderoso e dinâmico de execução para aplicações multimídia baseadas no tempo.

1.9.4 Perfis, componentes e níveis

O X3D é uma especificação complexa e, para gerenciar esta complexidade, foi adotada uma arquitetura modular. Em contraste com a natureza monolítica do VRML, que requer a adoção do conjunto de características em sua totalidade, o X3D permite aos desenvolvedores suportar subconjuntos da especificação (*Profiles*), composto de blocos modulares de funcionalidades (*Component*s).

Reconhecendo que nem todos os conteúdos requerem exatamente a mesma performance e qualidade visual, o X3D também permite que a especificação seja suportada com qualidades variáveis de serviço (*Levels*). Em conjunto aos perfis, os componentes e os níveis permitem um alcance de implementações para diferentes plataformas e necessidades de aplicações.

Um *Profile* é um subconjunto da especificação X3D desenvolvido para uma aplicação particular, uma plataforma ou necessidade de mercado. Cada perfil representa um conjunto de recursos e características que são comumente utilizados para juntos resolverem uma necessidade. Por exemplo, o perfil *Interchange* é utilizado para troca de conteúdo entre sistemas proprietários: ele contém todos os recursos requeridos para especificar geometria, agrupamento, propriedades de aparência e animações; entretanto não contém interatividade, controle de navegação, *scripts* e muitos outros recursos encontrados na especificação toda do X3D.

Uma descrição de perfil pode especificar as restrições em certos recursos para simplificar a implementação ou reduzir requerimentos de memória, impondo limitações em tamanho de *array* ou designando campos não suportados por um Nó. *Browsers* e ferramentas de criação podem ser consideradas conformes a um perfil particular se eles não implementam a especificação toda do X3D.

Na Figura 7 é mostrada a utilização dos *Profiles* na ferramenta X3D-Edit (BRUTZMAN, 2003). Esta ferramenta permite a criação do grafo de cena num modo gráfico, possibilitando a conversão posterior da codificação do grafo de cena para outro padrão como VRML. Nesta figura é possível notar a paleta de *Profiles* do X3D-Edit, fazendo com que ao selecionar um *Profile*, sejam disponibilizados os nós disponíveis aquele *Profile*.

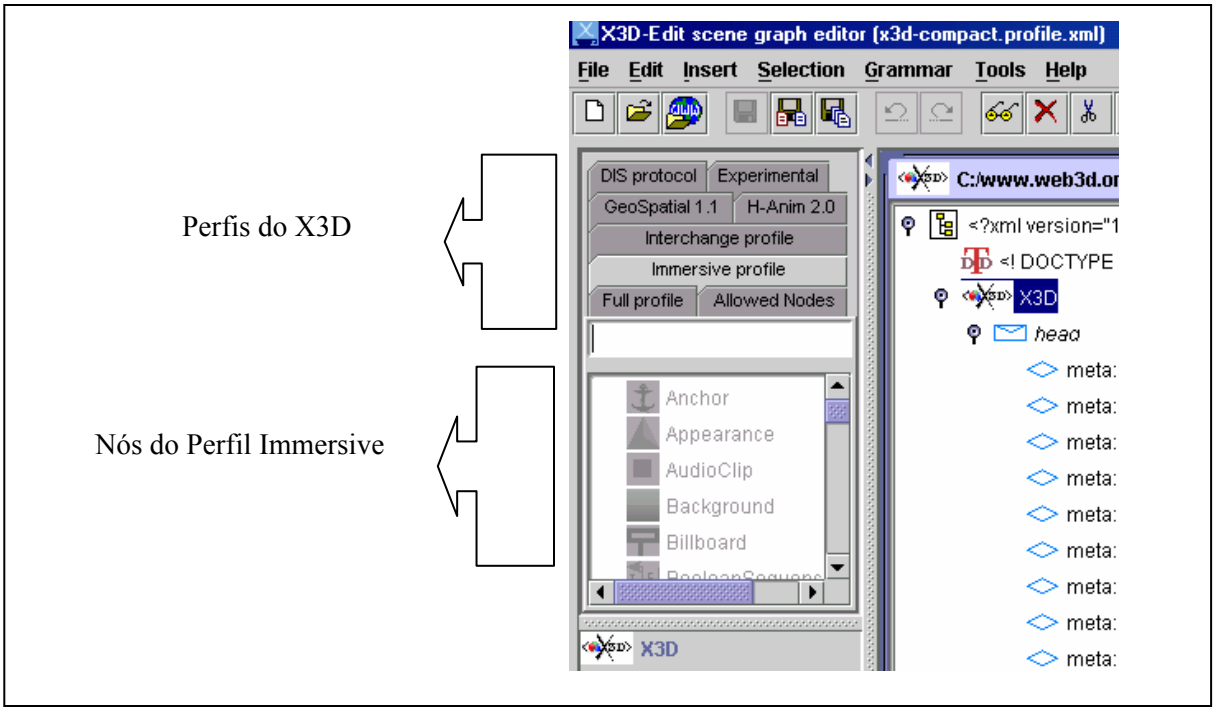

Figura 7: Hierarquia de perfis e nós no X3D-Edit

Para cada perfil selecionado, são mostrados os nós referentes ao perfil.

A capacidade do X3D é definida como distintos blocos de funcionalidade ou *Components*. Grupo de Componentes relacionam características. Exemplos de componentes são *Geometry*, *Grouping*, *Lighting*, *Texturing*, *Pointing Device Sensors* e N*avigation*. O componente é um paradigma conveniente pela identificação de blocos principais de funcionalidade do X3D. Componentes podem ser misturados e emparelhados de acordo com a necessidade específica do conteúdo; se o *browser* X3D suporta os componentes requisitados, o funcionamento é garantido.

O conteúdo e as aplicações podem especificar o perfil requerido e/ou os componentes a serem suportados, e em quais níveis de serviço. Um *browser* X3D pode usar a informação do cabeçalho do arquivo, ou o conjunto de opções de execução de uma API, para carregar somente os componentes e os perfis e/ou dinamicamente carregar novos módulos do programa, se os conteúdos de uma cena requerem níveis de componentes adicionais não

disponíveis atualmente no sistema do cliente. O *browser* usará valores padrões se um perfil, componente ou nível de suporte não forem especificados.

1.9.5 APIs X3D

O X3D provê um conjunto grande de APIs que geram acesso em tempo de execução à cena. Estas APIs permitem a geração de comportamentos dinâmicos, extensão de funcionalidades do *browser* embutido e integração com fontes externas de dados. Acesso programável pode ser interno – usado para criar elementos customizados dentro do grafo de cena – ou externo, conectado a elementos de programa fora da cena, por exemplo, em uma aplicação *host* como um *browser web*. As APIs X3D são especificadas como um conjunto de serviços independente de linguagem na IDL (*Interface Definition Language*). Serviços da API X3D são uma aproximação para muitas linguagens populares e de *script*, incluindo Java e ECMAScript, e integração com tecnologia de componentes como COM (*Component Object Model*), DOM e CORBA (*Common Object Request Broker Architecture*).

1.9.6 A Estrutura de um arquivo X3D

Conforme mostrado na Figura 8, um arquivo X3D consiste dos seguintes principais componentes funcionais: os cabeçalhos do documentos, as declarações de protótipo, o grafo de cena e as rotinas de eventos. Os conteúdos deste arquivo podem ser processados para apresentação e interação por um *browser*.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE X3D PUBLIC "http://www.web3d.org/TaskGroups/x3d/translation/x3d-
compact.dtd"
<X3D>
      <head>
            <meta name='filename' content='geometryExample.x3d'/>
            <meta name='author' content='Elvis Fusco'/>
            <meta name='created' content='17 Novembro 2003'/>
            <meta name='revised' content='17 Novembro 2003'/>
            <meta name='url'
            content='http://www.fundanet.br/x3d/example.x3d'/>
            <meta name='description' content='Exemplo utilizando o Nó
            Cylinder.'/>
            <meta name='generator' content='X3D-Edit'/>
      </head>
      <Scene>
            <Shape>
                  <Appearance>
                        <Material diffuseColor='0 .5 1'/>
                  </Appearance>
                  <Cylinder/>
            </Shape>
      </Scene>
</X3D>
```

Figura 8: Exemplo de um grafo de cena em X3D

1.9.7 Declarações X3D

Para fácil identificação de arquivos X3D, todos os arquivos X3D devem iniciar com uma declaração de arquivo XML e uma declaração de cabeçalho contendo caracteres UTF-8. Esta declaração deve conter as seguintes informações:

- Um identificador indicando que o arquivo contém uma codificação X3D;
- Um número da versão identificando a versão do X3D de acordo com o conteúdo do arquivo.;
- Uma indicação do tipo de codificação (por exemplo, XML, VRML ou binary);
- Um nome do perfil opcional com uma lista de componentes adicionais e níveis dos componentes;
- Comentários opcionais.

A forma exata do cabeçalho é específica da codificação e dependerá dos requisitos desta codificação. Entretanto, qualquer que seja a forma, deve ser legível como caracteres UTF-8 encaixados em outra estrutura sintática. Por exemplo, a codificação XML tem regras a serem seguidas, baseadas no padrão XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE X3D PUBLIC "+//ISO//DTD X3D//EN"
"http://www.web3D.org/Specifications/X3D/x3d-compact.dtd">
<X3D version="3.0" profile="Interactive">
<head>
<component name="GeoData"/>
<component name="Nurbs" level="2"/>
<meta name="description" content="Exemplo do cabeçalho da cena X3D"/>
</head>
<Scene>
<!—Nós do grafo de cena são adicionados aqui-->
</Scene>
</X3D>
```

Figura 9: Estrutura do arquivo X3D

Na Figura 9, o cabeçalho exemplo indica que o conteúdo é uma codificação XML e lista explicitamente os componentes.

Depois do cabeçalho requerido, um arquivo X3D pode conter algumas combinações das listadas a seguir:

- Um elemento *head* opcional único que pode conter informações de metadados;
- Um elemento *scene* requerido único, que serve como um Nó *root* e contém um ou mais elementos do grafo de cena;
- Alguns elementos do grafo de cena;
- Alguns elementos *ROUTE*;
- Alguns elementos *ProtoDeclare* ou *ExternProtoDeclare*.

Elementos gramaticais adicionais podem ser introduzidos pelos componentes, como algumas declarações IMPORT ou EXPORT.

Declarações são providas por vários componentes como mostrado na Figura 9. Alguns perfis podem, entretanto, não suportar todos os tipos de declarações possíveis.

1.10 A utilização do XML na RV

A especificação X3D é um exemplo prático de que o padrão XML pode ser utilizado na RV, pois já existem diversos projetos e esforços para substituir o padrão VRML pelo X3D.

Além do padrão X3D, recomendado pelo Web Consortium, outros esquemas têm sido desenvolvidos com o intuito de utilizar a estrutura do XML como plataforma de desenvolvimento de aplicações de RV nas mais diversas áreas.

O XJL (*eXtensible Juggler Language*) (GRIEPP, CRUZ-NEIRA, 2002) é um projeto que define um esquema para o desenvolvimento de ambientes sintéticos extensível, baseado no VR Juggler, projeto desenvolvido pelo *Iowa State University's Virtual Reality Application Center* que provê o desenvolvimento de aplicações orientadas a objetos baseadas em componentes. O XJL define alguns elementos fundamentais para a criação de documentos que representem ambientes sintéticos. Tais elementos são a estrutura básica de um sistema quando se quer criar um ambiente sintético imersivo. A Figura 10 mostra os elementos básicos do XJL.

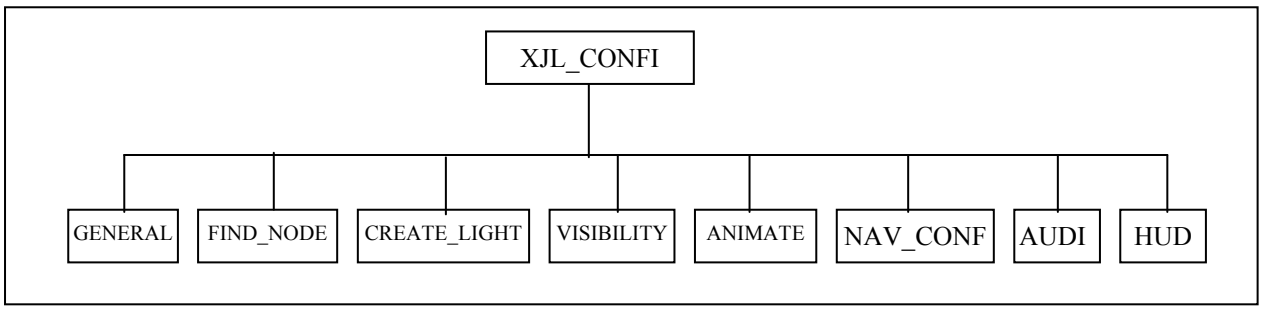

Figura 10: Elementos básicos do XJL (GRIEPP, CRUZ-NEIRA, 2002)

Esses elementos contêm outros elementos, formando uma estrutura de árvore do esquema XJL.

Este projeto mostra bem a utilização de esquemas XML para representar tecnologias já existentes, ou seja, a possibilidade de utilizar uma tecnologia que já funciona corretamente,

mesclando, no caso o VR Juggler, com as vantagens do padrão XML, criando novas oportunidades de padrões abertos.

Outra proposta que utiliza o padrão XML para descrição de ambientes virtuais é o sistema OpenTracker (REITMAR, SCHMALSTIEG, 2001), que implementa um *framework*[1] para diferentes tarefas, envolvendo dispositivos de rastreamento em aplicações de realidade aumentada. O sistema utiliza uma arquitetura orientada a objetos baseada em XML; a Figura 11 mostra o diagrama de classes da arquitetura do sistema.

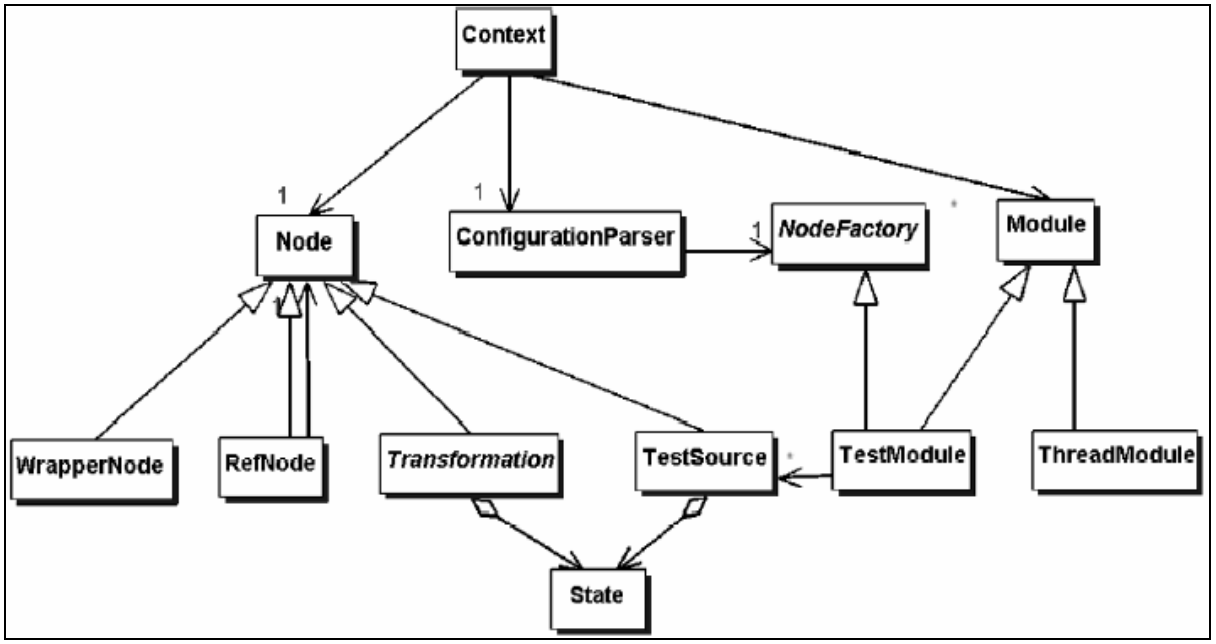

Figura 11: Arquitetura da biblioteca OpenTracker

Em Huang, Eliens e Visser (2002), é discutida a linguagem de *script* STEP, uma outra proposta que se baseia no padrão X3D para criação de um suporte computacional necessário para realizar padrões comportamentais complexos através de avatares humanóides, utilizando programação lógica distribuída.

[1] Um framework é um conjunto de classes que implementa um determinado conjunto de serviços.

Para tanto, foi desenvolvido o XSTEP, codificação XML para o STEP, que define um vocabulário com objetos definidos para a representação de velocidade, direção e partes do corpo humano. A Figura 12 mostra a especificação XSTEP para uma ação chamada *walk*.

```
<action name="walk(Agent)">
      <seq>
          <par>
              <turn actor="Agent" part="r_shoulder">
                   <dir value="back_down2"/><speed value="fast"/>
              </turn>
              <turn actor="Agent" part="r_hip">
                   <dir value="front_down2"/><speed value="fast"/>
              </turn>
              <turn actor="Agent" part="l_shoulder">
                   <speed value="fast"/>
                   <dir value="front_down2"/>
              </turn>
              <turn actor="Agent" part="l_hip">
                   <dir value="back_down2"/>
                   <speed value="fast"/>
              </turn>
          </par>
          <par>
              <turn actor="Agent" part="l_shoulder">
                   <dir value="back_down2"/>
                   <speed value="fast"/>
              </turn>
              <turn actor="Agent" part="l_hip">
                   <dir value="front_down2"/>
                   <speed value="fast"/>
              </turn>
              <turn actor="Agent" part="r_shoulder">
                   <dir value="front_down2"/>
                   <speed value="fast"/>
              </turn>
              <turn actor="Agent" part="r_hip">
                   <dir value="back_down2"/>
                   <speed value="fast"/>
              </turn>
          </par>
      </seq>
</action>
```

Figura 12: Documento XSTEP

Apaydin (2003) apresenta um projeto de VUI (*Voice User Interface*) que mapeia um conjunto de movimentos comportamentais para avatares humanóides utilizando gráficos X3D; este projeto ainda inclui um *framework* de reconhecimento de voz. Além destes projetos, podemos criar outros padrões XML para o desenvolvimento de sistemas de RV, utilizando alguns vocabulários ou perfis já existentes como o X3D.

CAPÍTULO 2 – LÍNGUA BRASILEIRA DE SINAIS - LIBRAS

> Nas mãos de seus mestres, a Língua de Sinais é extraordinariamente bela e expressiva, um veículo para atingir a mente dos Surdos com facilidade e rapidez, e para permitir-lhes comunicar-se; um veículo para o qual nem a ciência nem a arte produziu um substituto à altura. Aqueles que não a entendem falham ao perceber suas potencialidades para os Surdos, sua poderosa influência sobre o moral e a felicidade social daqueles que são privados da audição, e seu admirável poder de conduzir o pensamento a mentes que, de outro modo, estariam em perpétua escuridão. Tampouco podem avaliar o poder que ela tem sobre os Surdos. Enquanto houver dois surdos sobre a face da Terra e eles se encontrarem, haverá sinais.
> J. Schuyler Long (1910). The Sign Language

A comunicação é uma necessidade humana, e as linguagens oral e escrita são as formas mais comuns de comunicação. Por isso, pode-se dizer que: (a) a linguagem é natural do ser humano; (b) através da linguagem, o ser humano estrutura seu pensamento, traduz o que sente e o que quer, registra o que conhece, comunica-se com os outros, produz significação e sentido; (c) o ser humano cria novas linguagens para expressar o que pensa, sente, deseja e para comunicar-se com seus semelhantes (SANTAROSA, 2000).

A língua utilizada por um indivíduo para comunicação depende do grupo em que está inserido. Para os ouvintes, a comunicação se estabelece em termos oral-auditivos. No entanto, para os deficientes auditivos pode se estabelecer em termos gestual-visuais, em que gestual significa o conjunto de elementos lingüísticos manuais, corporais e faciais necessários para a articulação e a significação visual-cultural do sinal (GÓES, 1996). Nas línguas de sinais, enquanto o emissor constrói uma sentença a partir desses elementos, o receptor utiliza os olhos para entender o que está sendo comunicado. Desta forma, já que a informação lingüística é percebida pelos olhos, os sinais são construídos de acordo com as possibilidades perceptíveis do sistema visual humano (MACEDO, 1999a).

As línguas de sinais são utilizadas pela maioria das pessoas surdas. No Brasil, existem duas línguas de sinais: a língua brasileira de sinais Kaapor – LSKB, utilizada pelos

índios da tribo Kaapor, cuja maioria é surda, e a língua brasileira de sinais - LIBRAS, que é utilizada nos centros urbanos. A língua portuguesa, no caso dos deficientes auditivos, é considerada uma segunda língua (CAMPOS 2000). Em 24 de abril de 2002 a LIBRAS foi reconhecida como meio legal de comunicação e expressão das comunidades surdas do Brasil, de acordo com a lei Nº. 10.4361 (Anexo A), decretada pelo Congresso Nacional e sancionada pelo então presidente da república, Fernando Henrique Cardoso.

Santarosa (2000) afirma que "língua" designa um específico sistema de signos que é utilizado por uma comunidade para comunicação. Portanto, a LIBRAS é uma língua natural surgida entre os deficientes auditivos brasileiros com o propósito de atender às necessidades comunicativas de sua comunidade. Brito (1995) afirma que são línguas naturais porque, como as línguas orais, surgiram espontaneamente da interação entre os deficientes auditivos, além de, através de sua estrutura, poderem expressar qualquer conceito desde o descritivo/concreto ao emocional/abstrato. As línguas de sinais são dotadas de toda a complexidade e utilidade encontradas nas línguas orais e, assim como elas, possuem gramáticas próprias, com regras específicas em seus níveis lingüísticos, fonológico, morfológico e sintático. Um fator que as diferenciam é a estrutura seqüencial no tempo, onde as línguas orais são caracterizadas pela linearidade, pois os fonemas se sucedem seqüencialmente em contraste com simultaneidade das línguas de sinais, em que estes possuem uma estrutura paralela, podendo emitir sinais envolvendo simultaneamente diversas partes do corpo do sinalizador (BRITO, 1995) e (QUADROS, 1997).

Normalmente as pessoas têm idéias errôneas sobre as línguas de sinais, já que a maior parte da sociedade, que é ouvinte, associa comunicação com oralização e conseqüentemente acredita que a LIBRAS seja uma interpretação das palavras faladas, quando na verdade esta possui toda uma estrutura lingüística, assim como as línguas orais.

Quanto à utilização do termo linguagem ou língua de sinais, segundo Karnopp (1994) língua designa um sistema específico de signos que é utilizado por uma comunidade para comunicação. Linguagem, por sua vez, é tudo que envolve significação, que pode ser humano, animal ou artificial, como as linguagens de programação. Assim, não devemos nos referir à língua de sinais como "linguagem de sinais", da mesma forma como não nos referenciamos ao português por "linguagem portuguesa". Karnopp cita outras concepções inadequadas em relação à língua de sinais tais como:

Lei Nº. 10.436 (Anexo A)

· "A língua de sinais seria uma mistura de pantomima e gesticulação concreta, incapaz de expressar conceitos abstratos.";

· "A língua de sinais seria um sistema de comunicação superficial, com conteúdos restritos, sendo estética, expressiva e lingüisticamente inferior ao sistema de comunicação oral.";

· "Haveria uma única e universal língua de sinais usada por todas as pessoas surdas."

A língua de sinais não é uma língua universal e, da mesma forma que a língua oral, é diferente em vários países, podendo até mesmo apresentar sinais que variam entre regiões e entre comunidades de deficientes auditivos. Assim como cada país tem sua língua oficial, cada país também possui sua língua de sinais (MARCATO, et al. 2000).

Além disso, os sinais são considerados, por muitas pessoas, como mímicas pelo fato de alguns possuírem representações icônicas. Entretanto, esse não é o aspecto mais significante da estrutura e uso da língua. Os sinais podem ser icônicos ou arbitrários. Os icônicos reproduzem a forma ou o movimento do que se quer referir. Isso torna o sinal mais transparente e mais fácil de ser entendido. Porém, o estudo de Karnopp (1994) mostra que pesquisadores concluem que iconicidade não é relevante na determinação da forma do sinal.

Aliás, diversos processos lingüísticos e sociolingüísticos tendem a inibir a natureza icônica dos sinais, tornando-os mais arbitrários.

2.1 A LIBRAS e sua natureza 3D

As línguas de sinais são línguas naturais porque, como as línguas orais, sugiram espontaneamente da interação entre pessoas e devido à sua estrutura permitem a expressão de qualquer conceito - descritivo, emotivo, racional, literal, metafórico, concreto, abstrato - enfim, permitem a expressão de qualquer significado decorrente da necessidade comunicativa e expressiva do ser humano.

As línguas de sinais distinguem-se das línguas orais porque se utilizam de um meio ou canal visual-espacial e não oral auditivo. Assim, articulam-se espacialmente e são percebidas visualmente, ou seja, usam o espaço e as dimensões que ele oferece na constituição de seus mecanismos "fonológicos", morfológicos, sintáticos e semânticos para veicular significados, os quais são percebidos pelos seus usuários através das mesmas dimensões espaciais. Daí o fato de muitas vezes apresentarem formas icônicas, isto é, formas linguísticas que tentam copiar o referente real em suas características visuais. Esta iconicidade, mais evidente, nas estruturas das línguas de sinais do que nas orais, deve-se ao fato de que o espaço parece ser mais concreto e palpável do que o tempo, dimensão utilizada pelas línguas orais-auditivas quando constituem suas estruturas através de seqüências sonoras que, basicamente, se transmitem temporalmente. (BRITO, 2003)

Entretanto, as formas icônicas das línguas de sinais não são universais ou o retrato fiel da realidade. Cada língua de sinais representa seus refentes, ainda que de forma icônica, convencionalmente, porque cada uma vê os objetos, seres e eventos representados em seus sinais ou palavras sob uma determinada ótica ou perspectiva. Por exemplo, o sinal ÁRVORE

em LIBRAS representa o tronco da árvore através do antebraço e os galhos e as folhas através da mão aberta e do movimento interno dos seus dedos. Porém, o sinal para o mesmo conceito em CSL (língua de sinais chinesa) representa apenas o tronco com as duas mãos semiabertas e os dedos dobrados de forma circular. Em LIBRAS, o sinal CARRO/DIRIGIR é icônico porque representa o ato de dirigir, porém, é também convencional porque em outras línguas de sinais não toma necessariamente este aspecto dos referentes 'carro' e 'ato de dirigir' como motivação de sua forma, mas sim outros.

Conforme Brito (2003) a LIBRAS, como toda língua de sinais, é uma língua de modalidade gestual-visual porque utiliza, como canal ou meio de comunicação, movimentos gestuais e expressões faciais que são percebidos pela visão; portanto, diferencia da Língua Portuguesa, que é uma língua de modalidade oral-auditiva por utilizar, como canal ou meio de comunicação, sons articulados que são percebidos pelos ouvidos. Mas as diferenças não estão somente na utilização de canais diferentes, estão também nas estruturas gramaticais de cada língua.

2.2 Estrutura lingüística da LIBRAS

A LIBRAS é dotada de uma gramática constituída a partir de elementos constitutivos das palavras ou itens lexicais e de um léxico (o conjunto das palavras da língua) que se estruturam a partir de mecanismos morfológicos, sintáticos e semânticos que apresentam especificidade, mas seguem também princípios básicos gerais. Estes são usados na geração de estruturas lingüísticas de forma produtiva, possibilitando a produção de um número infinito de construções a partir de um número finito de regras. É dotada também de componentes pragmáticos convencionais, codificados no léxico e nas estruturas da LIBRAS e de princípios pragmáticos que permitem a geração de implícitos sentidos metafóricos, ironias e outros

significados não literais. Estes princípios regem também o uso adequado das estruturas lingüísticas da LIBRAS, isto é, permitem aos seus usuários usar estruturas nos diferentes contextos que se lhes apresentam de forma a corresponder às diversas funções lingüísticas que emergem da interação do dia a dia e dos outros tipos de uso da língua. (BRITO, 2003)

2.3 O Sinal e seus parâmetros

O que é denominado de palavra ou item léxico nas línguas orais-auditivas, são denominados sinais nas línguas de sinais. (BRITO, 2003)

Em Brito (2003) o sinal é formado a partir da combinação do movimento das mãos com um determinado formato em um determinado lugar, podendo este lugar ser uma parte do corpo ou um espaço em frente ao corpo. Estas articulações das mãos, que podem ser comparadas aos fonemas e às vezes aos morfemas, são chamadas de parâmetros, portanto, nas línguas de sinais podem ser encontrados os seguintes parâmetros:

1. **Configuração das mãos**: são formas das mãos, que podem ser da datilologia (alfabeto manual) ou outras formas feitas pela mão predominante (mão direita para os destros), ou pelas duas mãos do emissor ou sinalizador. Os sinais APRENDER, LARANJA e ADORAR têm a mesma configuração de mão.
2. **Ponto de articulação**: é o lugar onde incide a mão predominante configurada, podendo esta tocar alguma parte do corpo ou estar em um espaço neutro vertical (do meio do corpo até à cabeça) e horizontal (à frente do emissor). Os sinais TRABALHAR, BRINCAR, CONSERTAR são feitos no espaço neutro e os sinais ESQUECER, APRENDER e PENSAR são feitos na testa.

3. **Movimento:** os sinais podem ter um movimento ou não. Os sinais citados acima tem movimento, com exceção de PENSAR que, como os sinais AJOELHAR, EM-PÉ, não tem movimento;
4. **Orientação:** os sinais podem ter uma direção e a inversão desta pode significar idéia de oposição, contrário ou concordância número-pessoal, como os sinais QUERER E QUERER-NÃO; IR e VIR;
5. **Expressão facial e/ou corporal:** muitos sinais, além dos quatro parâmetros mencionados acima, em sua configuração têm como traço diferenciador também a expressão facial e/ou corporal, como os sinais ALEGRE e TRISTE. Há sinais feitos somente com a bochecha como LADRÃO, ATO-SEXUAL.

Na combinação destes quatro parâmetros, ou cinco, tem-se o sinal. Falar com as mãos é, portanto, combinar estes elementos que formam as palavras e estas formam as frases em um contexto.

Para conversar, em qualquer língua, não basta conhecer as palavras, é preciso aprender as regras de combinação destas palavras em frases.

2.4 Unidades mínimas distintivas

Os estudos dos parâmetros e configurações dos sinais e todas as características envolvendo o movimento com as mãos, expressões, regiões de sinalização foram estudados através de Brito (1995). A Tabela 1 mostra um resumo das unidades mínimas distintivas.

Configuração das Mãos (Um sinal pode conter uma ou mais configuração das mãos)	46 configurações, estudos de 63 configurações Eixos Referenciais OX: Fixando o centro O (que seria na parte inferior da mão ao centro) projeta-se perpendicularmente ao braço e paralelamente ao polegar estendido. OY: Paralela a palma e que se estende sobre os dedos da mão aberta.

	OZ: Projeta-se perpendicularmente à palma da mão
Ponto de Articulação (Local do corpo onde o sinal é realizado)	Divisão do Corpo (Cabeça/Tronco/Braços/Mão/ Espaço Neutro: vertical (do meio do corpo até à cabeça) e horizontal (à frente do emissor) Subdivisão (Ex: Tronco: Pescoço, ombro, busto, estômago, cintura) • Adjetivos Posição (lado esquerdo/direito, interno/externo/medial) Posição da mão: lateral, em frente, atrás Distância: próximo, distante, em contato, etc.
Movimento/Orientação	Movimentos Internos das mãos Subdivididos em Segmentos de Movimentos -> eixo Velocidade: tensão, retenção, continuidade, refreamento, repetido Tipo: retilíneo, circular, contínuo Direção: unidirecional, bidirecional, multidirecional Freqüência: simples/repetidos
Expressões Não Manuais	Cabeça e rosto Um sinal pode conter mais de uma expressão não manual Mais ou menos 16 expressões

Tabela 1: Unidades mínimas distintivas

Na Figura 13 podem ser destacados três parâmetros primários que se combinam para a constituição do sinal em LIBRAS: a **Configuração das Mãos (CM), o Movimento (M) e o Ponto de Articulação (PA)**.

Além destas características, ainda podem ser considerados, os componentes não-manuais dos sinais que são identificados como as expressões faciais, o movimento da cabeça e do corpo.

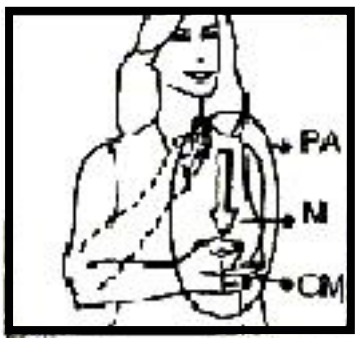

Figura 13: Parâmetros constituintes da LIBRAS. (BRITO, 2003)

A Configuração das Mãos é o modo como estão posicionados os dedos, a representação da mão, ou das duas mãos (conforme o sinal). Alguns estudos apresentam 46

configurações de mão diferentes para a LIBRAS, e elas podem ser diferenciadas quanto às posições, número de dedos estendidos, o contato e a contração (mãos fechadas ou compactas) dos dedos. A configuração da mão pode ser mantida constante durante a articulação de um sinal, ou ainda pode alterar para uma outra configuração.

Para que sejam estabelecidos pontos exatos na configuração das mãos, Brito (1995), esquematizou um estudo situando alguns eixos referenciais. Fixando um centro O (que seria na parte inferior da mão ao centro) é possível traçar a direção OX, que se projeta perpendicularmente ao braço e paralelamente ao polegar estendido, a direção OY, paralela à palma e que se estende sobre os dedos da mão aberta, e a direção OZ, que se projeta perpendicularmente à palma da mão. Além disto, ainda podem ser analisados alguns aspectos da configuração das mãos, considerando os movimentos internos realizados pelos dedos, através deste mesmo referencial OX, OY, OZ. Desta forma, são traçadas as posições relativas aos eixos da mão, juntamente com seus movimentos internos, sendo possível que as configurações sejam mais facilmente percebidas e visualizadas.

Esta forma de analisar a configuração das mãos através das posições que são traçadas ao longo destes eixos, proporciona melhor precisão na visualização da configuração, porém, já no caso do ponto de articulação isso não é tão simples assim.

O Ponto de Articulação é o local do corpo do sinalizador onde o sinal é realizado, assim uma maior especificação da posição é necessária, já que a região no espaço é muito ampla. A pessoa pode realizar o sinal no espaço próximo ao abdômen, como também pode ser um ponto ao lado do olho direito, por exemplo. Esta especificação divide o corpo das pessoas em cabeça, tronco, braços e mão, que ainda podem ser subdivididos em outras pequenas partes próximas a estas (olhos, pescoço, pulso, palma, etc.). Alguns adjetivos explicam ainda mais o ponto, são aqueles que especificam a subdivisão do corpo em questão (lado direito,

esquerdo, interno, externo, etc.) além daqueles que informam sobre o contato e a distância que podem ser realizados (imediatamente próximo, em contato, distante, etc.).

O Movimento é realizado pela mão (ou mãos) ou pelos dedos quando o sinal é produzido. Porém, é um tanto complicado fazer as observações quanto ao movimento, pois a mão é um objeto altamente assimétrico, além do que os eixos podem se deslocar simultaneamente, dificultando traçar o percurso. Mas a maior parte dos sinais podem ser subdivididos em pequenos segmentos de movimentos, a cada um dos quais pode ser relacionado um eixo. Outra característica importante para descrever o movimento é a sua velocidade que pode carregar algumas variáveis durante a realização do sinal: tensão, retenção, continuidade e refreamento. Uma característica também muito interessante, quanto ao movimento, é o fato da necessidade de repetições de sinais em algumas situações (por exemplo, para explicar mais de uma vez, ou várias coisas como no plural), onde o movimento de um sinal precisa ser reduplicado no tempo.

O Movimento pode ser analisado através do tipo, da direção, a maneira e a freqüência do sinal. O tipo seria as variações que as mãos, os pulsos e antebraços podem assumir durante o movimento. A direção pode ser unidirecional, bidirecional ou multidirecional. A maneira é em relação à qualidade, à tensão e à velocidade. E a freqüência indica se os movimentos são simples ou repetidos.

Estes parâmetros constituintes dos sinais são elementos distintivos em alguns sinais, como por exemplo, a representação em LIBRAS das palavras IGUAL e MAS são apenas diferenciadas pelo Movimento, já as palavras APRENDER e SÁBADO são diferenciadas pelo Ponto de Articulação e no caso também das palavras ESTÁTUA e DURO a Configuração das Mãos é o parâmetro diferenciador nos sinais.

Além dos parâmetros constituintes dos sinais, outros elementos complementam sua formação. São as expressões não-manuais nas línguas de sinais, componentes extremamente

importantes para a transmissão da mensagem. Muitas vezes, o sinal requer características adicionais para expressar realmente o que deseja, seja uma expressão facial, ou dos olhos, para que sentimentos de alegria, de tristeza, uma pergunta ou uma exclamação possam ser completamente representados ao receptor da mensagem. Assim, as expressões não-manuais podem assumir tanto uma função léxica quanto uma função sintática na estrutura dos sinais. Algumas expressões podem ser identificadas em regiões do corpo: rosto, cabeça, rosto e cabeça, e tronco. Estas regiões estão configuradas conforme o sinal, como por exemplo, estar com os olhos arregalados ou com a cabeça inclinando para trás. Duas expressões ainda podem ser utilizadas simultaneamente para representar algum sinal.

Um outro aspecto interessante é que muitas representações das palavras são formadas a partir de alguns componentes que podem lembrar a palavra referida, ou seja, caracterizam perfeitamente o que o deficiente auditivo está querendo dizer, mesmo que a pessoa desconheça línguas de sinais. Esta característica, muitas vezes presente na representação do sinal, é chamada da iconicidade que o sinal apresenta, a representação de aspectos da forma real do significado.

Outro ponto importante é quanto à iconicidade, que é explorada para a obtenção de efeitos gramaticais, e responsável pela transparência semântica de certos classificadores ou de certos sinais. Esta iconicidade é mais evidente nas línguas de sinais que em línguas orais, talvez pelo fato do espaço ser mais concreto e palpável. Um exemplo está na palavra DIRIGIR, que na LIBRAS é representada com as duas mãos realizando um movimento para os lados num volante imaginário; portanto, o sinal é icônico, porque representa a ação de dirigir. Porém esta representação é convencional e depende de como as pessoas estão observando o sinal.

Geralmente, é a partir das características icônicas que os sinais são produzidos próximos às partes do corpo que pertencem. O que se refere a visão é realizado perto dos

olhos; a alimentação perto da boca; a sentimentos perto do coração; ao raciocínio, perto da cabeça.

Os sinais icônicos tem a tendência de alterarem suas formas durante o tempo, mas alguns permanecem com sua identidade icônica, principalmente aqueles representados por objetos concretos ou conceitos. Embora eles possam ser utilizados por sinalizadores experientes, tem uma importância muito grande para o processamento da memória, para a aprendizagem dos sinais por iniciantes.

2.5 Análise sublexical dos sinais

Um outro estudo dos sinais foi feito por Capovilla (2003), que realizou a análise e indexação sublexical da LIBRAS que se utiliza de 5 parâmetros: mãos, dedos, localização, movimento e expressão facial.

Cada parâmetro tem um número de componentes querêmicos[1]:

I. Mãos

a) Articulação da Mão (AMDn | AMEn)

b) Orientação da palma (OPn)
1 – Cima
2 – Baixo
3 – Frente
4 – Atrás
5 – Esquerda
6 – Direita

c) Orientação da Mão (OMDn | OMCn)

d) Relacionamento entre mãos (RMn)

[1] Querema: Na linguagem vocal, a unidade básica do som é chamada fonema. A unidade básica na linguagem gestual é chamada querema. A estrutura querêmica da Libras refere-se às similaridades fisicas (formais) e diferenças entre os gestos que formam a lingua.

II. Dedos

a) Tipo do Dedo (QDn)

b) Articulação do Dedo (ADn)

III. Local

a) Local de Articulação no espaço do sinal (LAn)

IV. Movimento

a) Movimento da Mão (MMDn | MMEn)

b) Movimento do Dedo (MDDn | MDEn)

c) Movimento do Corpo (MCn)

d) Tipo de Movimento (TMn)

e) Intensidade-Frequência do Movimento (FIn)

V. Expressão Facial

a) Tipo de Expressão Facial (TEFn)

Cada querema contém um número de alocadores. Por exemplo, Orientação da Palma tem os seguintes alocadores:

- Cima
- Baixo
- Frente
- Atrás
- Esquerda
- Direita

2.6 Tecnologias e as línguas de sinais

Segundo Rocha (2000), ainda são raros no Brasil ambientes computacionais que trabalhem com a língua de sinais, porém cada vez mais este quadro vem sendo alterado.

A seguir, são citados alguns trabalhos já desenvolvidos com e sobre língua de sinais no Brasil e no exterior.

Inicialmente podem ser considerados os trabalhos do grupo do professor Fernando Capovilla (CAPOVILLA, 1996), nos quais são desenvolvidos sistemas para comunicação de deficientes auditivos utilizando sinais, textos e símbolos através de computadores em rede.

Em Capovilla (2003) são discutidos três desenvolvimentos de tecnologias em LIBRAS, citados a seguir:

I. Dicionário enciclopédico trilingue ilustrado com 1.620 páginas e 9.500 entradas em Inglês, Português e Signwriting, com classificações gramaticais, definições e exemplos do uso dos sinais, bem como milhares de ilustrações naturais e descrições das formas dos sinais através da estrutura querêmica e significados, ou seja, as referências dos sinais.

II. Enciclopédia digital com um banco de dados de cinqüenta e seis mil sinais em Português e Inglês, cada um deles descrito e ilustrado na forma e significado do sinal. A enciclopédia é composta de dois subsistemas: a) um sistema de indexação sublexical que analisa as estruturas das formas dos sinais e mostra-os como uma sequência de códigos alfanuméricos na qual as letras correspondem aos queremas e os dígitos correspondem ao seus respectivos alocadores. b) um *menu-based*, sistema de recuperação de sinais que permite aos usuários deficientes auditivos procurar e localizar sinais específicos, baseados em 5 parâmetros (mãos, dedos, localização, movimento

e expressão facial) através de seus respectivos queremas (articulação, orientação) e alocadores (i-9, A-Z).

III. Sistema de comunicação e telecomunicação face-a-face no qual usuários deficientes auditivos paraplégicos podem operar com um piscar dos olhos ou sopro. Isto lhes permite facilmente selecionar sinais animados da LIBRAS e, então, compor mensagens de sinais baseadas em LIBRAS. Uma vez composta a mensagem, o sistema habilita estes usuários enviar a mensagem através de redes de computadores. Ele também lhes permite converter as mensagens em sinais baseados em ASL (*American Sign Language*).

Destacam-se, também, muitos trabalhos desenvolvidos em vários países que fazem uso do *SignWriting* tais como o SignDic (MACEDO, 1999a), SignHTML (MAZUTTI, et al., 2001), SIGNED (CAMPOS, 2001), SIGNSIM (CAMPOS, 2001) e SIGNTALK (CAMPOS, 2001), entre outros.

No Japão, foi desenvolvida uma ferramenta que permite a tradução de vídeos falados em língua de sinais – a *Japanese SignLanguage Transcribed from Vídeo*[1]. Para tanto, é preciso que uma fita VHS esteja conectada (via televisor, por exemplo) a um computador com placa de vídeo. O *software* permite a visualização do vídeo na tela do computador e a captura de pacotes. Sendo assim, a ferramenta analisa os pacotes capturados das imagens e traduz para a língua de sinais japonesa. A Figura 14 ilustra um vídeo com o intérprete fazendo as traduções (SUTTON, 2002).

[1] http://www.signwriting.org/lessons/transcribe/scribe003.html

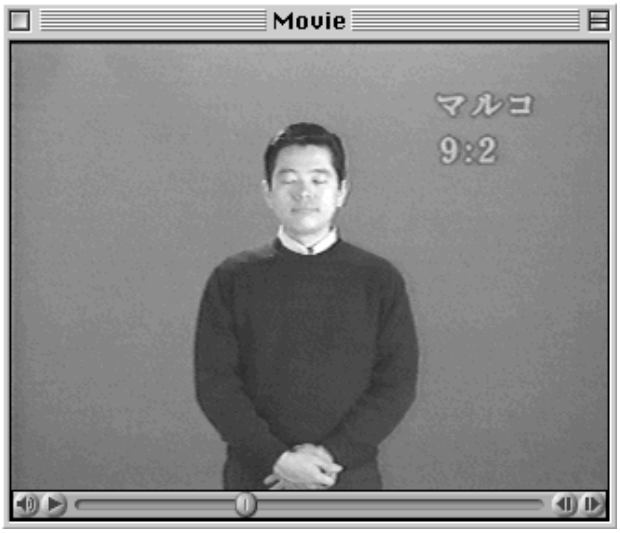

Figura 14: *Japanese SignLanguage Transcribed from Vídeo*

O ***SigningAvatar***[1] é um *software* disponibilizado em CD-ROM contendo diversas histórias escritas na língua oral, acompanhadas de figuras, cujo objetivo é facilitar o entendimento da história. Além destes recursos, há também a utilização de personagens em 3 dimensões para transmitir as histórias em ASL. Sua interface é simples, possibilitando ao usuário a escolha do modo como a informação será visualizada, palavra a palavra ou feita de acordo com o contexto das frases. O programa tem uma base de dados de sinais que correspondem às palavras das histórias. A Figura 15 apresenta uma tela de tradução do sistema feita por um personagem.

Figura 15: Sistema SigningAvatar

[1] http://www.vcom3d.com

O ***SignStream***[1] é uma ferramenta para a análise dos dados lingüísticos capturados em vídeo e foi desenvolvido para estudantes deficientes auditivos e ouvintes, professores e lingüistas que precisem traduzir língua de sinais para língua oral escrita. Essa ferramenta divide os vídeos em diversos segmentos e traduz cada um desses para língua de sinais de acordo com o contexto. Os vídeos mostram pessoas fazendo conversações em língua de sinais, como pode ser visto na Figura 16.

O usuário tem a opção de alterar o tamanho da tela, entre outras configurações. Além disso, múltiplos usuários podem utilizar simultânea e assincronamente a ferramenta, desde que estejam traduzindo o mesmo vídeo. O programa tem uma base de dados dos sinais que correspondem às palavras das histórias. Não é possível a edição e a inserção de sinais que não constem nessa base. O *SignStream* não somente simplifica extremamente o processo da transcrição e aumenta a exatidão das transcrições, em virtude da ligação de eventos lingüísticos com frames vídeo, mas realça a habilidade do investigador de executar análises lingüísticas de vários tipos.

[1] http://www.bu.edu/asllrp/signstream

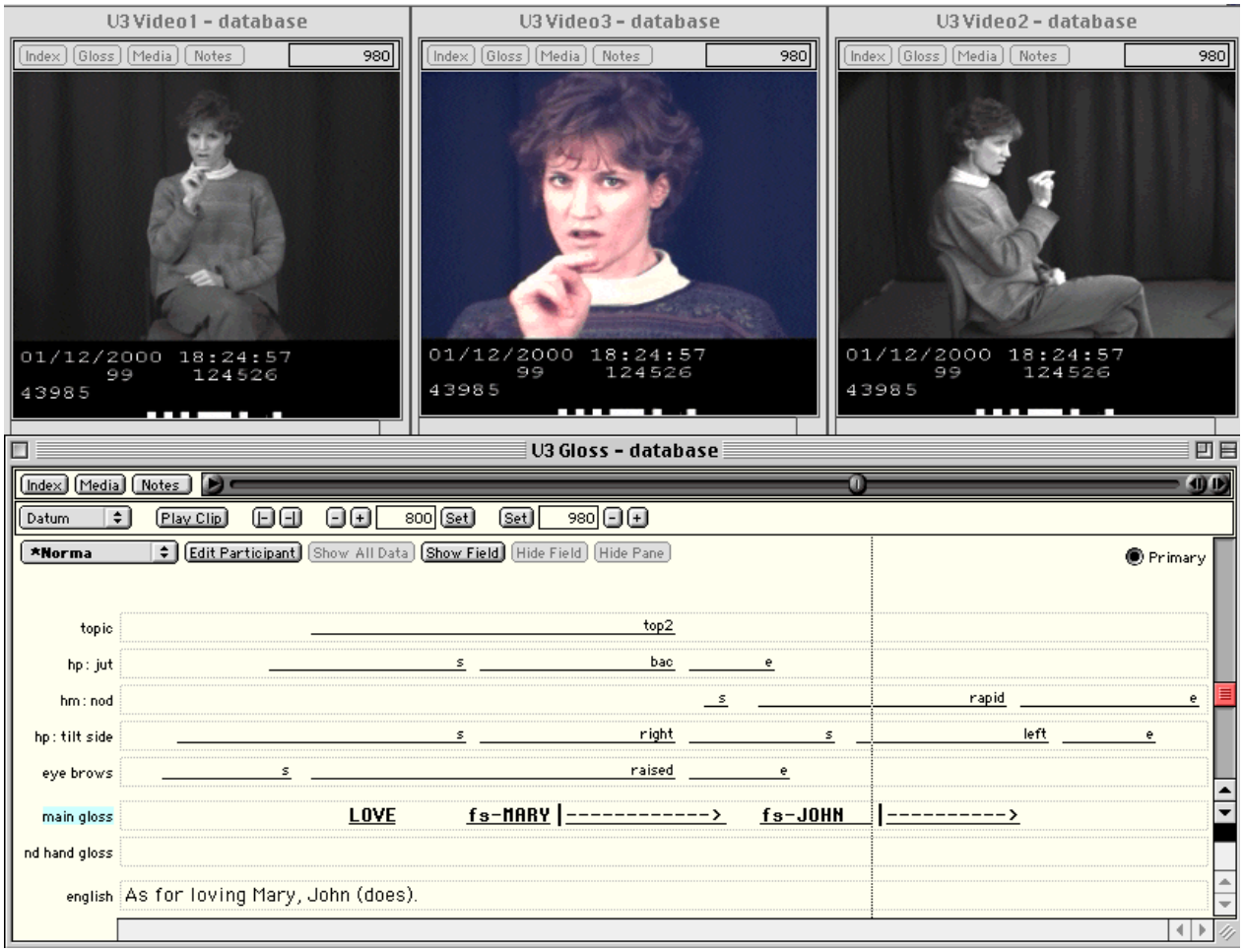

Figura 16: Sistema SignStream

O **Dicionário de LIBRAS – *on-line*[1]** tem como objetivo divulgar em larga escala a Língua Brasileira de Sinais, assim como o desenvolvimento de material didático lúdico e a sua utilização *on-line*, visando tornar-se potencializador de instrutores e agentes multiplicadores e capaz de atingir, através da Internet, todos os pontos do Brasil por mais distantes e inacessíveis que sejam.

Na primeira etapa do trabalho, foi dado ênfase ao desenvolvimento de material educativo para crianças na faixa escolar, tendo a LIBRAS sempre presente e com temas relacionados ao currículo do ensino fundamental, de acordo com as normas do ministério de educação. Este trabalho foi executado com a colaboração incondicional da equipe da Federação Nacional para Educação e Integração de Surdos – Feneis. Como visto na Figura 17,

o dicionário disponibiliza um pequeno vocabulário, que será ampliado constantemente, e está dividido em quinze categorias: alimentos, animais, repartições públicas e afins, cores, datas, diversos, lar, natureza, números, países, estados e cidades; pessoas e parentesco; profissões, roupas, transportes e verbos.

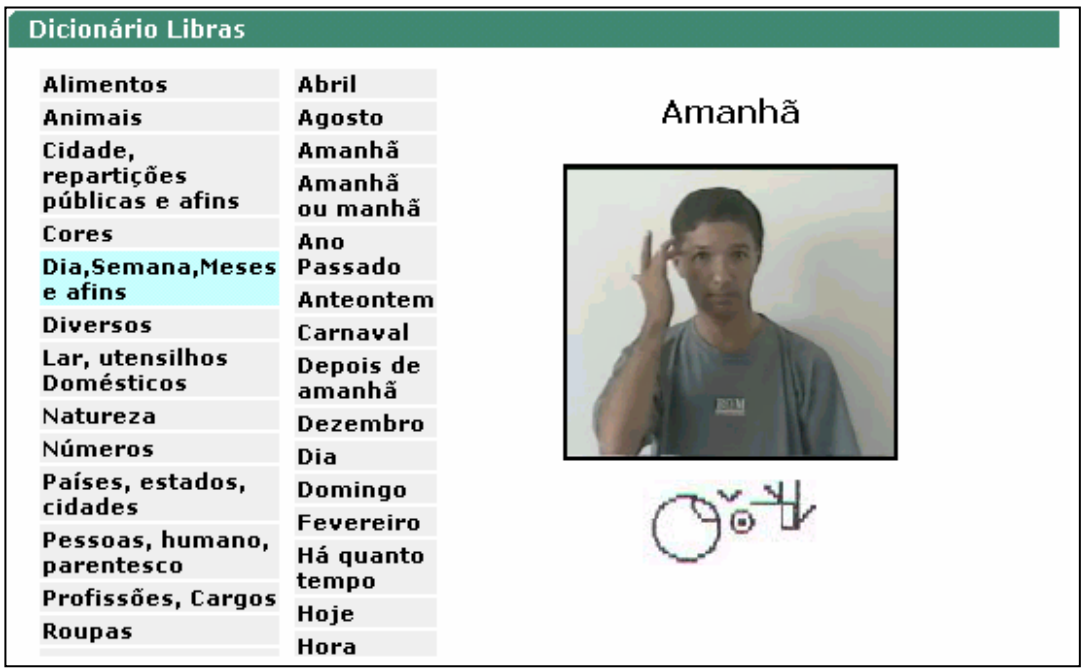

Figura 17: Dicionário de LIBRAS *On-Line*

[1] http://www.dicionariolibras.com.br

CAPÍTULO 3 - ESPECIFICAÇÃO H-ANIM

Este capítulo descreve os conceitos da especificação H-Anim, padrão utilizado no projeto para desenvolvimento dos avatares humanóides, incluindo como os avatares do H-Anim são estruturados e manipulados.

3.1 Avatares H-Anim

Em ISO/IEC FCD 19774:200x (2004) é especificada a estrutura e manipulação de avatares H-Anim. Avatares H-Anim são representação 3D articuladas que descrevem humanóides animados. Enquanto avatares H-Anim têm a intenção de representar figuras como humanas, eles são um conceito geral que não é limitado ao mesmo número de membros, cabeças e outras partes do corpo que são típicas do ser humano. Um avatar simples H-Anim é chamado de humanóide.

Avatares H-Anim são descritos utilizando os seguintes objetos H-Anim:

- *Humanoid*,
- *Joint*,
- *Segment*,
- *Site* e
- *Displacer*.

O objeto *Humanoid* é o *root* de um avatar H-Anim e providencia a ligação para todas as outras partes do humanóide.

O objeto *Joint* é associado ao objeto *Humanoid* ou outros objetos *Joint* utilizando um nó *transform* que especifica o estado atual da articulação junto com a geometria associada com a parte do corpo em anexo.

O objeto *Segment* especifica os atributos de ligação físicas entre os joints da figura humanóide.

O objeto *Site* especifica localizações, nas quais semânticas conhecidas podem ser associadas.

O objeto *Displacer* especifica informações sobre o alcance dos movimentos permitidos para os objetos nos quais ele está embutido.

3.1.1 Manipulação do avatar

Figuras H-Anim são animadas pela aplicação de transformações nas articulações. A habilidade de uma figura H-Anim rearranjar seus membros e corpo é ditado pelo número de articulações e segmentos utilizados para especificar cada membro e o corpo. Mais articulações e segmentos resultam em uma imagem mais flexível. Menos articulações resultam numa imagem menos flexível.

3.2 Objeto Humanoid

A geometria que especifica o corpo de um humanóide H-Anim pode ser descrito de duas maneiras:

- *skeletal*
- *skinned*

Cada um dos casos é descrito a seguir.

3.2.1 Especificação da geometria do corpo *skeletal*

O método *skeletal* especifica a geometria dentro do grafo de cena da hierarquia esqueletal, o qual é definido no campo *skeleton* do objeto *Humanoid*. A geometria definida dentro dos objetos *Segment* desta hierarquia descreve o corpo como partes geométricas separadas. Este método, enquanto computacionalmente eficiente, pode causar certas anomalias visuais (como costuras ou pregas) que diminui a aparência da figura humanóide.

3.2.2 Especificação da geometria do corpo *skinned*

O método *skinned* especifica o corpo como uma porção contínua de geometria, com o campo *skin* do objeto *Humanoid*. Para este método, conjuntos de dados de vetor e ponto são primeiramente definidos nos campos *skinCoord* e *skinNormal* do objeto *Humanoid*.

Os dados são definidos desta maneira para separá-los dos mecanismos internos do objeto *Humanoid* que utiliza esta informação. O objeto *Humanoid* utiliza os conjuntos de dados de vetor e coordenada para descrever a geometria que configurará a superfície da figura humanóide.

Esta superfície pode ser implementada como um simples *IndexedFaceSet*, como múltiplos *IndexedFaceSet*, ou como uma outra representação que providencie a mesma funcionalidade. Dependendo de como o *IndexedFaceSet* é renderizado dentro do ambiente gráfico e a configuração da figura humanóide, é possível que múltiplos *IndexedFaceSet* possam gerar uma melhor performance pelo isolamento de contínuas malhas para superfícies localizadas. Por esta razão a especificação não restringe a implementação da superfície em um simples método.

O objeto *Humanoid* também manipula os conjuntos de dados do vetor normal e das coordenadas definidos nos campos *skinCoord* e *skinNormal* para refletir as mudanças que

ocorrem com o grafo de cena esqueletal do campo *Skeleton*. Neste contexto da deformação da pele, cada objeto *Joint* da hierarquia esqueletal serve ao propósito da definição da coordenada com a qual os vértices das malhas contínuas são deformadas.

3.2.3 Objeto Joint

O objeto *Joint* é utilizado como um bloco de construção das articulações da figura humanóide. Cada articulação da figura humanóide é representada por um objeto *Joint*. Estes objetos *Joint* são organizados em uma hierarquia que descreve o relacionamento pai-filho, inerente aos objetos *Joint* do esqueleto e providencia um recipiente de informação que é específico de cada articulação do esqueleto.

O objeto *Joint* especifica o sistema de coordenada para si mesmo e os objetos definidos no campo *children.* Este sistema de coordenada é relativo ao sistema de coordenada do objeto pai, que na maioria dos casos é um outro objeto *Joint*, mas podendo também ser o objeto *Humanoid*.

Assim, o objeto *Joint* é um objeto de grupo especializado que pode somente ser um filho de um objeto *Joint* ou, no caso do *Joint HumanoidRoot*, o primeiro objeto da hierarquia esqueletal definida no campo *skeleton* do objeto *Humanoid*.

Um objeto *Joint* tem dois campos que permitem manipular vértices individuais definidos no campo *skinCoord* do objeto *Humanoid*. Colocando eventos do campo *rotation* do objeto *Joint,* são afetados os vértices indicados pelo campo *skinCoordIndex* por um fator que é descrito pelos valores correspondentes no campo *vertexWeight* do objeto *Joint*.

O campo *vertexWeight* contém uma lista de valores reais que descrevem uma quantidade de “pesos” a serem usados para afetar os vértices apropriados (como indicado pelo campo *skinCoordIndex*) do campo *skinCoord* do objeto humanóide. Os campos *vertexWeight*

e *skinCoordIndex* são utilizados somente quando um *Mesh* contínuo do modelo H-Anim está sendo definido.

O objeto Joint também é utilizado para armazenar outra informação específica da articulação. Em particular, um j*oint name* é providenciado de modo que aplicações possam determinar a identidade do objeto *Joint*. O objeto *Joint* pode também conter atributos para sistemas cinemáticos inversos[1] que controlem a figura H-Anim. Estes atributos incluem os limites superior e inferior da articulação, a orientação dos limites da articulação e um valor de dureza e resistência.

3.2.4 Objeto Segment

Cada parte do corpo (antebraço, coxa, etc.) da figura humanóide é representada por um objeto *Segment*. Estes objetos *Segment* estão organizados na hierarquia esquelética do objeto *Joint* do humanóide e providencia um recipiente para informação que é específica de cada segmento do corpo.

O objeto *Segment* é um objeto de grupo especializado que gera um recipiente para objetos no campo *children* e pode somente estar como um filho de um objeto *Joint* e deve estar emparelhado com um objeto *Joint* correto.

3.2.5 Objeto Site

O objeto *Site* pode ser utilizado para três propósitos. O primeiro é definir uma localização com “efeito fim” que pode ser utilizado por um sistema de cinemática inversa. O

[1] Cinemática inversa é o método que permite determinar os valores das coordenadas articulares, se conhecida a localização do extremo do membro.

segundo é para definir um ponto de anexação para acessórios como jóias e roupas. O terceiro é para definir uma localização para um câmera virtual em referência de um objeto *Segment* (como uma visualização "através dos olhos" do humanóide para utilização em mundos multi-usuários). Pretende-se utilizar os objetos *Site* como pontos de anexação, dos quais uma certa perspectiva de visualização pode ser vista (como os olhos esquerdo e direito), de modo que estes objetos estejam orientados de uma forma que fiquem na direção da câmera de quem está olhando.

Objetos Site são objetos de agrupamento que podem somente ser definidos com o campo *children* de um objeto *Segment*. Os campos *rotation* e *translation* do objeto *Site* definem a localização e a orientação do efeito fim com a coordenada *frame* do *Segment*.

O campo *children* do objeto *Site* é utilizado para armazenar alguns acessórios que podem ser anexados ao objeto *Segment*. O objeto *Site* especifica um sistema de coordenada para objetos no campo *children* que é relativo ao sistemas de coordenadas do objeto pai.

3.2.6 Objeto Displacer

A forma de objetos *Mesh* individual pode ser alterada de acordo com os requisitos da aplicação. No nível básico, isto é feito pela manipulação dos dados armazenados no campo *coord* dos objetos *Mesh*. No caso de avatares articulados, os objetos *Mesh* residem nos objetos *Segment*. No caso de avatares *Mesh* deformáveis, os objetos *Mesh* são especificados pelo campos *Mesh* do objeto *Humanoid*.

Pode ser necessário identificar grupos de vértices para com o *Mesh*, por exemplo, a aplicação pode precisar saber quais vértices dentro do crânio inclui a sobrancelha esquerda.

Pode também ser necessário providenciar "sugestões", como para a direção na qual cada vértice deveria mover-se. Tal informação é armazenada em um objeto *Displacer*. Para

avatares articulados, os objetos *Displacer,* para um particular objeto *Segment,* são armazenados em campos *displacers* dos objetos *Joint* no avatar. Esta informação, chamada *displacements*, é especificada no espaço local do objeto *Joint* particular e transformado em espaço *Humanoid* antes de serem aplicados ao *Mesh*.

Um objeto *Displacer* pode ser utilizado de três maneiras diferentes: no nível mais básico, pode simplesmente ser utilizado para identificar vértices correspondentes ao um recurso particular no *Mesh*; no próximo nível, pode ser utilizado para representar uma ação muscular particular do qual desloque os vértices em várias direções. A terceira maneira na qual um objeto *Displacer* poder ser utilizado, é para representar uma completa configuração dos vértices no *Mesh*.

Por exemplo, no caso do rosto, pode existir um objeto *Displacer* para cada expressão facial.

Cada objeto *Displacer* especifica uma localização, chamada *morph target*, que pode ser utilizada para modificar as propriedades de deslocamento da figura. A magnitude escalar de deslocamento dos objetos *Displacer* podem ser dinamicamente dirigidos por uma fonte externa como um interpolador.

Objetos *Displacer* são mais freqüentemente utilizados para controlar a forma do rosto porém, eles podem ser utilizados para outras partes do corpo.

Por exemplo, objetos *Displacer* podem ser utilizados para controlar a forma de mudança de um *Segment* braço como o bíceps, simulando o efeito da inflar do músculo.

3.3 Especificação de humanóides

O ISO/IEC FCD 19774:200x (2004) restringe a modelagem de figuras humanas do H-Anim para assegurar que animações projetadas para uma figura humana H-Anim estejam

compatíveis com outra figura humana do H-Anim. Estas restrições especificam o estado de um modelo antes de algumas animações serem aplicadas e também assegura consistência através de modelos de figuras humanas H-Anim.

O humanóide deve ser modelado em pé, disposto na direção +Z com a posição vertical em +Y e +X para a posição horizontal à esquerda. A origem (0, 0, 0) deve estar localizada ao nível do chão, entre os pés do humanóide.

Os pés devem estar posicionados no chão separadamente, espaçados com a mesma distância da largura dos quadris. O lado de baixo do pé deve estar em Y=0. Os braços devem estar retos e paralelos ao lado do corpo com as palmas das mãos de frente com as coxas. As mãos devem estar abertas, com os eixos das articulações “1” até “3” dos dedos sendo paralelo ao eixo X e os eixos dos dedo polegar com ângulo em $\pi/2$ radianos para a direção +Z. Assim, o sistema de coordenadas para cada articulação no dedo polegar é ainda orientado para alinhar com o do humanóide global.

Movimentos da articulação “0” dos dedos são tipicamente limitados na sua totalidade, e a rigidez destas articulações variam de dedo para dedo.

O rosto deve ser modelado com as sobrancelhas em repouso, a boca fechada e os olhos abertos.

O humanóide deve ser construído com a medida de um humano médio. Todas as dimensões são em metros e a modelagem típica de um humano é de 1.75 metros. A Figura 18 descreve a posição *default* de um humanóide.

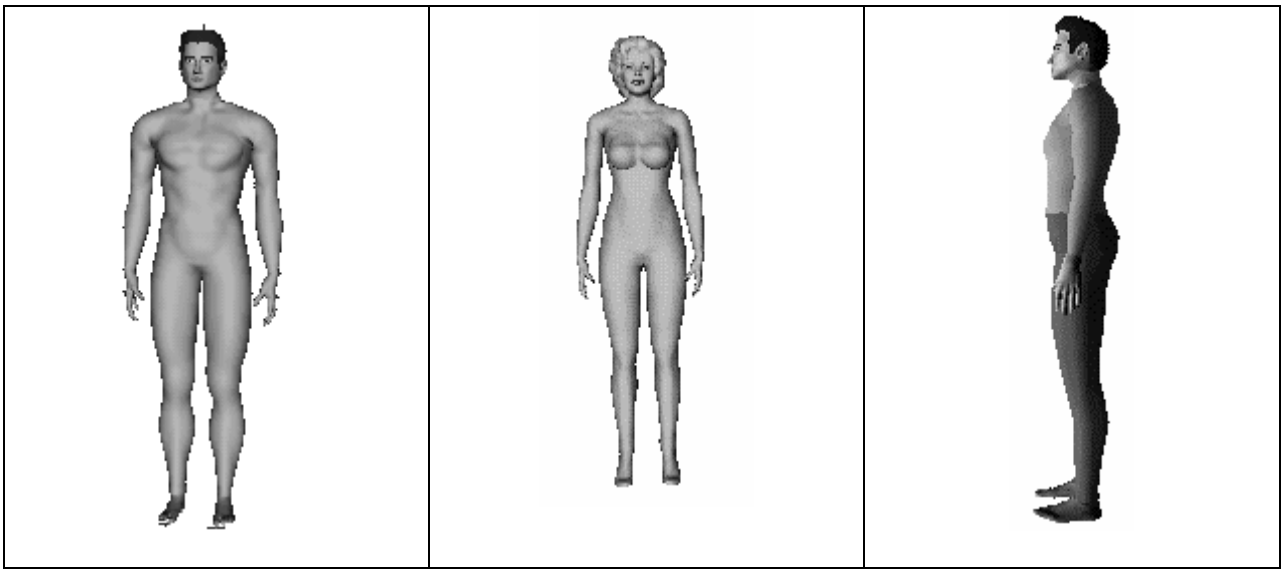

Figura 18: Posição padrão do humanóide

Nesta posição, todos os ângulos das articulações devem estar em zero. Isto é, todos os campos (*fields*) *rotation* em todos os objetos *Joint* devem ter um valor *default* de (0 0 1 0). Além disso, os campos *translation* devem ter o valor *default* de (0 0 0) e os fatores *scale* devem ter o valor *default* (1 1 1). O único campo que deve ter um valor não *default* é o *centre*, que é utilizado para especificar o ponto no qual a articulação (e os filhos e o seguimento *body* associados a ele se existirem) rotacionará.

Aplicando os valores *default* para a translação, rotação e escala de todos os *Joints* no corpo, este deverá ser retornar à posição neutra descrita anteriormente. Para facilitar isso, o sistema de coordenada para cada objeto *Joint* é orientado para alinhar com o do objeto Humanóide global.

O campo *centre* de cada objeto *Joint* deve estar localizado de modo que as articulações rotacionem da mesma forma que fariam num corpo humano real.

A Figura 19 mostra a orientação da mão. As cruzes sugerem possíveis localizações para o valores do campo *centre* do objeto *Joint* para articulações dos dedos e valores para o campo *centre* do objeto *Site* para a ponta do dedo.

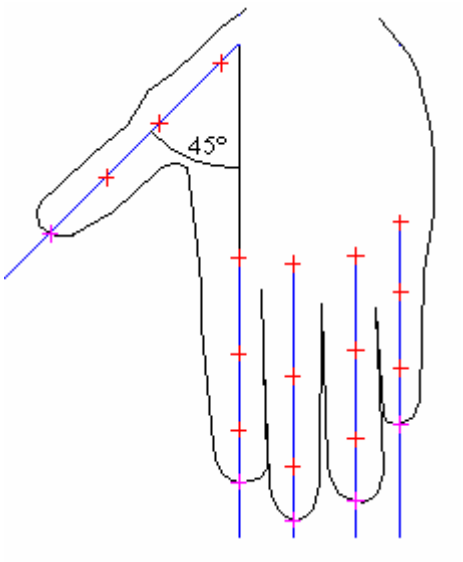

Figura 19: Orientação da mão

Todos os objetos *Segment* do corpo devem ser construídos no local, isto é uma sugestão e não um requisito. Isto é, eles não requerem que a translação, rotação ou escala estejam conectados com seus vizinhos.

Por exemplo, a mão é construída de tal modo esteja na posição correta relativa ao antebraço. O antebraço é construído de tal modo que esteja na posição correta relativa à parte de cima do braço e assim por diante.

Todas as coordenadas do corpo compartilham uma origem comum, que é a do próprio corpo. Se isto causa dificuldade de implementação para uma ferramenta de modelagem, é aceitável utilizar todo mecanismo de transformação geométrica disponível na linguagem de codificação para mover a geometria dentro de cada objeto *Segment* na posição correta.

3.4 Estrutura do humanóide

O corpo humano consiste em um número de segmentos (*Segment*), como o antebraço, a mão e o pé, que estão conectados aos outros segmentos através de articulações (*Joint*), como cotovelo, pulso e tornozelo. ISO/IEC FCD 19774:200x (2004) define abstrações para este segmentos e articulações que permitem que um corpo humano possa ser descrito de uma maneira estruturada e padronizada. Um corpo H-Anim é construído como uma série de objetos *Joint* aninhados, cada um dos quais pode ter um segmento associado a ele.

A Figura 20 mostra pares de objetos *Joint:Segment* que estão definidos pelo ISO/IEC FCD 19774:200x (2004). Se um objeto *Joint H-Anim* padrão é definido, e este *Joint* define um objeto *Segment*, este objeto *Segment* deve utilizar o nome apropriado desta hierarquia.

Por exemplo, se um objeto *Segment* l_upperarm é definido, deve ser definido como um filho do objeto *Joint* l_shoulder e, similarmente, se um objeto *Joint* r_knee tem um objeto *Segment* definido dentro do campo filho, este deve ser um objeto *Segment* r_calf.

3.4.1 O corpo

Os nomes dos objetos *Joint* para o corpo estão especificados na Tabela 2.

l_hip	l_knee	l_ankle	l_subtalar	l_midtarsal	l_metatarsal	
r_hip	r_knee	r_ankle	r_subtalar	r_midtarsal	r_metatarsal	
vl5	vl4	vl3	vl2	vl1		
vt12	vt11	vt10	vt9	vt8	vt7	
vt6	vt5	vt4	vt3	vt2	vt1	
vc7	vc6	vc5	vc4	vc3	vc2	vc1
l_sternoclavicular	l_acromioclavicular	l_shoulder	l_elbow	l_wrist		

r_sternoclavicular	r_acromioclavicular	r_shoulder	r_elbow	r_wrist		
HumanoidRoot	sacroiliac (pelvis)	skullbase				

Tabela 2: Nomes do objeto *Joint Body*

Os objetos *Joint* vl5 e sacroiliac são filhos do objeto *Joint* HumanoidRoot. O HumanoidRoot é armazenado no campo humanoidBody do objeto *Humanoid*, mas todos os outros objetos *Joint* são descendentes de vl5 ou sacroiliac. Se estes objetos *Joint* estão faltando, objetos *Joint* do nível mais baixo podem ser filhos de HumanoidRoot.

3.4.2 As mãos

Os nomes do objetos Joint para as mãos estão especificados na Tabela 3.

l_pinky0	l_pinky1	l_pinky2	l_pinky3	l_ring0	l_ring1	l_ring2	l_ring3
l_middle0	l_middle1	l_middle2	l_middle3	l_index0	l_index1	l_index2	l_index3
l_thumb1	l_thumb2	l_thumb3					
r_pinky0	r_pinky1	r_pinky2	r_pinky3	r_ring0	R_ring1	r_ring2	r_ring3
r_middle0	r_middle1	r_middle2	r_middle3	r_index0	R_index1	r_index2	r_index3
r_thumb1	r_thumb2	r_thumb3					

Tabela 3: Nomes do objeto *Joint Hand*

3.4.3 O rosto

Muitas implementações de humanóide têm feito uso de estrutura faciais articuladas para simular expressões faciais.

O sufixo "_joint" é utilizado aqui por causa de seus recursos serem controlados pelos grupos de músculo ao invés de articulações atuais, sendo a exceção o objeto *Joint* temporomandibular. O sufixo "_joint" providencia uma distinção entre o nome do objeto *Joint* e o nome do objeto *Segment* correspondente.

Todos os objetos faciais *Joint* são filhos do objeto *Joint skullbase*. O centro de rotação do olho e a pálpebra são o centro geométrico da bola do globo ocular. A rotação da pálpebra é de zero radianos e uma rotação positiva de π radianos fecha a pálpebra até ela estar completa.

As sobrancelhas estão em zero radiano na rotação por *default* e pode ser rotacionada ao redor do meio da sobrancelha. A boca está fechada quando o objeto *Joint* temporomandibular está em zero radiano.

O nome dos objetos *Joint* para o rosto estão especificados na Tabela 4.

l_eyeball_joint	r_eyeball_joint
l_eyebrow_joint	r_eyebrow_joint
l_eyelid_joint	r_eyelid_joint
Temporomandibular	

Tabela 4: Nomes do objetos *Joint Face*

No capítulo Objetos adicionais *Joint* e *Segment*, são mostrados detalhes de como adicionar articulações faciais.

Os objetos *Joint Face* do conjunto básico de objetos *Joint* somente oferecem uma forma de animação facial.

3.4.4 Hierarquia

A hierarquia completa formada pelo conjunto básico de objetos *Joint* está especificada na Figura 20 com os nomes dos segmentos ao lado das articulações às quais estão anexados.

```
HumanoidRoot : sacrum
  sacroiliac : pelvis
  |   l_hip : l_thigh
  |     l_knee : l_calf
  |       l_ankle : l_hindfoot
  |         l_subtalar : l_midproximal
  |           l_midtarsal : l_middistal
  |             l_metatarsal : l_forefoot
  |   r_hip : r_thigh
  |     r_knee : r_calf
  |       r_ankle : r_hindfoot
  |         r_subtalar :  r_midproximal
  |           r_midtarsal : r_middistal
  |             r_metatarsal : r_forefoot
  vl5  : l5
    vl4 : l4
      vl3  : l3
        vl2 : l2
          vl1  : l1
            vt12 : t12
              vt11 : t11
                vt10  : t10
                  vt9 : t9
                    vt8 : t8
                      vt7 : t7
                        vt6  : t6
                          vt5 : t5
                            vt4 : t4
                              vt3 : t3
                                vt2 : t2
                                  vt1  : t1
                                    vc7 : c7
                                    | vc6 : c6
                                    |   vc5 : c5
                                    |     vc4  : c4
                                    |       vc3 : c3
                                    |         vc2  : c2
                                    |           vc1 : c1
                                    |             skullbase : skull
                                    |               l_eyelid_joint : l_eyelid
                                    |               r_eyelid_joint : r_eyelid
                                    |               l_eyeball_joint : l_eyeball
                                    |               r_eyeball_joint : r_eyeball
                                    |               l_eyebrow_joint : l_eyebrow
                                    |               r_eyebrow_joint : r_eyebrow
                                    |               temporomandibular : jaw
                                    l_sternoclavicular : l_clavicle
                                    | l_acromioclavicular : l_scapula
                                    |   l_shoulder : l_upperarm
                                    |     l_elbow : l_forearm
                                    |       l_wrist : l_hand
                                    |         l_thumb1 : l_thumb_metacarpal
                                    |           l_thumb2 : l_thumb_proximal
                                    |             l_thumb3 : l_thumb_distal
                                    |         l_index0 : l_index_metacarpal
                                    |           l_index1 : l_index_proximal
                                    |             l_index2 : l_index_middle
                                    |               l_index3 : l_index_distal
```

```
|          l_middle0 : l_middle_metacarpal
|            l_middle1 : l_middle_proximal
|              l_middle2 : l_middle_middle
|                l_middle3 : l_middle_distal
|          l_ring0 : l_ring_metacarpal
|            l_ring1 : l_ring_proximal
|              l_ring2 l_ring_middle
|                l_ring3 : l_ring_distal
|          l_pinky0 : l_pinky_metacarpal
|            l_pinky1 : l_pinky_proximal
|              l_pinky2 : l_pinky_middle
|                l_pinky3 : l_pinky_distal
r_sternoclavicular : r_clavicle
  r_acromioclavicular : r_scapula
    r_shoulder : r_upperarm
      r_elbow : r_forearm
        r_wrist : r_hand
          r_thumb1 : r_thumb_metacarpal
            r_thumb2 : r_thumb_proximal
              r_thumb3 : r_thumb_distal
          r_index0 : r_index_metacarpal
            r_index1 : r_index_proximal
              r_index2 : r_index_middle
                r_index3 : r_index_distal
          r_middle0 : r_middle_metacarpal
            r_middle1 : r_middle_proximal
              r_middle2 : r_middle_middle
                r_middle3 : r_middle_distal
          r_ring0 : r_ring_metacarpal
            r_ring1 : r_ring_proximal
              r_ring2 : r_ring_middle
                r_ring3 : r_ring_distal
          r_pinky0 : r_pinky_metacarpal
            r_pinky1 : r_pinky_proximal
              r_pinky2 : r_pinky_middle
                r_pinky3 : r_pinky_distal
```

Figura 20: Conjunto básico da hierarquia de *Joints*

3.4.5 Objetos adicionais *Joint* e *Segment*

Objetos adicionais *Joint* e *Segment* do corpo podem ser definidos; para tanto existem somente 3 requisitos:

1. Os nós *Joint* listados na hierarquia, se presentes, devem utilizar nomes específicos.

2. Novos nós *Joint* não são permitidos dentro da cadeia do padrão da hierarquia *Joint*. Estes nós *Joint* não-padrão podem ser filhos de nós *Joint* padrão ou de outros nós *Joint* não padrões.

Exemplo: Um cotovelo adicional não pode ser adicionado a um braço. Entretanto, novos apetrechos (como cabelo, trança, etc.) podem ser adicionados a um humanóide pela criação de novos nós *Joint* que existam como filhos de outros nós *Joint*.

3. Nós *Joint* adicionais devem ser adicionados de forma a não interferir no movimento de nós *Joint* padrões, até no caso de não haver animação disponível para estes nós.

Animações para nós *Joint* do conjunto básico não devem ser dependentes em animações de algum nó *Joint* adicional (ou de seus filhos) que podem ser pai deles. Sistemas de cinemática inversa podem considerar nós *Joint* adicionais quando da execução de cálculos, mas não são requeridos para tanto.

Nós *Joint* adicionais devem ter o prefixo "x_" (e.g. hanim_x_cabelo) para distinguí-los do conjunto básico de objetos *Joint* que podem ter nomes similares.

CAPÍTULO 4 – ASPECTOS DE MODELAGEM DE AMBIENTES VIRTUAIS

A RV é vista como uma plataforma promissora para o desenvolvimento de novas aplicações em muitas áreas como a medicina, entretenimento, ciências e negócios. Apesar destas vantagens potenciais, não vemos ainda o desenvolvimento difundido e o uso de aplicações de RV na prática.

De acordo com Astheimer (1999), a falta da proliferação de aplicações de RV pode ser atribuída particularmente aos desafios na construção de aplicações de RV. Em particular, interfaces de aplicações de RV são muito complexas em comparação às interfaces de aplicações convencionais (BOWMAN, 1999). Interfaces RV exibem características visuais, comportamentais e de interação distintas das aplicações convencionais.

4.1 Características visuais

Enquanto aplicações convencionais utilizam principalmente interfaces 2D, interfaces de RV utilizam saídas 2D e 3D. O maior objetivo de ambientes virtuais é prover aos usuários um ambiente realista, ou seja, provocar a sensação de “estar lá”. Interfaces de RV utilizam, em sua grande maioria, gráficos 3D, e o projeto e a implementação de interfaces 3D são normalmente mais difíceis que interfaces 2D.

Interfaces convencionais contém tipicamente somente objetos virtuais, em contrapartida, interfaces de RV podem conter objetos virtuais e físicos que coexistem e trocam informações entre si, como no caso de sistemas de RV aumentada. A necessidade de

alinhar corretamente esses objetos virtuais e físicos é um desafio no projeto de interfaces de RV (AZUMA, 1997).

4.2 Características comportamentais

Nas interfaces convencionais, os objetos normalmente exibem comportamentos estáticos; em geral, têm comportamentos predeterminados que são ativados em respostas às ações, também chamados eventos dos usuários. Interfaces de RV contêm tanto objetos estáticos como objetos dinâmicos que exibem comportamentos autônomos. Ao contrário de objetos estáticos, objetos autônomos conseguem mudar seu próprio estado. Eles podem se comunicar entre si, conseguindo afetar os padrões de comunicação e comportamento (TANRIVERDI e JACOB, 2001).

4.3 Características de interação

Enquanto interfaces convencionais suportam principalmente interações explícitas, interfaces de RV normalmente suportam tanto interações implícitas como explícitas. Interações explícitas permitem um meio mais natural e fácil de interação homem-computador pelo uso da mão, braço, cabeça ou interações baseadas nos movimentos dos olhos. Entretanto, este estilo de interação é muito mais complexo no seu desenvolvimento, quando comparado ao estilo de interação explícito (TANRIVERDI e JACOB, 2001).

A Tabela 5 mostra as diferenças entre as características de interfaces convencionais e de RV, como visto na tabela; aplicações de RV contêm uma maior variedade de tipos de objetos, de comportamentos e padrões de comunicação.

Na ausência de um modelo conceitual e uma metodologia que providenciem uma direção durante o projeto de aplicações de RV, desenvolvedores enfrentam desafios importantes como compreender as características de uma interface de um sistema de RV, decompor as tarefas de projeto em tarefas menores e de melhor compreensão e manter uma comunicação entre a equipe de projeto de aplicações de RV.

Características	**Interfaces Convencionais**	**Interfaces de RV**
Objetos gráficos	2D	2D e 3D
Tipos de objetos	Objetos virtuais	Objetos virtuais e físicos
Comportamentos do objetos	Objetos passivos	Objetos estáticos e dinâmicos
Padrões de comunicação	Simples	Complexo
Interações homem-computador	Explícitos	Explícitos e implícitos

Tabela 5: Características de interfaces convencionais e interfaces de RV

4.4 Modelos e metodologias de projeto de aplicações

O modelo de projeto de interface *Four Level,* desenvolvido por Foley, et al. (1996), descreve a interface do usuário como um meio de diálogo entre o usuário e o computador. Os quatro níveis do modelo são organizados baseando-se no significado e na forma de diálogo entre o usuário e o computador. Os níveis se focam principalmente nas especificações de interações do usuário utilizando comandos explícitos. Este modelo é bastante apropriado aos comandos de linguagens e interfaces do tipo GUI (*Graphics User Interface*). Mas em interfaces de RV, que necessitam de interações implícitas, objetos dinâmicos e físicos, este modelo não consegue abranger, como também não existem especificações de comunicação entre objetos.

Outro relevante modelo de projeto de interface é o *Command Language Grammar* (CLG) desenvolvido por Moran. Este modelo possibilita um método de descrever e modelar interfaces de linguagem de comando. Como no modelo de FOLEY, et al. (1996), o CLG divide o projeto da interface em níveis, porém as especificações dos níveis são mais formais e detalhadas no CLG. Embora este modelo trabalhe bem em interfaces de linguagem de comando, a aplicabilidade em interface de RV é limitada. Objetos dinâmicos, interações utilizando comandos implícitos, objetos físicos e padrões de comunicação entre objetos estão fora do escopo deste modelo.

Um terceiro modelo de projeto de interface é o modelo *Object Action Interface* (OAI) (SHNEIDERMAN, 1998). Foi desenvolvido particularmente para o projeto de interfaces GUI. Para ir de encontro às necessidades de interfaces GUI, este modelo enfatiza a importância de representações de objetos e suas ações. O modelo OAI foca-se nas interações explícitas utilizando manipulações diretas, e mantém uma quantidade de pequenas sintaxes nas especificações de interação. Entretanto, o modelo OAI não especifica as características distintivas das interfaces RV, como objetos dinâmicos, interações implícitas, objetos físicos e padrões de comunicação entre objetos.

Outra possibilidade para desenvolvedores é utilizar metodologias e modelos de projeto de propósito geral como o modelo orientado a objetos e o *Object Modeling Technique* (OMT), que são propostas para desenvolvimento de aplicações (RUMBAUGH, 1991). Porém, por serem modelos de propósito geral, estas metodologias não atendem aos requisitos de interfaces de RV vistos anteriormente.

Apesar da falta de modelos e metodologias que compreenda as características de interfaces RV, existem vários projetos relevantes na área de RV que podem ser utilizados na construção de aplicações RV. A complexidade de interfaces RV tem gerado esforços para o

desenvolvimento de linguagens de projeto, *frameworks* e outras ferramentas que dêem suporte ao desenvolvimento de interfaces RV.

Muitos pesquisadores têm desenvolvido aplicações ou *frameworks* para uso por não programadores ou programadores inexperientes. Alice (PAUSCH, 2003) é descrito como um sistema rápido de protótipos para RV; faz uso de uma interface GUI que permite manipulação interativa da cena do AV corrente por usuários sem habilidades técnicas. O desenvolvedor é capaz de ver as mudanças feitas imediatamente. Entretanto, sua funcionalidade é muito limitada, a menos que o usuário seja capaz de programar na linguagem de *script Python*.

O *Virtual Reality Interface Design Model* (VRID) desenvolvido por Tanriverdi e Jacob (2001) providencia um modelo e uma metodologia para criação de AV, bem como um *framework* para o seu desenvolvimento. Entretanto, não possibilita a habilidade para usuários implementar sua especificação do projeto.

4.5 Metodologia de projeto de interfaces

Existem diversas fontes de informação no projeto de interfaces 2D e 3D. Princípios gerais de interface incluem a visibilidade e o *feedback*. Interfaces 2D têm sido pesquisadas, levando a muitas ferramentas eficientes de projeto de interface 2D. Estratégias para o projeto efetivo de uma ferramenta de interface 3D podem ser aprendido destas ferramentas 2D, como a alta funcionalidade e uma baixa curva de aprendizado da ferramenta.

Shneiderman (1998) cita 5 fatores humanos mensuráveis que devem ser levados em consideração no projeto de interfaces, estes fatores estão descritos na Tabela 6.

Tempo de aprendizado	Quantidade de tempo que os usuários típicos levam para aprender utilizar os comandos relevantes para executar um conjunto de tarefas?
Velocidade de performance	Quantidade de tempo que ele leva para

	devolver o resultado das tarefas?
Taxa de erros pelos usuários	Quantos e quais tipos de erros as pessoas geram na realização das tarefas?
Retenção do aprendizado	Quanto do que aprenderam os usuários mantêm depois de uma hora, um dia ou uma semana?
Satisfação subjetiva	Qual o grau de satisfação do usuários na utilização de diversos aspectos do sistema?

Tabela 6: Fatores relevantes em projetos de interface

4.6 Engenharia de *software* e projeto

O trabalho de engenheiros de *software* no projeto de aplicações é também valioso. O ciclo de vida do desenvolvimento de *software* providencia uma estrutura a seguir no desenvolvimento de uma ferramenta. Linguagens de especificação podem ser utilizadas para estruturar o processo de projeto, por exemplo o UML, o qual facilita na identificação dos casos de uso e papéis no projeto de um sistema.

Um projeto de um AV gera inúmeras questões a serem resolvidas, uma delas é a noção estética de projetar algo para criar respostas perceptivas diretas e em segundo lugar, existe a noção de engenharia no projeto como a criação de planos, modelos e a documentação utilizados no desenvolvimento de testes desejados.

No projeto do AV, podemos observar uma tensão entre a estética e a engenharia do projeto em termos de percepção e estrutura. O processo de desenvolvimento de AVs não é bem documentado ou profundamente pesquisado. Ao contrário de animações 3D, há, é claro, a inclusão de requerimentos de performance aceitável. Práticas de engenharia de *software* podem realmente ajudar com a modelagem estrutural, mas não podem ajudar com a modelagem perceptiva.

4.7 Etapas do projeto de ambientes virtuais

Kulwinder (1998) construiu um esboço de uma das etapas de um projeto de um AV:

1. Especificação de Requisitos;
2. Levantamento de Material de Referência de Objetos do Mundo Real;
3. Estruturação do Modelo Gráfico;
4. Construção e Posicionamento dos Objetos no AV;
5. Melhorando o Ambiente com Textura, Luzes, Som e Interação;
6. Otimizando o Ambiente

A Figura 21 representa um modelo de processo para metodologia de projeto.

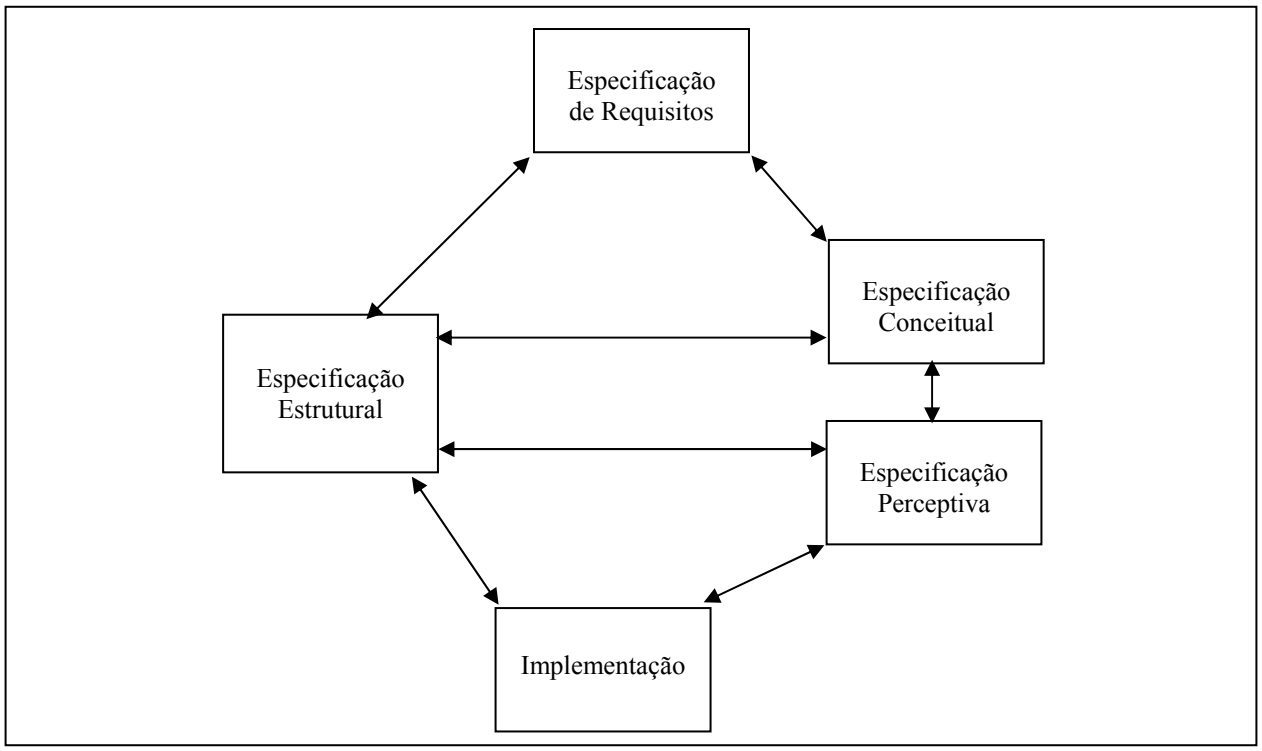

Figura 21: Modelo de processo de projeto

4.7.1 Especificação de Requisitos

Referente à etapa de Especificação de Requisitos de Kulwinder (1998) e muito próximo do conceito paralelo ao conceito de Análise de Requisitos ou Levantamento de Requisitos da Engenharia de *Software*. Um dos requisitos básicos é que a proposta deveria ser claramente estabelecida aqui (FENCOTT, 2003).

4.7.2 Especificação Conceitual

Referente à etapa de Levantamento de Material de Referência de Objetos do Mundo Real, é efetivamente a atividade de pesquisa em muitos projetos de *design*, mas em particular naqueles com componentes estéticos. Envolve, definição de materiais, tiragem de fotos, esboços, gravação de vídeo e som, etc. Pode também incluir a construção de *storyboards*, etc. Esta é a etapa em que os desenvolvedores ou a equipe de desenvolvimento conhece o mundo que eles terão que construir (FENCOTT, 2003).

4.7.3 Especificação Perceptiva

Nesta etapa é construído um modelo da natureza das oportunidades de percepção e seus inter-relacionamentos. Equivale à etapa 5 da metodologia de Kulwinder (1998). Traz convicção no AV, realismo perceptivo, sem consideração de algumas contrapartidas do mundo real. Oportunidades perceptivas são especificamente definidas na noção de presença, mas podem ser estendidas de acordo com a co-presença (FENCOTT, 2003).

4.7.4 Especificação Estrutural

Refere-se à etapa 3 da metodologia de Kulwinder (1998), cobre uma variedade de atividades, que relata a realização estrutural sobre a qual a plataforma do AV está colocada, objetiva a construção de um incentivo sensorial em tempo de execução. Modelagem estrutural pode ser vista como o projeto a partir da modelagem conceitual e como pré-requisito para a modelagem perceptiva. A conclusão da etapa da modelagem estrutural resultará em diagramas de grafo de cena que demonstra o *lay-out* da estrutura de código do AV e é programado em componentes comportamentais. Em termos de prática de engenharia de *software*, o UML poderia ser incorporado neste modelo. Por exemplo, utilizar diagramas de caso de uso (*Use Case*) servirá para identificar os relacionamentos (associações) entre usuários e o AV, outra prática de utilidade seria modelar as estruturas dos nós do grafo de cena como modelos de objetos em diagramas de classes para componentes programáveis (FENCOTT, 2003).

4.7.5 Implementação

Esta etapa relata a fase da codificação da engenharia de *software* que deve ocorrer depois de todos os requisitos serem levantados, e as atividades de análise, projeto e *design* terem sido realizadas. A implementação se refere a utilização de uma ferramenta de desenvolvimento que permite a codificação do grafo de cena diretamente numa linguagem de programação como, por exemplo, o VRML e Java3D, ou a utilização de um ambiente de desenvolvimento integrado (IDE) como o World Tool Kit ou X3D Edit.

CAPÍTULO 5 – METODOLOGIA PROPOSTA PARA O PROJETO DE AMBIENTES VIRTUAIS

A principal dificuldade do desenvolvimento de aplicações de RV está no gerenciamento da complexidade. A complexidade surge porque o desenvolvedor da aplicação deve realizar o projeto de três aspectos interrelacionados, isto é, forma, função e comportamento ao mesmo tempo.

A forma se refere à aparência dos objetos virtuais, sua estrutura (para objetos compostos) e a estrutura da cena do mundo virtual. Outras propriedades físicas (que podem ser requisitos para simulação física) como massa, propriedade do material, velocidade, aceleração podem ser parte da informação da forma.

A função, basicamente, se refere à codificação que os objetos virtuais devem realizar em seus comportamentos, se autônomos, ou em resposta a algum evento ou estímulo externo, enquanto comportamento refere-se a como os objetivos virtuais individuais mudam dinamicamente e executam diferentes funções sobre um determinado período de tempo, normalmente expressados por estado, troca de dados/eventos e restrições entre objetos (KIM e KO, 1998).

A maioria das aplicações de RV é desenvolvida na seqüência do projeto da forma, seguido da programação das funções e comportamentos. Isto é, objetos virtuais são modelados utilizando sistemas de projetos assistidos por computador (CAD - *Computer Aided Design*), então convertidos em formatos de arquivos apropriados ou estrutura de dados para serem utilizados por ambientes virtuais em tempo de execução (KIM e KO, 1998). O programa de aplicação é normalmente escrito pelo carregamento inicial demonstrado no modelo de código da Figura 22. Por razões de performance (i.e. para gerenciar diferenças na taxa de simulação e renderização), o *loop* de renderização pode ser dividido em múltiplos

processos cliente/servidor. Depois deste ponto, nenhuma metodologia particular ou diretrizes existem para o desenvolvimento das próximas etapas do sistema.

O resultado é a codificação não estruturada em relação às informações de função, comportamento e, restrições entre eles estão todas misturadas, gerando uma leitura (*readability*) difícil para futuras manutenções de *software*.

```
Main() {
  ...
  Inicializa o ambiente de execução
  Configura as condições do ambiente
  ...
  Importa ou cria objetos
  Cria a estrutura da cena
  ...
  /* laço de renderização infinito */
  While (1) {
    Leia rastreadores e outros dispositivos de entrada
    Cheque interferências
    Renderiza gráficos
  }
}
```

Figura 22: Um processo típico de um sistema de RV (PAUSCH, 1993)

A construção de mundos virtuais freqüentemente requer muitas revisões, e a mudança de um aspecto do mundo afetará indubitavelmente outros aspectos. Por exemplo, diferentes formas e configurações (posições e orientações no espaço) podem resultar em diferentes comportamentos dinâmicos. A forma também pode afetar a funcionalidade.

Como o ciclo de desenvolvimento é difícil para administrar o trabalho a um nível simples de abstração, um processo estruturado e preliminar de “engenharia de requisitos” pode ser de ajuda na gestão dos requisitos de usuário e sistema, tornando representações

explícitas destes requisitos e certificando-se que eles serão considerados durante o ciclo do desenvolvimento do sistema de RV.

O desenvolvimento de sistemas de RV deve incluir especificações de modelos de objetos virtuais e seus controles de tempo real dentro do ambiente.

Normalmente, a comunidade de pesquisa de engenharia de *software* tem se focado na especificação de sistemas de controle embutido em termos de suas funcionalidades e comportamentos, enquanto pesquisadores da computação gráfica têm-se concentrado na visualização das formas dos dados. Harel, et al. (1990) desenvolveu uma ferramenta CASE[1] gráfica chamada STATEMATE, para especificação, análise, projeto e documentação de sistemas complexos. Esta ferramenta tem recursos como um simulador, um conjunto de procedimentos de teste e três diferentes métodos de especificação para suporte à múltiplas visões. ASADAL de Kang e Ko (1995), é também uma ferramenta CASE desenvolvida para sistemas de tempo real com um ambiente para simulação.

Fishwick (1996) propõe uma convergência de sistemas e engenharia de *software* para a construção de modelos de simulação física pelo uso de modelos e métodos orientados a objetos, e a separação dos modelos estáticos e dinâmicos do sistema.

Dentro deste conceito, a ferramenta ASADAL/PROTO incorpora o projeto da forma em adição ao comportamento e função. Este método permite a especificação das funções e comportamentos através de DFD (Diagrama de Fluxo de Dados) e a representação da forma é feita pelo *Visual Object Specification* (VOS) (KIM e KO, 1998), permitindo a descrição das propriedades físicas dos objetos (i.e. forma, parâmetros de dimensão, massa, material), configuração (i.e. posição, velocidade), restrições espaciais entre os objetos e seus comportamentos primitivos encapsulados.

[1] CASE – *Computer Aided Software Engineering*, significa Engenharia de Software apoiada por Computador

Uma das dificuldades dos processos de desenvolvimento de AV é a reutilização de modelos de projetos 3D entre diferentes AVs, devido à falta de padrões de métodos e linguagens de especificação e notação. Linguagens como VRML, MPEG4, Java 3D, etc., são utilizadas somente na fase de implementação, mas não durante o projeto. Além disso, a manutenção do AV é também dificultado por não haver um projeto bem estruturado.

5.1 O Processo de projeto para ambientes virtuais

Os métodos que melhor preenchem o desenvolvimento de AV são os do modelo orientado a objetos. Casey (1994) disse que o uso de técnicas de orientação a objetos deveria ser a chave para um avanço real no desenvolvimento de AV baseados na Engenharia de *Software*. Porém, somente as técnicas tradicionais de projeto orientado a objetos não são suficientes para projetos de AV.

O projeto de AV inclui três processos principais: Projeto 3D, Projeto da Arquitetura Interna de Componentes e Projeto de Sistema. Na Tabela 7, os projetos são decompostos em tarefas.

Processo	**Tarefa**
Projeto 3D (P3D)	Modelagem do AV (P3D-MAV)
	Modelagem de Avatares (P3D-MA)
Projeto da Arquitetura Interna de Componentes (AIC)	Modelagem da Percepção (AIC-MP)
	Modelagem e Seleção de Recursos Internos dos Componentes (AIC-SRI)
	Modelagem das Ações Físicas (AIC-MAF)
	Modelagem de Reações (AIC-MR)
Projeto do Sistema (PS)	Projeto de Interface (PS-PI)
	Modelo Estática Expandido (PS-MEE)
	Modelo Dinâmico Expandido (PS-MDE)

	Projeto da Arquitetura do Sistema (PS-PAS)
	Descrições Detalhadas dos Métodos (PS-DDM)
	Projeto de Dados Persistentes (PS-PDS)

Tabela 7: Processos e tarefas de projeto de AV (SÁNCHEZ-SEGURA, 2001)

5.2 Processo: Projeto 3D

O desenvolvedor, nesse processo, deve ter um bom conhecimento das características físicas do mundo a ser modelado, mas deve ter também um conhecimento básico de projeto gráfico. O resultado desta tarefa será um conjunto de documentos a ser utilizado pelo projetista gráfico durante o Processo de Implementação da construção de modelos tridimensionais com as ferramentas selecionadas.

5.2.1 Modelagem do AV

Esta tarefa inclui a definição de um conjunto de espaços virtuais e de objetos a serem incluídos dentro deles.

5.2.2 Modelagem de Avatares

Esta tarefa inclui a definição de habitantes virtuais, suas aparências e estruturas físicas.

A Figura 23 mostra as dependências entre as tarefas de Projeto 3D e quais resultados são gerados a partir do Projeto 3D.

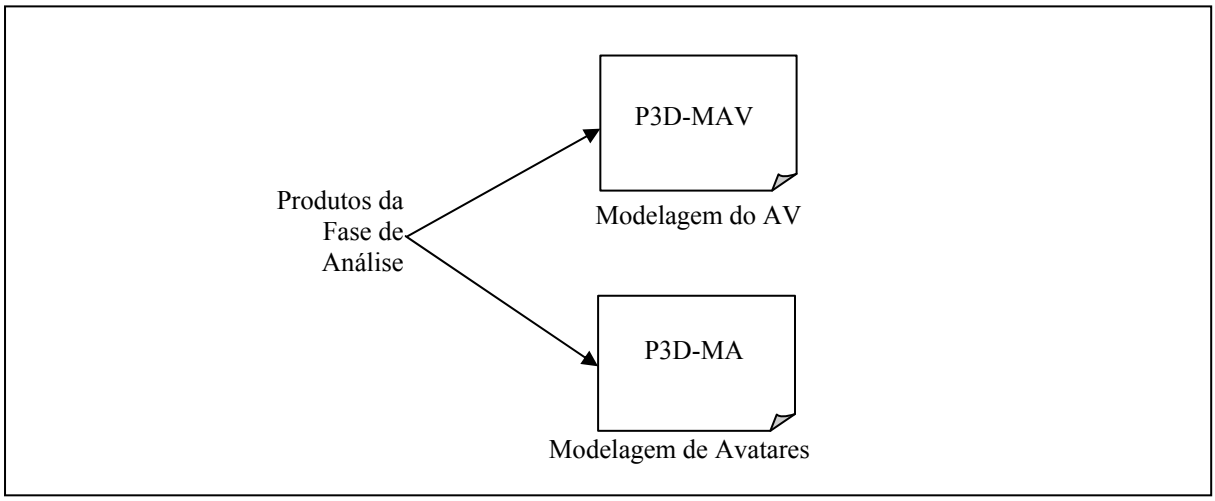

Figura 23: Tarefas do processo de projeto 3D (SÁNCHEZ-SEGURA, 2001)

5.3 Processo: Projeto da Arquitetura Interna de Componentes

Este processo pode envolver diferentes profissionais de diferentes áreas; isto se deve ao fato de ser um trabalho multi-disciplinar, necessitando da dotação, nestes tipos de sistemas, de interesses suficientes de recursos de interação.

Deve-se enfatizar, ainda, a importância de relacionamentos entre este processo e o processo de Projeto 3D; pois cada um deve ser coerente com o outro.

5.3.1 Modelagem de Percepção

Nesta tarefa é definida a maneira com a qual avatares e agentes podem perceber a presença de outros habitantes e objetos no AV.

5.3.2 Modelagem de Ações Físicas

As atividades que os avatares devem ser capazes de executar no ambiente devem ser projetadas levando em consideração a estrutura do avatar definida na tarefa de modelagem de avatares.

5.3.3 Modelagem e Seleção de Recursos Internos dos Componentes

Muitas vezes, a credibilidade dos avatares depende mais da sua habilidade de mostrar um comportamento racional e expressão de emoções do que de seu realismo físico, então é necessário dotar os avatares com um modelo de personalidade e emoção e associar as ações definidas acima com as emoções apropriadas.

5.3.4 Modelagem de Reações

Nesta tarefa é definida a maneira com a qual elementos do AV serão capazes de reagir e realizar decisões sobre como agir.

A Figura 24 mostra as dependências entre as tarefas do Projeto de Arquitetura Interna dos Componentes.

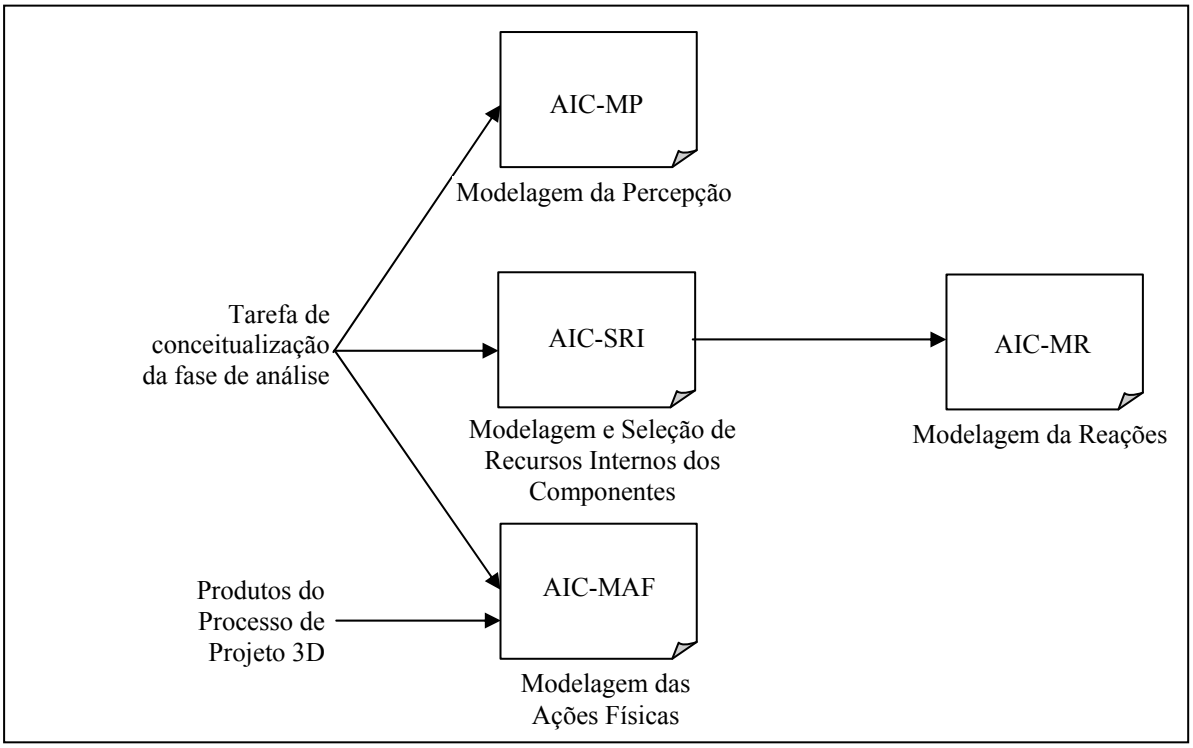

Figura 24: Tarefas de Projeto da Arquitetura Interna de Componentes (SÁNCHEZ-SEGURA, 2001)

5.4 Processo: Projeto do Sistema

Este processo é similar ao processo de projeto orientado a objetos.

5.4.1 Projeto de Interface

Nesta tarefa um protótipo da interface deve ser desenvolvido.

5.4.2 Modelo Estático Expandido

O modelo estático desenvolvido neste processo de análise deve ser expandido nesta tarefa.

5.4.3 Modelo Dinâmico Expandido

O modelo dinâmico desenvolvido no processo de análise deve ser expandido nesta tarefa.

5.4.4 Descrições Detalhadas dos Métodos

As ações identificadas no Processo de Projeto da Arquitetura Interna de Componentes são detalhadas em pseudo-código.

5.4.5 Projeto de Dados Persistentes

Deve ser definido o meio pelo qual os dados persistentes do comportamento dos habitantes são gerenciados.

5.5 Dependências das tarefas de projeto do sistema

A Figura 25 mostra as dependências entre as tarefas do Projeto de Sistema.

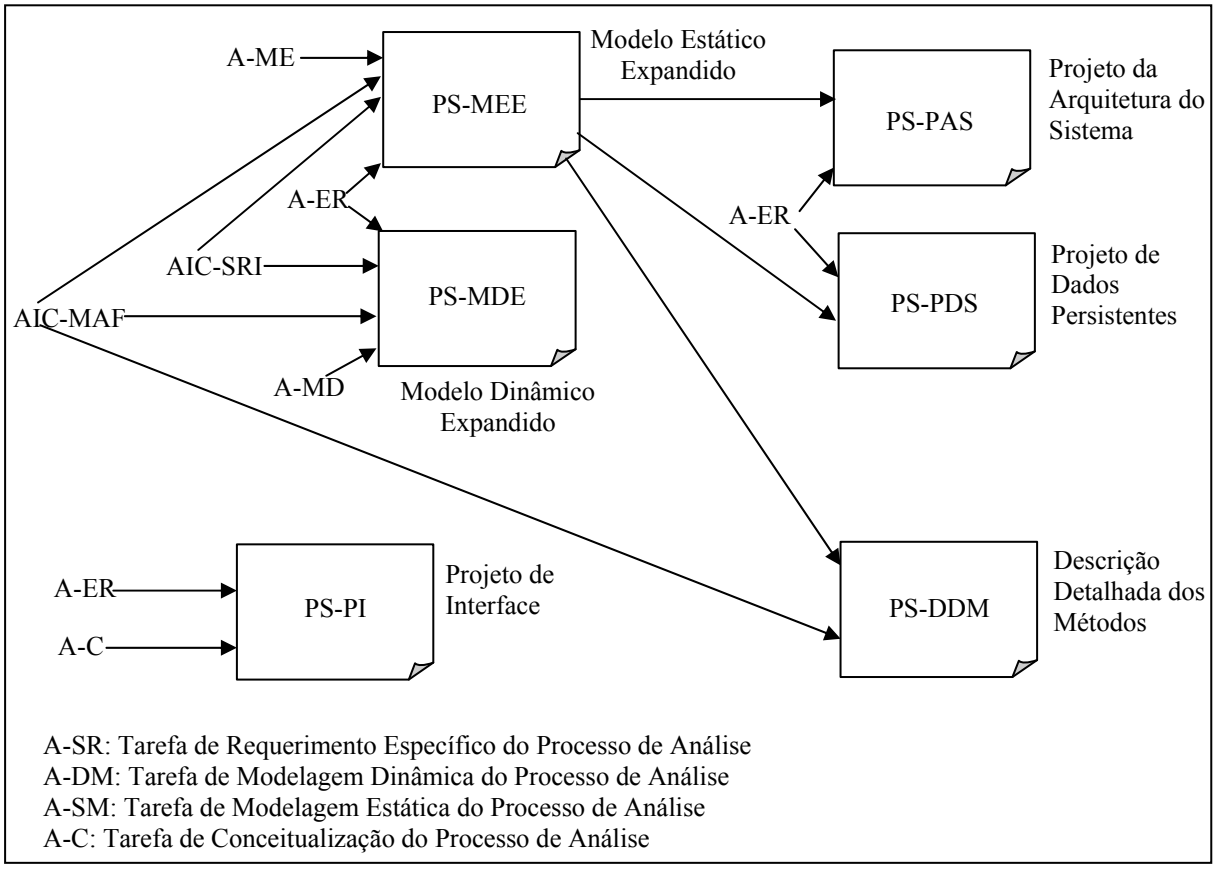

Figura 25: Tarefas do Projeto do Sistema (SÁNCHEZ-SEGURA, 2001)

Na Figura 25 aparecem as referências às tarefas de análise, bem como as tarefas de implementação e gerenciamento. Todas essas etapas são parte do processo global definido para o desenvolvimento formal de AVs.

5.6 Relacionamento entre os processos do projeto

A Figura 26 exemplifica o relacionamento entre o processo de projeto e o processo de análise.

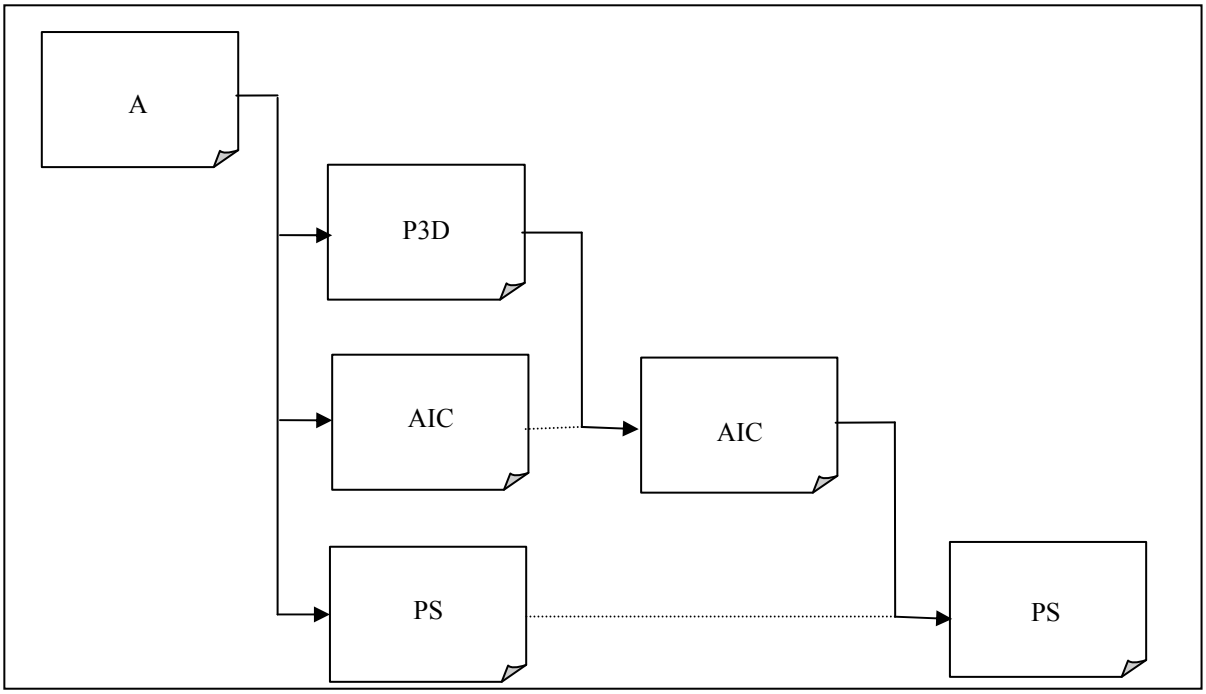

Figura 26: Relação entre os processos de Análise e Projeto (SÁNCHEZ-SEGURA, 2001)

Os processos Projeto 3D, Projeto da Arquitetura Interna de Componentes e Projeto de Sistema podem iniciar quando o processo de Análise termina, mas algumas tarefas do Projeto da Arquitetura Interna de Componentes podem não iniciar até o Projeto 3D finalizar e algumas tarefas do Projeto de Sistema podem não iniciar até o processo do Projeto da Arquitetura Interna de Componentes finalizar.

Por exemplo, a etapa da Modelagem das Ações Físicas pode não iniciar até o Projeto 3D não acabar, porque a definição física das ações dependem da aparência dos modelos 3D e a etapa das Descrições Detalhadas dos Métodos pode não iniciar até a Modelagem das Ações Físicas estar completada; isto se deve ao fato de que o pseudo-código que representa os métodos a serem implementados dependem da maneira que as ações serão representadas.

Estes relacionamentos podem ser demonstrados no seguinte estudo de caso:

1. Num sistema de AV, uma estrutura de avatar é definida na etapa da Modelagem de Avatar com duas pernas e um corpo, e o avatar é definido com a habilidade de pular.

2. Na etapa de Modelagem de Ações Físicas, a maneira com que o avatar pula deve ser descrito como sendo: “o corpo salta para cima e os braços se elevam”.

3. Na etapa das Descrições Detalhadas dos Métodos, a ação de saltar é descrita em pseudo-código:

```
Procedimento Saltar (Avatar)
{
          Avatar.Corpo = 50 pixels sobre o chao
          ...
    }
```

Na maioria dos desenvolvimentos de AV, a maior dedicação se dá na fase de implementação, porém, mesmo sendo uma parte importante do desenvolvimento, esta deveria ser a última fase do processo. No caso, das etapas anteriores serem ignoradas, o AV desenvolvido não será facilmente reutilizável, devido à inexistência de documentação sobre a especificação, a análise e o projeto dos componentes e seus requisitos.

CAPÍTULO 6 – IMPLEMENTAÇÃO DO PROJETO PROPOSTO

Neste capítulo serão apresentados a especificação, análise, projeto e implementação de um protótipo para validar a metodologia do sistema proposto.

6.1 Visualização do sistema em camadas

Para um melhor entendimento da arquitetura do sistema, este foi dividido em camadas lógicas para separar a interface, a lógica da aplicação (*middleware*) e a persistência de informações, num ambiente multi-camadas.

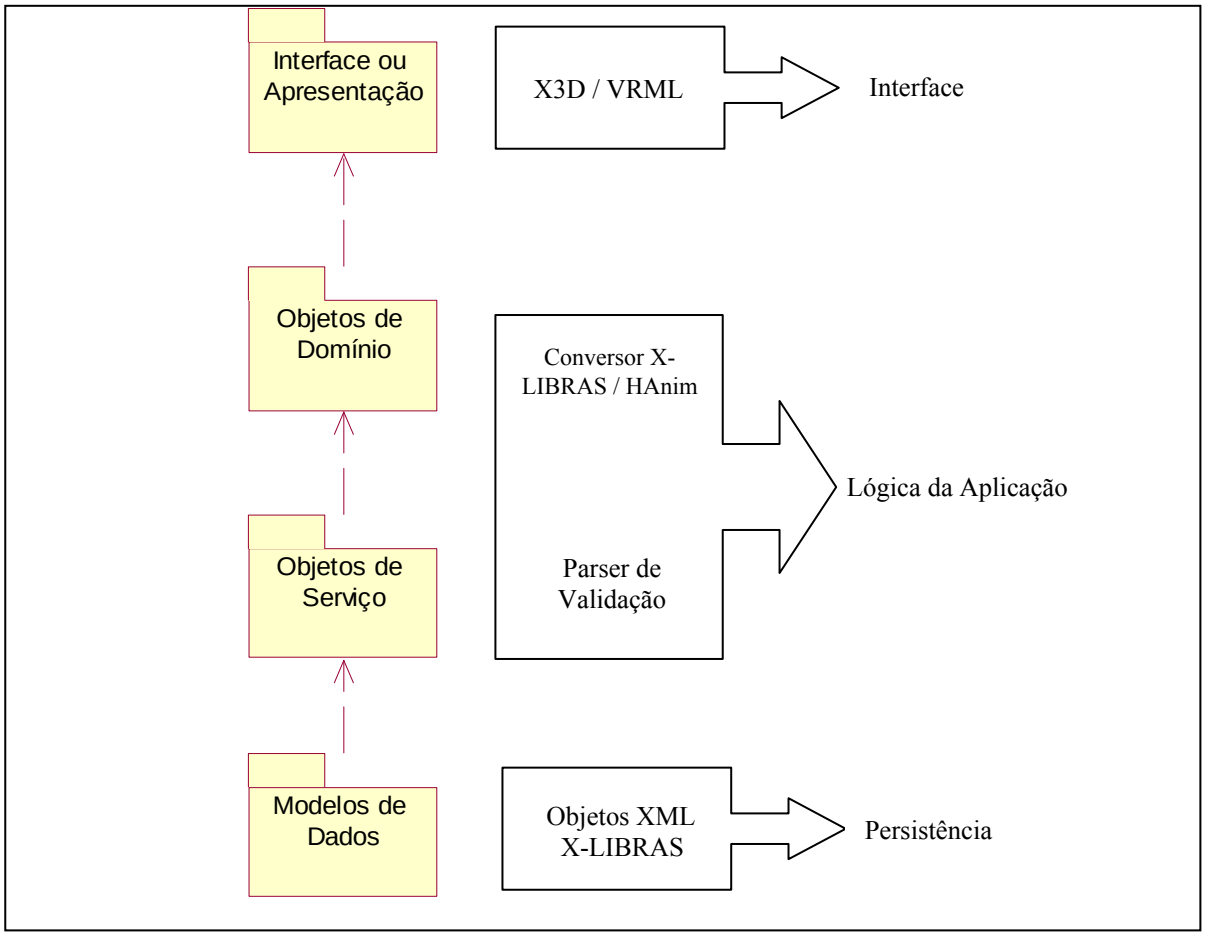

Figura 27: Camadas lógicas do sistema

A Figura 27 mostra as camadas lógicas e suas representações dentro do sistema.

Na camada de persistência, serão armazenados em documentos XML, por meio do vocabulário X-LIBRAS, as informações concernentes ao sinais LIBRAS que serão especificadas no tópico sobre a implementação do vocabulário X-LIBRAS. Na camada de lógica da aplicação, é definida a validação dos documentos XML através do *parser* e da API DOM (objetos de serviço), utilizando o DTD X-LIBRAS ou o *Schema* X-LIBRAS e, então, mapeado o sinal para sua representação no H-Anim, através do conversor X-LIBRAS (objeto de domíno), utilizando o X3D ou VRML. E, finalmente, na camada da interface, o sinal LIBRAS é apresentado ao usuário num ambiente sintético virtual tridimensional.

6.2 Especificação da camada lógica da aplicação

Como parte do processo de engenharia de software de um sistema, a camada de lógica da aplicação do sistema em questão foi particionado em módulos para sua especificação, como visto na Figura 28.

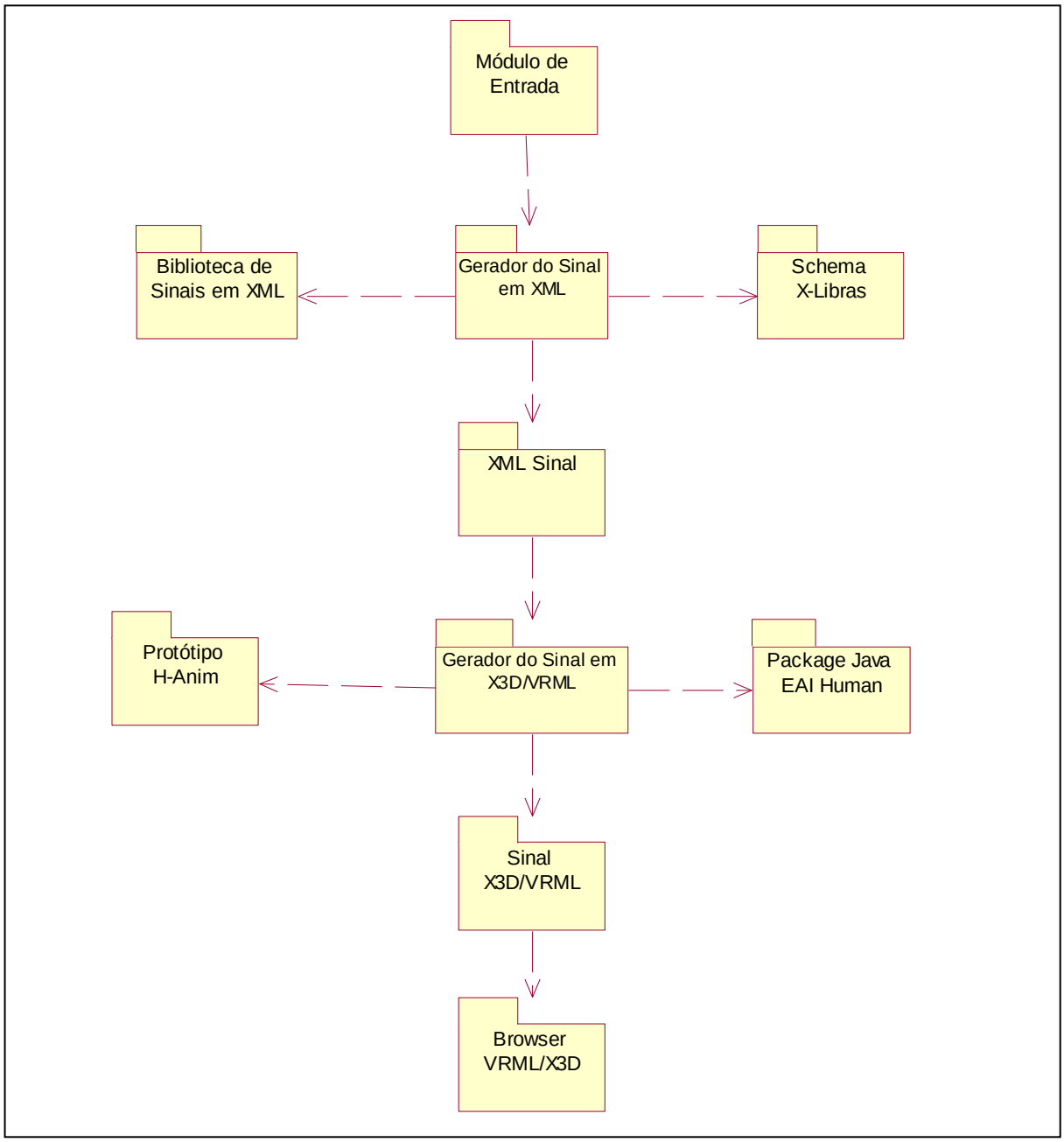

Figura 28: Módulos do sistema proposto

A partir do módulo de entrada, o usuário escolhe o sinal desejado, gerando, então, o acesso à base XML de sinais especificada, que é validada pelo *Schema* ou DTD X-LIBRAS; este sinal é mapeado para um documento X3D/VRML, tendo como base o protótipo H-Anim. Essa manipulação do mundo virtual pode ser feita através de uma biblioteca externa Java utilizando a interface EAI (*External Authoring Interface*) (KIMEN, 1997) no caso do VRML e SAI (*Scene Access Interface*) (ISO/IEC FCD 19777-2:200x) no X3D.

A metodologia UML foi empregada para definir as interações do usuário com o sistema (Definição dos Requisitos Funcionais da Aplicação). Os diagramas de Use Case e de Classes da UML (BOOCH, RUMBAUGH, JACOBSON, 2000) foram construídos utilizando-se a ferramenta Rational Rose.

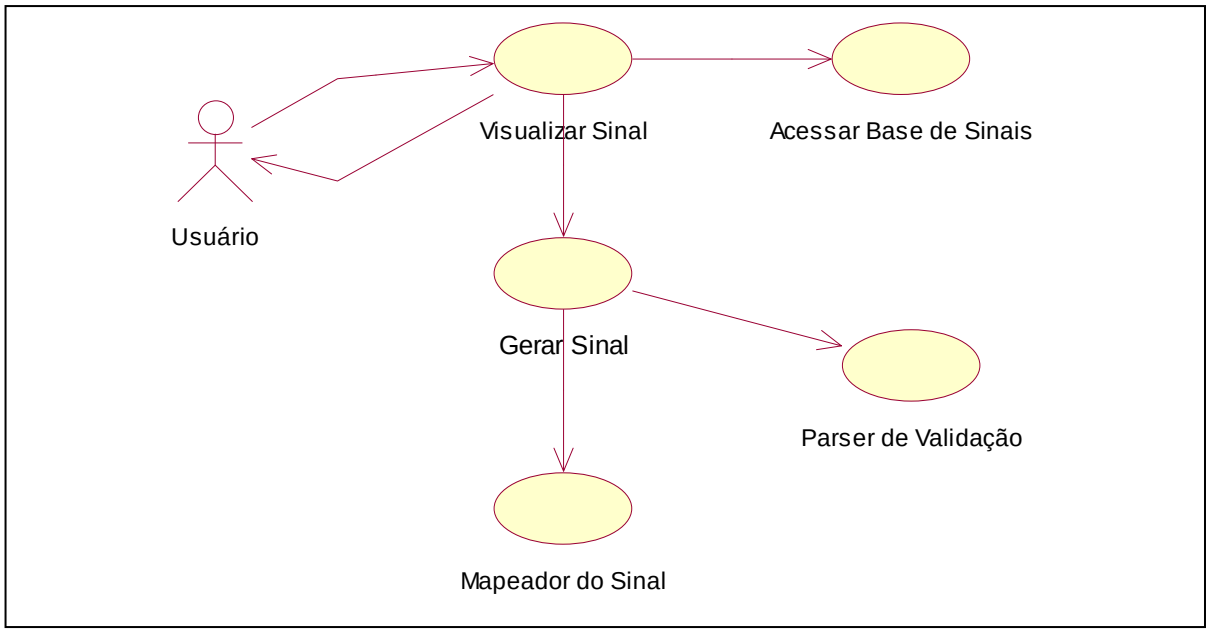

Figura 29: Diagrama Use Case da aplicação

A Figura 29 mostra o diagrama de casos de uso da aplicação, demonstrando a interação do usuário com a aplicação e os processos disparados através dessa interação.

6.3 Modelagem de Percepção

Nesta tarefa é definida a maneira com a qual avatares e agentes podem perceber a presença de outros habitantes e objetos no AV, por não haver outros agentes ou objetos no ambiente e o avatar se relacionar diretamente com o usuário, não há necessidade de especificação desta etapa.

6.4 Modelagem das expressões faciais

A modelagem das expressões faciais do sinal no sistema em questão é de extrema importância, pois uma expressão pode definir um sinal no conjunto dos elementos mínimos do sinal em LIBRAS.

Tão importante quanto a funcionalidade de um programa de computador é seu aspecto final, sua apresentação, pois é ele que praticamente cativa o usuário. Por isso, como observado em Campos (1998), devem ser evitados problemas, como o uso de uma comunicação não eficaz, desorganização, falta de padronização, uso extravagante de cores, tamanhos inadequados de letras e desenhos, uso não adequado de tecnologias disponíveis, entre outros.

Ainda segundo Campos (1998), para a construção de *softwares* deve-se, antes, verificar quais as necessidades dos usuários e avaliar quais as preferências destes quanto a um ou outro sistema de representação para comunicação; então, projetar a interface do programa. Assim, para o desenvolvimento de *softwares* destinados a pessoas com deficiência auditiva, deve-se respeitar os requisitos de interface, apresentando as informações de forma gráfica, sem exigir do usuário conhecimento prévio da escrita de línguas orais e de sinais.

Dentro desta perspectiva de definição de um avatar com um maior realismo facial, o projeto SMILE (*Multlayered Facial Animation System*) (KALRA, et.al., 1991) define um sistema de animação facial que divide o rosto em um conjunto de fragmentos, como pode ser visto na Figura 30, que pode ser deformado pela mudança de um fator de intensidade normalizado. Estas regiões são baseadas na noção de Ações Perceptivas Mínimas (APM) que definem o movimento básico que o rosto pode realizar.

Uma animação facial final (expressão significante) será uma composição destas mudanças APMs através do tempo. Estas regiões podem receber parâmetros de animação facial a serem deformadas.

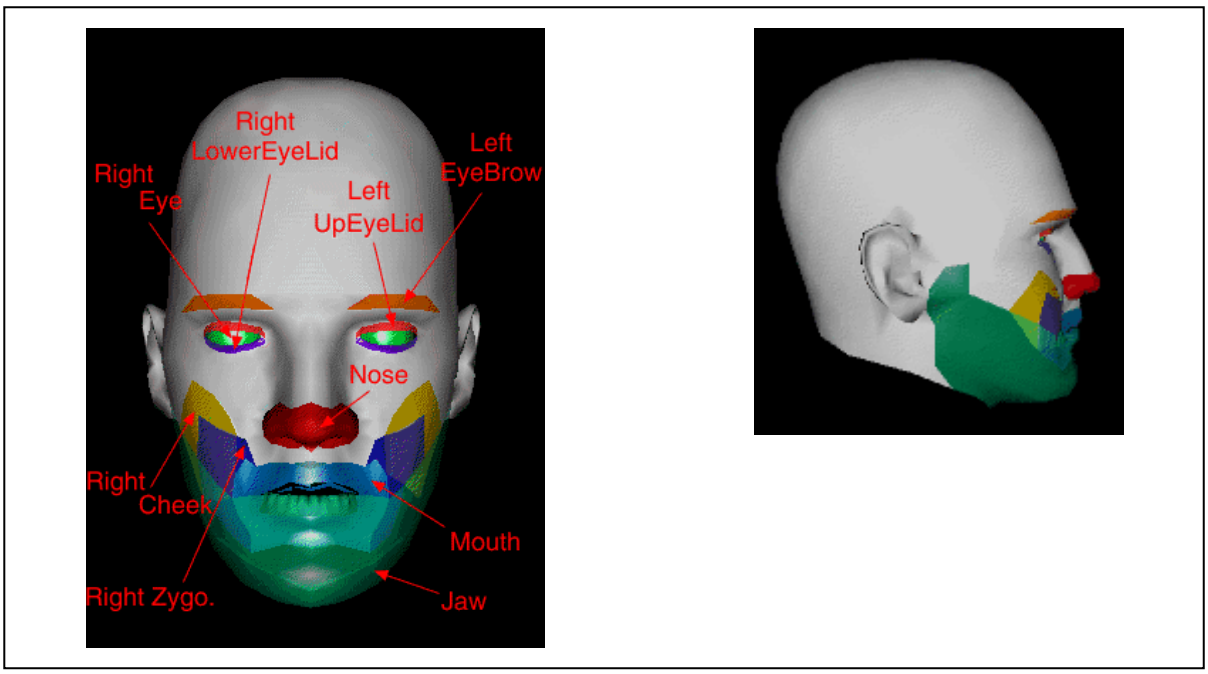

Figura 30: Divisão do rosto no sistema SMILE

Este recurso pode ser adaptado a este sistema para resolver alguns problemas presentes no X3D, principalmente na maneira de organizar a superfície final. Para minimizar o tamanho do sistema de animação (baseado nos nós *PositionInterpolator*), a melhor solução é animar somente partes que são efetivamente alteradas.

Mais da metade da cabeça é estática, mas fazendo assim, a continuidade entre as partes que são móveis e as partes estáticas não é mais assegurada. Uma solução é definir uma sub-superfície circunvizinha que incluirá todos as superfícies APMs, mais uma zona estática fina que assegurará a manutenção da continuidade com as partes estáticas completas da cabeça. Então, o arquivo de animação será composto somente pela superfície circunvizinha; tal solução minimiza o número de pontos necessários para realizar uma animação facial.

Na Figura 31, pode-se visualizar regiões específicas, olho direito (vermelho), olho esquerdo (roxo), dentes (alaranjado), rosto móvel (azul) e rosto estático (verde).

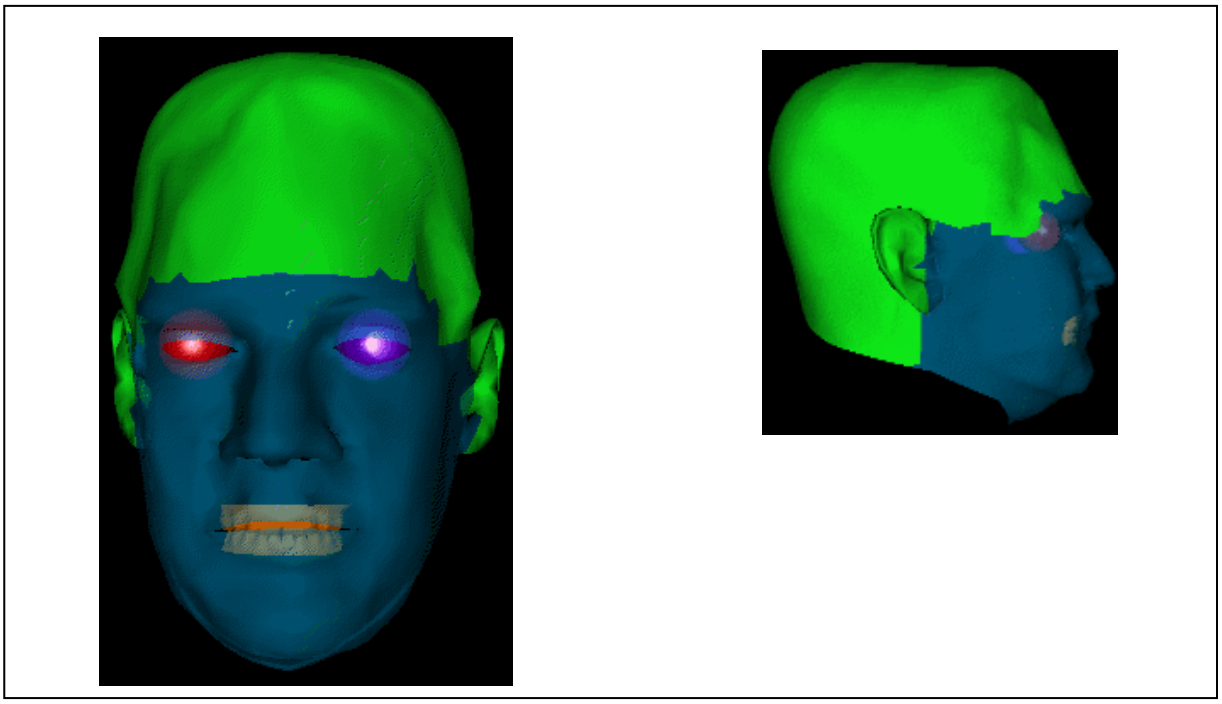

Figura 31: Superfícies de manipulação do rosto

Um outro problema concerne ao material: a superfície circunvizinha também pode ter que ser dividida em regiões baseadas em materiais, se queremos recuperar as cores dos olhos ou dos lábios (isto também multiplicará os *ROUTES* necessários para executar uma animação facial) (KALRA, et.al., 1992). Um meio muito evidente para evitar dividir superfícies móveis é utilizar um mapa de textura (*Texture Map*). A textura moverá com os pontos quando a animação facial for executada e permitirá receber um resultado realista apenas pelo uso de uma ferramenta de mapeamento de textura.

6.4.1 Animações faciais com o protótipo Displacer

O objeto *Displacer* (ISO-IEC-19775-FDIS-X3DABSTRACTSPECIFICATION) é um *container* com informações que especificam os deslocamentos permitidos de partes da figura H-Anim.

```
interface Displacer {
sequence<int> coordIndex
sequence<float> displacements
string name
float weight
}
```

Figura 32: Objeto *Displacer*

Como mostrado na Figura 32, cada objeto Displacer tem um campo *name* requerido para identificação do objeto. Todos os outros campos são opcionais. Dentro do escopo local de um objeto *Humanoid*, cada objeto *Displacer* pode ser referenciado pelo seu nome sozinho (por exemplo, l_eyebrow_feature ou l_eyebrow_raiser_action). Entretanto, quando se referenciar com um objeto *Displacer* dentro de um escopo global, o nome do objeto *Humanoid* deve ser adicionado como um prefixo distinguível.

Por exemplo, se um campo *name* de um objeto *Humanoid* contém o valor "joe", o objeto *Displacer* l_eyebrow_feature deste humanóide é referenciado globalmente como "joe_l_eyebrow_feature".

Se somente um humanóide é definido dentro de um escopo global, este campo *name* do objeto *Humanoid* contém o valor "hanim". Em tal caso, o objeto *Displacer* l_eyebrow_feature deste humanóide deveria ser referenciado globalmente como "hanim_l_eyebrow_feature".

Objetos *Displacer* que são utilizados para identificar recursos, devem ter um nome com um sufixo "_feature". Objetos *Displacer* que são utilizados para mover um recurso terão no nome um sufixo "_action", normalmente com um pré-sufixo adicional para indicar o tipo de movimento (como "l_eyebrow_raiser_action"). Objetos *Displacer* que correspondem a um configuração particular de vértices deve ter um sufixo "_config".

O campo *coordIndex* contém os índices no *array* de coordenadas para o *Mesh* dos vértices que são afetados pelo *Displacer*. Este recurso é especificado na Figura 33.

```
Displacer {
    coordIndex [ 7, 12, 21, 18 ]
    name "l_eyebrow_feature"
}
```

Figura 33: Definição do campo *coordindex*

Neste exemplo, os vértices 7, 12, 21 e 8 do *Mesh* significam que são da sobrancelha esquerda.

O campo *displacements*, se presente, providencia um conjunto de valores 3D que são adicionados à posição neutra ou em descanso de cada vértice referenciado no campo *coordIndex* do *Mesh* associado. Estes valores correspondem um-a-um com os valores do *array coordIndex* do *Mesh*. Os valores serão os *displacements* máximo e a aplicação fará o escalonamento utilizando os valores do campo *weight* antes de adicioná-los às posições de vértice neutro.

```
Displacer {
   coordIndex [ 7, 12, 21, 18 ]
   displacements [ 0 0.0025 0, 0 0.005 0, 0 0.0025 0, 0 0.001 0 ]
   name "l_eyebrow_raiser_action"
}
```

Figura 34: Definição do campo *displacements*

A Figura 34 mostra um exemplo que elevaria o vértice quatro da sobrancelha esquerda na direção vertical. O vértice 7 seria deslocado para cima em 2 milímetros na direção vertical (Y), o vértice 12 seria posicionado para cima em 5 milímetros, o vértice 21 seria deslocado para cima em 2,5 milímetros, e o vértice 18 seria posicionado apenas em um milímetro.

A aplicação uniforme escalaria o *displacements* pelo valor do campo *weight*.

Por, dinamicamente, modificar o valor dos campos *weight* dos objetos *Displacer*, uma aplicação pode distorcer o objeto *Mesh* utilizando uma combinação linear dos destinos de distorção definidos nos objetos *Displacer*. Para estes objetos *Displacer* que estão contidos no

campo *displacers* dos objetos *Segment*, os *displacements* são definidos e aplicados ao sistema de coordenadas *Segment*. O *Mesh* base para o *Segment* distorcido é o *Mesh* original definido no *Segment*. Para os objetos *Displacer* que estão contidos no campo *displacers* dos objetos *Joint*, os *displacements* são aplicados para *Mesh* deformado especificado no campo *skinCoord* do objeto *Humanoid*.

Os *displacements* são definidos no sistema de coordenadas local do objeto *Joint* que contém o *Displacer*. Os *displacements* configurados serão aplicados no sistema de coordenadas do objeto *Humanoid*. Então, o *displacements* serão transformados do objeto *Joint* para o objeto *Humanoid*.

A Figura 35 mostra animações faciais realizadas com o protótipo *Displacer*, estes exemplos foram implementados na ferramenta Vizx3D.

Figura 35: Animações faciais com o protótipo *Displacer*

6.5 Alternativa para a implementação das expressões faciais

A realidade misturada é uma modalidade de RV (TAMURA, YAMAMOTO e KATAYAMA, 2001) e pode ser definida, segundo Milgram e Kishino (1994a), como a união de mundos reais e mundos virtuais. Sistemas de realidade misturada têm quebrado barreiras entre o mundo virtual e o mundo real (BROWN, 2003) e abrange conceitos de realidade aumentada e virtualidade aumentada (MILGRAM e KISHINO, 1994a) (TAMURA, YAMAMOTO e KATAYAMA, 2001).

A realidade aumentada envolve a sobreposição de informações digitais (por exemplo, texto, ou gráficos) sobre cenas do mundo real, de modo que informações digitais e objetos reais pareçam estar no mesmo ambiente, mesmo quando em movimento. A cena física pode ser um ambiente local, com informações digitais introduzidas e visualizadas por meio de um *Head Mounted Display* – HMD. Alternativamente, a visualização pode ser feita por meio de monitor de vídeo, onde a visualização da cena real é ampliada com a adição de informações digitais (KOLEVA, BENFORD e GREENHALGH, 1999).

A virtualidade aumentada, em oposição à realidade aumentada, incrementa ambientes virtuais com dados do mundo real, por exemplo, vídeo ou mapeamento de texturas (MILGRAM, 1994b) (YAMAMOTO, 1999).

No escopo da virtualidade aumentada, a utilização de vídeo de uma pessoa real inserida em um AV, denominada vídeo avatar, possibilita a comunicação visual com alto grau de co-presença e realça a interação com o participante da aplicação, como por exemplo, sistemas de teleconferência (WANG, 1998) (RAJAN, 2002). Segundo Yura, Usaka, e Sakamura (1999), a comunicação visual traz três vantagens principais:

- identificação: possibilita o reconhecimento do usuário pelos demais participantes, o que não ocorre com o avatar tradicional gerado pelo computador;

- expressão facial: possibilita ao usuário poder demonstrar informações emocionais de forma direta aos demais usuários;

- gestos: permite a melhoria da comunicação entre os usuários por meio de expressões corporais, principalmente quando conversam sobre o ambiente e os objetos que os rodeiam.

A inserção de vídeo em ambientes virtuais pode ser realizada por meio de modelagem, ou reconstrução tridimensional da imagem do avatar (THALMANN, 1999). Dentro desse conceito, foi especificado uma solução que se baseia na inserção do vídeo sobre

uma imagem 3D no AV, através do nó *MovieTexture*, especificado na próxima seção, sem a necessidade do processo de reconstrução tridimensional.

6.5.1 MovieTexture

O nó *MovieTexture* define um mapa de textura dependente de tempo (contendo um arquivo de filme) e parâmetros para controlar o filme e o mapeamento da textura. Um nó Movietexture pode ser utilizado também como a fonte dos dados de som para um nó Sound. Neste caso em especial, o nó *MovieTexture* não é utilizado para renderização. O nó *MovieTexture* está especificado na Figura 36.

```
MovieTexture : X3DTexture2DNode, X3DSoundSourceNode, X3DUrlObject {
  SFBool   [in,out] loop             FALSE
  SFNode   [in,out] metadata         NULL  [X3DMetadataObject]
  SFTime   [in,out] resumeTime       0     (-∞,∞)
  SFTime   [in,out] pauseTime        0     (-∞,∞)
  SFFloat  [in,out] speed            1.0   (-∞,∞)
  SFTime   [in,out] startTime        0     (-∞,∞)
  SFTime   [in,out] stopTime         0     (-∞,∞)
  MFString [in,out] url              []    [urn]
  SFBool   []       repeatS          TRUE
  SFBool   []       repeatT          TRUE
  SFTime   [out]    duration_changed
  SFTime   [out]    elapsedTime
  SFBool   [out]    isActive
  SFBool   [out]    isPaused
}
```

Figura 36: Especificação do nó *MovieTexture*

O campo *url*, que define o filme deve suportar os formatos de arquivo de vídeo MPEG1-System (áudio e vídeo) ou MPEG1-Video (somente vídeo). Os nós *MovieTextures* podem ser referenciados pelo campo *texture* do nó *Appearance* e pelo campo *source* do nó *Sound*.

Assim que o filme é carregado, um campo *duration_changed* é enviado. Este indica a duração do vídeo em segundos. Este valor do campo pode ser lido (por exemplo, por um nó *Script*) para determinar a duração de um vídeo. Um valor de “-1” implica que o vídeo ainda

não foi carregado ou que o valor está indisponível por alguma razão. O ciclo de um nó *MovieTexture* é o tamanho do tempo em segundos para uma execução do vídeo especificado no campo *speed*. A Figura 37 mostra o exemplo de um vídeo avatar implementado em X3D.

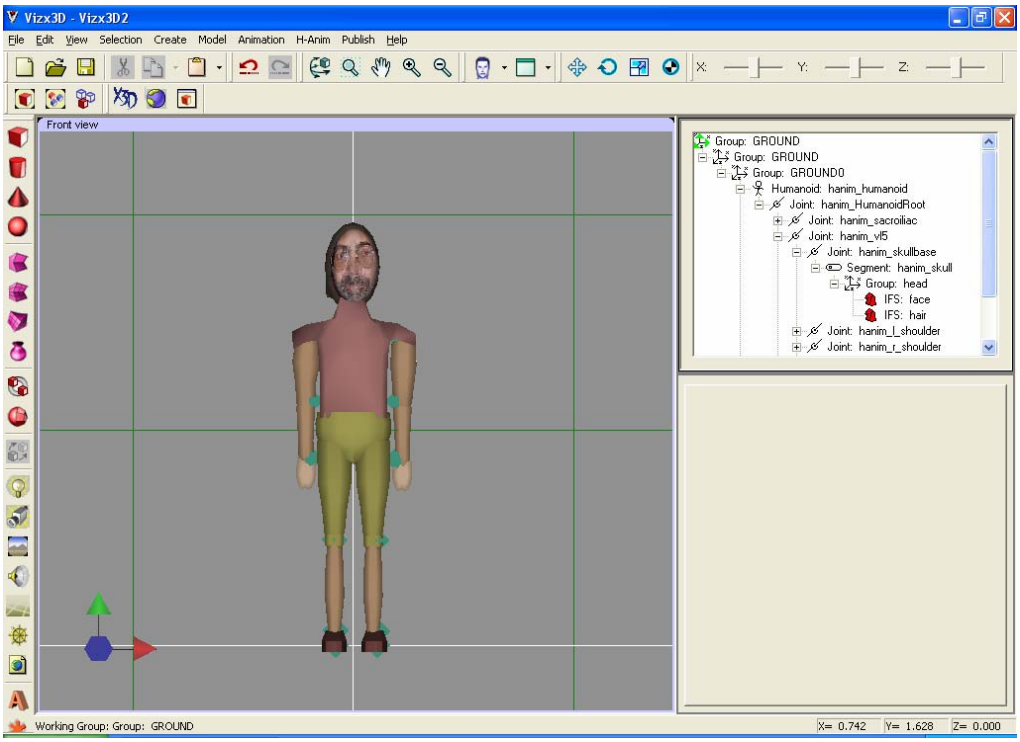

Figura 37: Exemplo de um vídeo avatar

6.6 Projeto de interface

Dentro da abordagem vista no capítulo sobre Metodologia de Modelagem de Ambientes Virtuais, a Camada de Apresentação ou Interface é descrita na etapa Projeto 3D, com as tarefas Modelagem do AV e Modelagem de Avatares e na etapa Projeto do Sistema na tarefa Projeto de Interface.

Para o AV proposto no projeto é necessária a especificação de um avatar humanóide que contenha a habilidade de realizar os sinais da LIBRAS, permitindo ao usuário uma visão tridimensional do sinal e por conseqüência, um melhor entendimento de sua natureza.

Para tanto, foi necessária, primeiramente, a modelagem do avatar que serviu como protótipo para implementação dos sinais e depois o estudo e a implementação da animação desse avatar.

6.6.1 Modelagem de avatares

A implementação do avatar baseia-se na especificação *HAnim* para modelar o corpo de um humanóide. A escolha desta especificação deveu-se ao fato da mesma definir uma forma padrão para representar humanóides e de ser de implementação bastante acessível e flexível.

A especificação *HAnim* define uma forma padrão de representar humanóides em VRML ou X3D. O objetivo da especificação é permitir que humanóides criados com ferramentas de autoria possam ser animados utilizando ferramentas de um outro autor.

A especificação *HAnim* define o corpo humano como um conjunto de segmentos (como braço, pé, mão) que estão ligados uns aos outros por articulações (como cotovelo, pulso, tornozelo). Cada humanóide contém um conjunto de articulações organizadas hierarquicamente. Cada nó articulação pode conter outros nós articulação e pode também conter um nó segmento que representa a parte do corpo associada a essa articulação. A Figura 38 mostra um diagrama com todas as articulações definidas pela especificação bem como a sua estrutura hierárquica. A articulação *HumanoidRoot* é a raiz de toda a hierarquia de articulações. O corpo é dividido em duas seções principais: a parte superior e a parte inferior. Todas as outras articulações são filhas desta articulação. Transformações aplicadas a este nó afetarão todo o corpo humano. Por exemplo, mover o humanóide é conseguido aplicando translações a este nó.

A figura mostra ainda que não é preciso implementar todas as articulações para que uma implementação esteja de acordo com a especificação *HAnim*. O projeto em questão especifica um conjunto mínimo de articulações de forma a permitir animar as partes inerentes à definição de um sinal da LIBRAS.

Por não haver *browsers* ou *plug-ins* que executem aplicações X3D utilizando o perfil H-Anim, a implementação do protótipo foi realizada com avatares H-Anim desenvolvidos em VRML através de classes Java (WITHERS, 1999). Porém, também foi implementado o avatar em X3D utilizando o *software* VizX3D da Virtock Technologies, Inc.[1] que permite a visualização de aplicações X3D.

[1] http://www.vizx3d.com

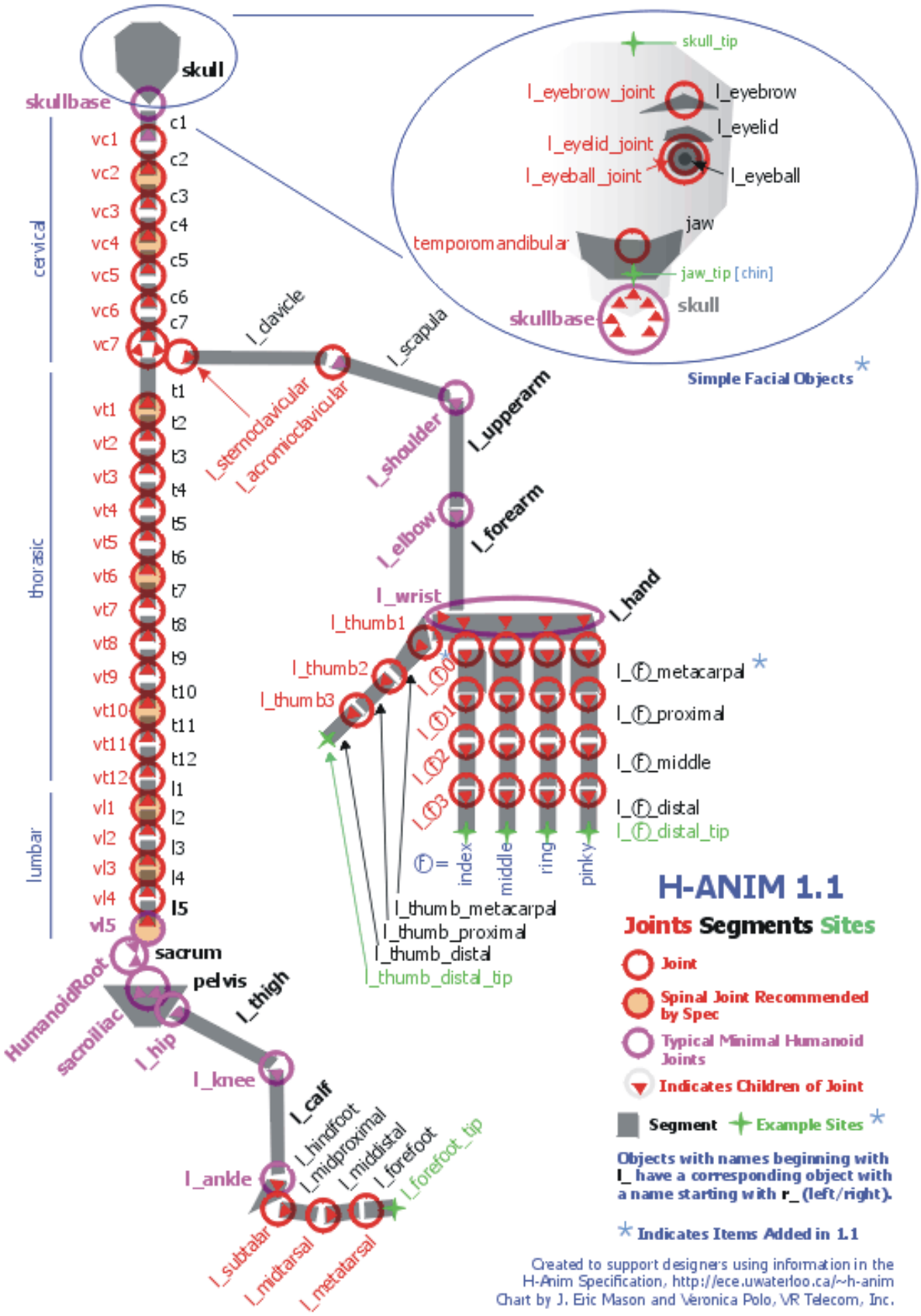

Figura 38: Hierarquia de nós do H-Anim (HUMANOID ANIMATION WORKING GROUP, 2002)

A hierarquia de articulações torna extremamente simples animar partes do corpo de um humanóide. Basta mudar o ângulo da articulação (efetuando rotações), e todas as outras articulações "filhas", bem como os respectivos segmentos, serão afetados pela transformação, permitindo assim mover as diversas partes do corpo. Quanto mais articulações forem implementadas, mais articulado será o humanóide, o que permitirá uma maior mobilidade das várias partes do corpo. Foi esta simplicidade de manipulação que levou à sua utilização.

6.6.2 Especificação do Objeto *Joint*

Cada articulação no corpo é representada por um nó *Joint* que é utilizado para definir relações entre segmentos do corpo. A definição do nó PROTO *Joint* é definida na Figura 39:

```
PROTO Joint [
  exposedField SFString name ""
  exposedField SFVec3f center 0 0 0
  exposedField SFRotation rotation 0 0 1  0
  exposedField SFVec3f scale 1 1 1
  exposedField SFRotation scaleOrientation 0 0 1  0
  exposedField SFVec3f translation 0 0 0
  exposedField MFFloat ulimit []
  exposedField MFFloat llimit []
  exposedField SFRotation limitOrientation 0 0 1 0
  exposedField MFFloat stiffness [ 1 1 1 ]
  exposedField MFNode children []
]
{
  Transform {
    translation IS translation
    rotation IS rotation
    scale IS scale
    scaleOrientation IS scaleOrientation
    center IS center
    children IS children
  }
}
```

Figura 39: Especificação do objeto *Joint*

O campo *name* serve para identificar as articulações de forma a que as aplicações possam manipulá-las em *run-time*.

Um nó *Joint* também é utilizado para conter informação específica das articulações. Essa informação pode ser utilizada por aplicações para animar o humanóide. Os campos *children* servem para conter informação acerca da manipulação da rotação que uma dada articulação pode ter. Os restantes dos campos possuem o mesmo significado que os campos de um nó *Transform*, tendo sido esse o nó utilizado para implementar o *PROTO Joint*. No entanto, convém notar que o campo *center* contém a posição do centro de rotação da articulação, relativa à raiz da descrição do corpo do humanóide. E como as localizações dos centros das articulações estão todas no mesmo sistema de coordenadas, o comprimento de cada segmento pode ser determinado calculando a distância entre o centro do nó *Joint* pai e o centro do nó *Joint* filho. A única exceção será para os segmentos terminais, como as pontas dos dedos das mão e dos pés, por exemplo. As localizações dos centros das articulações utilizadas nesta implementação estão de acordo com o que é aconselhado na especificação.

6.6.3 Especificação do Objeto *Segment*

Cada segmento do corpo é guardado num nó *Segment*. Segundo a especificação, um nó *Segment* deverá ser tipicamente implementado utilizando um nó *Group* de forma a conter um número de figuras ou de transformações que posicionem a parte do corpo correspondente ao sistema de coordenadas do corpo. No entanto, uma vez que o avatar tem a sua dimensão pré-definida pelas posições dos centros das articulações implementadas, e pretendia-se que o aspecto do avatar pudesse ser facilmente alterado, optou-se por implementar o *PROTO Segment* como mostrado na Figura 40. Desta forma, a implementação do humanóide terá de início todas as articulações e segmentos devidamente posicionados.

Note-se, no entanto, que a dimensão das figuras associadas aos segmentos terá de estar de acordo com as dimensões definidas pelas distâncias dos centros das articulações associadas a esse segmento.

```
PROTO Segment [
  exposedField SFString name ""
  exposedField SFVec3f centerOfMass 0 0 0
  exposedField SFVec3f momentsOfInertia 1 1 1
  exposedField SFFloat mass 0
  exposedField MFNode children []
  exposedField SFNode coord NULL
  exposedField MFNode displacers []
  eventIn MFNode addChildren
  eventIn MFNode removeChildren
]
{
  Group {
    children IS children
    addChildren IS addChildren
    removeChildren IS removeChildren
  }
}
```

Figura 40: Especificação do objeto Segment.

6.6.4 Especificação do Objeto *Humanoid*

O nó *Humanoid* é utilizado para guardar informação legível como os dados dos autores, guardar as referências para as articulações e segmentos, servindo como um *container* para todo o humanóide. Também oferece uma forma de se mover o humanóide através dos ambientes virtuais.

```
PROTO Humanoid [
  exposedField SFString name ""
  exposedField MFString info []
  exposedField SFString version "1.1"
  exposedField MFNode joints []
  exposedField MFNode segments []
  exposedField MFNode sites []
  exposedField MFNode viewpoints []
  exposedField MFNode humanoidBody []
  exposedField SFVec3f center 0 0 0
  exposedField SFRotation rotation 0 0 1 0
  exposedField SFVec3f scale 1 1 1
```

```
  exposedField SFRotation scaleOrientation 0 0 1 0
  exposedField SFVec3f translation 0 0 0
]
{
  Transform {
    center IS center
    rotation IS rotation
    scale IS scale
    scaleOrientation IS scaleOrientation
    translation IS translation
    children [
      Group {
        children IS viewpoints
      }
      Group {
        children IS humanoidBody
      }
    ]
  }
}
```

Figura 41: Especificação do objeto *Humanoid*

Como visto na Figura 41, o campo *humanoidBody* serve para conter o nó *HumandoidRoot*, que é o nó articulação raiz de toda a hierarquia de articulações. Os campos *joints* e *segments* servem para conter referências para os nós *Joint* e *Segment* do humanóide. O campo *viewpoints* serve para conter referências para os nós *Viewpoints* que possam estar associados a partes do corpo do humanóide. Estes nós podem ser utilizados para que o utilizador possa observar as animações de vários pontos de vista, por exemplo. Na implementação proposta, os nós *Viewpoint* são referenciados para permitir ao usuário uma visualização de diversos pontos de vista do sinal 3D. No entanto esses nós não podem ser inseridos no campo *viewpoints* do humanóide, porque todos os nós inseridos nesse campo estão sujeitos a todas as transformações efetuadas no humanóide, fato esse que interferiria com os mecanismos utilizados para controlar o avatar.

6.6.5 Especificação da hierarquia de articulações

Como já foi referido anteriormente, o objetivo era definir um avatar (humanóide) que o utilizador pudesse controlar. A utilização da especificação *Hanim* permitiu definir um

humanóide capaz de ser movido com animações associadas. A especificação *Hanim* não aborda aspectos de controle dos humanóides, no entanto, a forma como estão definidos os nós *PROTO* permite desenvolver aplicações que manipulem em *run-time*, os nós do humanóide. Essas aplicações deverão recorrer, muito possivelmente, a interfaces oferecidas pelos *browsers* como o EAI, que define um conjunto de funções no *browser* que o ambiente externo pode utilizar para afetar o AV. A Figura 42 mostra a definição da hierarquia de articulações, exemplificando até o segmento hanim_c7.

```
DEF humanoid Humanoid {
  humanoidBody [
  DEF hanim_HumanoidRoot Joint{
    name "HumanoidRoot"
    center 0.0 0.707 -0.024
    children [
      DEF hanim_sacrum Segment {
        name "sacrum"
        children [] }
      DEF hanim_vc7 Joint{
        name "vc7"
        center -6.631E-4 0.79796004 -0.0019899998
        children [
          DEF hanim_c7 Segment {
            name "c7"
            children [
              Shape { appearance Appearance { texture ImageTexture { url
"texturas/torax.jpg" repeatS TRUE repeatT TRUE } material Material {
ambientIntensity .668 shininess .2875 transparency 0 diffuseColor .7882
.6824 .5333 emissiveColor 0 0 0 specularColor .045 .045 .045 } } geometry
DEF libraslow_c7-FACES IndexedFaceSet { creaseAngle 3 coord DEF
libraslow_c7-COORD Coordinate { point [ coordenadas] } } }
              ]
            }
...
```

Figura 42: Especificação da hierarquia de articulações

Nesta especificação foi utilizada o nó *ImageTexture* para preenchimento do segmento hanim_c7 que define o tórax do humanóide; isso foi feito utilizando a imagem torax.jpg. O protótipo do avatar é visualizado na Figura 43.

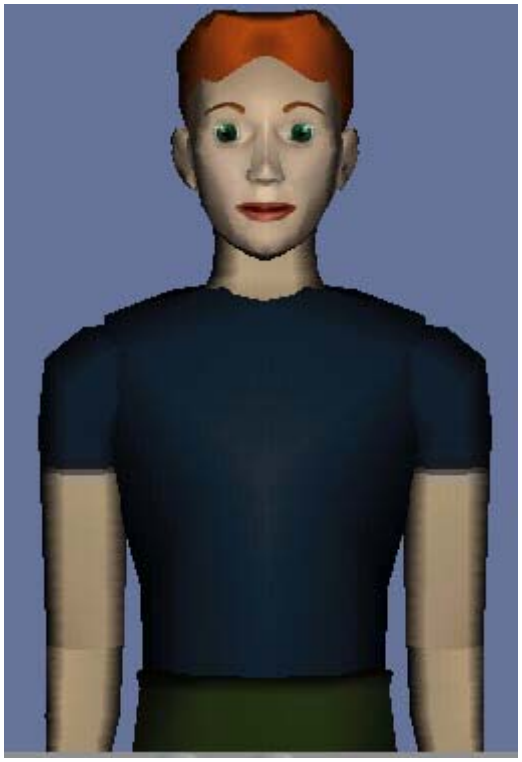

Figura 43: Protótipo H-Anim

6.6.6 Implementação das animações

Devido a se ter utilizado a especificação *H-Anim*, a implementação de animações para o avatar torna-se uma tarefa bastante simplificada. Uma vez definidas as articulações, animar o avatar é uma questão de efetuar rotações às articulações associadas às partes do corpo que se pretende mover. Por exemplo, se efetuar uma rotação no eixo X com um ângulo negativo à articulação do ombro do braço direito, o avatar irá levantar o braço direito esticado.

A Figura 44 mostra o código em Java que mostra a implementação da movimentação do cotovelo direito do avatar através do nó *hanim_r_elbow*.

```
0     import java.applet.*;
1     import vrml.external.field.*;
2     import vrml.external.Node;
3     import vrml.external.Browser;
4
5     public class Gera_Sinal extends Applet {
```

```
6
7       Browser browser;
8       Node cotoveloDir;
9       EventInSFRotation rotacao;
10
11      public void init () {
12        browser = Browser.getBrowser(this);
13        cotoveloDir = browser.getNode("hanim_r_elbow");
14        rotacao = (EventInSFRotation) cotoveloDir.getEventIn("rotation");
15        float[] val = new float[4];
16        val[0] = 30;
17        val[1] = 0;
18        val[2] = 20;
19        val[3] = 10;
20
21        rotacao.setValue(val);
22      }
23    }
```

Figura 44: Rotina de movimentação do avatar

Primeiramente são carregadas as classes Java para manipulação do *browser*, dos nós e dos campos dos grafo de cena; nas linhas 7 e 8 são definidas a variável para o acesso ao conteúdo do *browser* e outra para manipulação de um nó do grafo de cena. Na linha 9 é definida uma variável para manipulação do campo *Rotation* do nó desejado. A linha 12 instancia a variável de manipulação do *browser* e a linha 13 instancia a variável de manipulação do nó *hanim_r_elbow*, que define a articulação do cotovelo direito do humanóide. Na linha 14 é instanciada a variável que manipula o campo de rotação através do *field rotation*. E, finalmente, na linha 21 é definida uma nova configuração para o campo *rotation* do segmento *hanim_r_elbow* através do método *setValue* do campo *EventInSFRotation*.

A Figura 45 mostra o avatar depois da animação ser executada.

Figura 45: Avatar animado

A composição da movimentação dos sinais LIBRAS é baseada nesta arquitetura do EAI, configurando as rotações das articulações através de um aplicativo Java externo.

6.7 Desenvolvimento da Arquitetura de Dados Persistentes

O sucesso de qualquer aplicação XML dependerá do modo como é projetado os documentos XML; não apenas precisam ser capazes de carregar as informações que se desejam comunicar atualmente, mas precisam ser flexíveis o suficiente para acomodar também os requisitos futuros.

A primeira regra de modelagem de informações é concentrar-se no “mundo real”, e não na tecnologia. Um modelo de informação é uma descrição da informação usada em uma organização, independente de qualquer sistema de informática.

Por que a modelagem de informação é importante? Sem um modelo não há informações, apenas dados. O modelo de informação define o significado dos dados. Na verdade, sempre haverá um modelo de informação: a única escolha é ter um modelo de informação compartilhado que seja registrado e acordado, ou assumir o risco de qualquer um

ter um modelo de informação diferente em mente, com as confusões invevitáveis resultantes (ANDERSON, 2001).

Neste trabalho, é proposto um modelo de informação (vocabulário X-LIBRAS) para que desenvolvedores que desejam implementar futuras aplicações utilizando a LIBRAS, possam utilizar esse modelo padrão XML como ponto de partida e aplicações diversas possam se comunicar utilizando esse protocolo.

6.7.1 Projeto do Modelo Estático

Os modelos estáticos se concentram nas descrições dos estados admissíveis do sistema. Descrevem o tipo do objeto no sistema, suas propriedades e seus relacionamentos. Além de descrevê-los, logicamente, também definem nomes acordados para estes objetos. Por exemplo, na aplicação que deseja se implementar, nomes como sinal, forma da mão, movimento.

Concordar quanto aos nomes para as coisas é metade do trabalho, e é por isto que os modelos de informações XML são algumas vezes chamados de vocabulários, por exemplo; um dos objetivos deste trabalho é gerar um vocabulário XML chamado X-LIBRAS, que dite as regras para o modelo de documentos XML contendo informações da LIBRAS.

Os modelos dinâmicos se concentram em descrever o que acontece à informação: exemplos de tais modelos são diagramas de fluxo de processo, modelos de fluxos de dados e histórico da vida do objeto. Os modelos dinâmicos descrevem um conjunto de troca de informações: dados que são enviados de um lugar para outro para um determinado objetivo. Por exemplo, num sistema de comunicação de sinais, quando um usuário envia um sinal (mensagem LIBRAS) para outro usuário.

Em geral, os modelos estáticos são imediatamente relevantes ao projeto de um banco de dados, onde as informações são armazenadas por um longo período de tempo para servir uma variedade de propósitos; enquanto modelos dinâmicos são mais diretamente relevantes aos projetistas das mensagens, que têm apenas existência transitória e um propósito bem específico.

Este trabalho se concentra no desenvolvimento de um modelo de informação estática que sirva de base para o desenvolvimento de aplicações que necessitem de informações de sinais LIBRAS; o modelo dinâmico fica na dependência da implementação dessas aplicações.

Neste tópico, é analisada a estrutura de armazenamento dos sinais cujo conteúdo é utilizado pelo *parser* de validação e o módulo de mapeamento para gerar os sinais 3D através do H-Anim.

Para tanto foram utilizadas algumas etapas no processo de especificação do sinal no formato XML; tais etapas são citadas na Tabela 8.

Etapas
Mapeamento do Sinal em um *Schema* XML
Definição do Vocabulário XML para a representação da LIBRAS
Implementação do X-LIBRAS – Uma Linguagem de Marcação para LIBRAS
Representação de Sinais em Documentos XML
Transposição de um Documento XML do Vocabulário X-LIBRAS para um Documento X3D/VRML no Perfil H-Anim

Tabela 8: Etapas do processo de implementação do X-LIBRAS

6.7.2 Mapeamento do sinal em um *Schema* XML

A partir das unidades mínimas distintivas que formam um sinal da LIBRAS (configuração da mão, ponto de articulação, movimento e expressão facial), foi gerado um modelo de dados utilizando o UML, através de um diagrama de classes, demonstrando o relacionamento entre os sinais, suas unidades e as características dessas unidades. A Figura 46 mostra o diagrama de classes do sinal especificado.

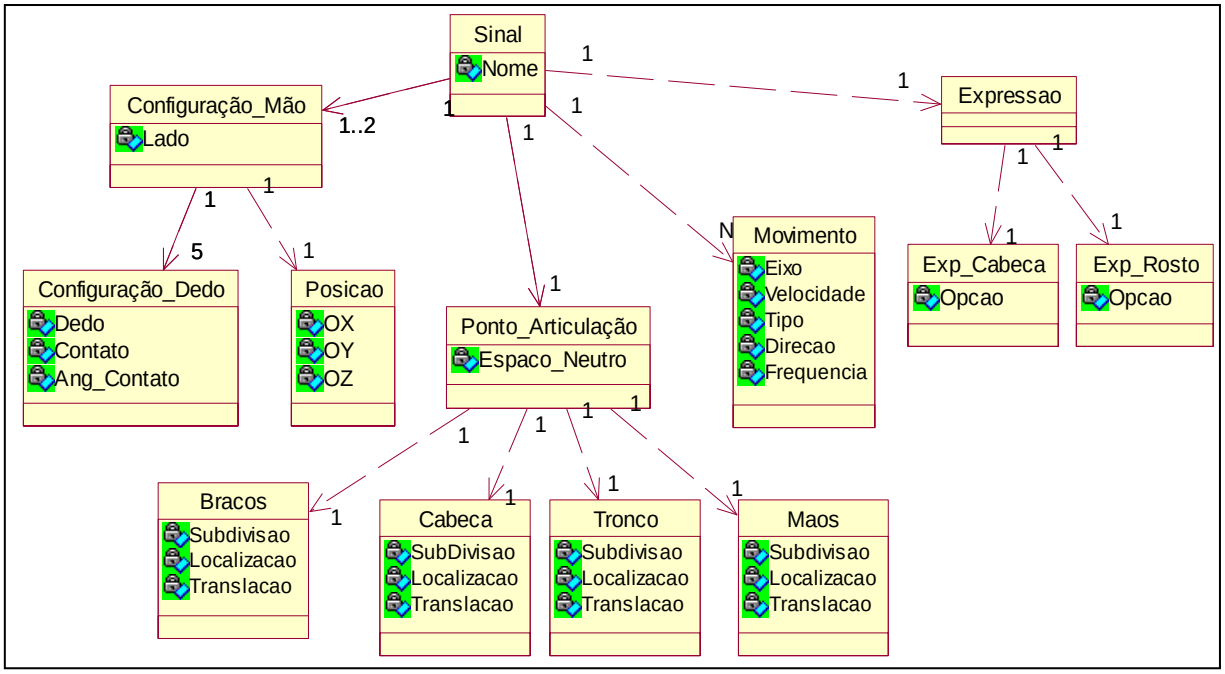

Figura 46: Diagrama de Classes da aplicação

6.7.3 Definição do vocabulário XML para a representação da LIBRAS

Na definição do vocabulário XML, é necessário definir o elemento *root* dos documentos XML; a classe Sinal foi definida como o elemento *root* para cada documento XML do vocabulário e as outras classes (Configuração_Mão, Ponto_Articulação, Movimento e Expressão) como elementos encadeados no elemento Sinal.

6.7.4 Implementação do X-LIBRAS – Uma Linguagem de Marcação para LIBRAS

O desenvolvimento do vocabulário XML foi baseado em (BRITO, 2003) que divide o sinal em 4 unidades mínimas distintivas:

- Configuração das Mãos
- Ponto de Articulação
- Movimento
- Expressão

A partir destas unidades, foram criados elementos para cada uma delas e atributos ou elementos para a configuração destas unidades.

A Figura 47 mostra o DTD X-LIBRAS gerado a partir do mapeamento dos sinas LIBRAS para o vocabulário XML. Para validar o projeto proposto, foi utilizado o DTD X-LIBRAS; porém, também foi projetado e especificado o *Schema* X-LIBRAS para implementações em aplicações que queiram utilizar este tipo de validação XML; este *Schema* está no Anexo B *Schema* X-LIBRAS.

```
<!ELEMENT Sinal (Config_Mao+, Ponto_Articulacao*, Movimento*, Expressao*)>
<!ATTLIST Sinal Nome CDATA #REQUIRED>
<!ELEMENT Config_Mao (Posicao, Config_Dedo*)>
<!ATTLIST Config_Mao Lado (Direito | Esquerdo) #REQUIRED>
<!ELEMENT Posicao EMPTY>
<!ATTLIST Posicao OX CDATA #REQUIRED>
<!ATTLIST Posicao OY CDATA #REQUIRED>
<!ATTLIST Posicao OZ CDATA #REQUIRED>
<!ELEMENT Config_Dedo EMPTY>
<!ATTLIST Config_Dedo Dedo (1 | 2 | 3 | 4 | 5) #REQUIRED>
<!ATTLIST Config_Dedo Contato (Sim | Nao)    #REQUIRED>
<!ATTLIST Config_Dedo Ang_Contracao (0 | 45 | 90) #REQUIRED>
<!ELEMENT Ponto_Articulacao (Cabeca | Tronco | Bracos | Mao)>
<!ATTLIST Ponto_Articulacao Espaco_Neutro (Sim | Nao) #REQUIRED>
<!ELEMENT Cabeca EMPTY>
<!ATTLIST Cabeca Subdivisao (Topo | Testa | Rosto | Rosto_Sup | Rosto_Inf |
Orelha | Olhos | Nariz | Boca | Bochecha | Queixo) #REQUIRED>
<!ATTLIST Cabeca Localizacao (Lado_Direito | Lado_Esquerdo | Medial | Interna |
Externa) #REQUIRED>
<!ATTLIST Cabeca Translacao (Lateral | Frente | Atras) #REQUIRED>
<!ELEMENT Tronco EMPTY>
```

```
<!ATTLIST Tronco Subdivisao (Pescoco | Ombro | Busto | Estomago | Cintura)
#REQUIRED>
<!ATTLIST Tronco Localizacao (Lado_Direito | Lado_Esquerdo | Medial | Interna |
Externa) #REQUIRED>
<!ATTLIST Tronco Translacao (Lateral | Frente | Atras) #IMPLIED>
<!ATTLIST Tronco Posicao (Proximo | Distancia_Media | Distante | Contato |
Contato_Inicial | Contato_Medial | Contato_Final) #IMPLIED>
<!ELEMENT Bracos EMPTY>
<!ATTLIST Bracos Subdivisao (Braco | Queixo | Antebraco | Cotovelo | Pulso)
#REQUIRED>
<!ATTLIST Bracos Localizacao (Lado_Direito | Lado_Esquerdo | Medial | Interna |
Externa) #REQUIRED>
<!ATTLIST Bracos Translacao (Lateral | Frente | Atras) #IMPLIED>
<!ELEMENT Mao EMPTY>
<!ATTLIST Mao Subdivisao (Palma | Costa | Lado_Indicador | Lado_Minimo | Dedos |
Ponta_Dedos | Juncao_Mao | Junta_Dedos | Dedo_Minimo | Anular |
Dedo_Medio | Indicador | Polegar | Intersticio_Polegar |
Intersticio_Indicador | Intersticio_Medio | Intersticio_Anular) #REQUIRED>
<!ATTLIST Mao Localizacao (Lado_Direito | Lado_Esquerdo | Medial | Interna |
Externa) #REQUIRED>
<!ATTLIST Mao Translacao (Lateral | Frente | Atras) #IMPLIED>
<!ELEMENT Movimento EMPTY>
<!ATTLIST Movimento Eixo CDATA #IMPLIED>
<!ATTLIST Movimento Velocidade (Tensao | Retencao | Continuidade | Refreamento |
Repetido) #IMPLIED>
<!ATTLIST Movimento Tipo (Retilineo | Circular | Continuo) #IMPLIED>
<!ATTLIST Movimento Direcao (Unidirecional | Bidirecional | Multidirecional)
#IMPLIED>
<!ATTLIST Movimento Frequencia (Simples | Repetido) #IMPLIED>
<!ELEMENT Expressao (Exp_Cabeca | Exp_Rosto* | (Exp_Cabeca, Exp_Rosto*))>
<!ELEMENT Exp_Cabeca EMPTY>
<!ATTLIST Exp_Cabeca Opcao (Frente_Tras | Lados | Frente | Lado | Tras) #IMPLIED>
<!ELEMENT Exp_Rosto EMPTY>
<!ATTLIST Exp_Rosto Opcao (Sobrancelha_Franzida | Olhos_Arregalados | Lance_Olhos |
Sobrancelha_Levantada | Bochecha_Inflada |
Bochecha_Contraida | Labios_Contraidos | Lingua |
Bochecha_Direita_Inflada | Contracao_Labio | Franzir) #IMPLIED>
```

Figura 47: DTD X-LIBRAS

Foi utilizado o *software* XMLwriter[1] para criação e definição do DTD e do *Schema* XML; a Figura 48 mostra o DTD X-LIBRAS no XMLwriter.

[1] http://xmlwriter.net

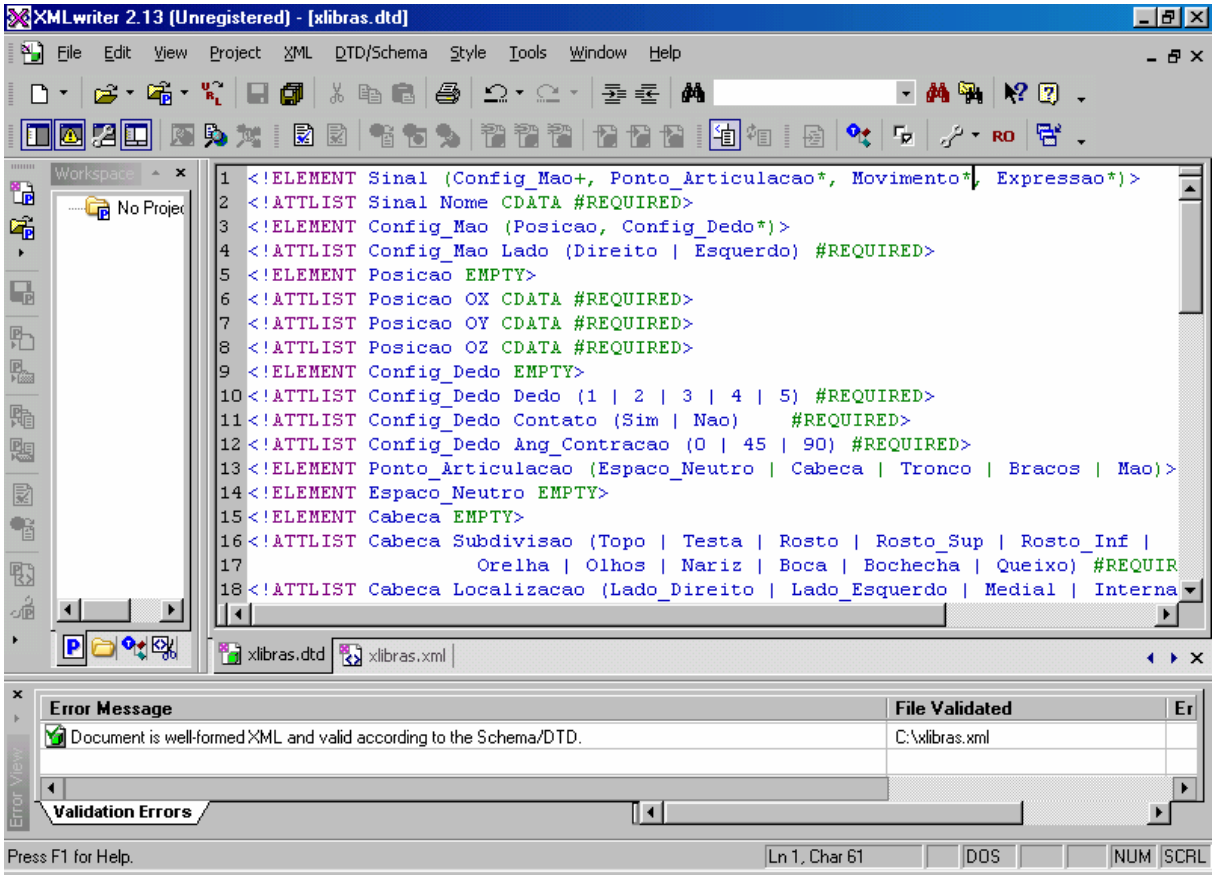

Figura 48: Ferramenta XMLwriter

6.7.5 Transposição do vocabulário X-LIBRAS para o avatar H-Anim

A configuração da mão é definida pela identificação da mesma através do atributo *Lado*, sua posição que define a orientação da palma da mão cujos atributos OX, OY e OZ definem o ângulo de rotação do pulso, ou seja, o *joint hanim_r_wrist*.

Ainda na configuração da mão o atributo Config_Dedo determina a rotação dos dedos, na qual o atributo Ang_Contracao determina o ângulo, a Tabela 9 mostra a relação da identificação do dedo com seu respectivo *joint*.

Dedo1	hanim_r_middle1
Dedo2	hanim_r_ring1
Dedo3	hanim_r_pinky1
Dedo4	hanim_r_index1
Dedo5	hanim_r_thumb1

Tabela 9: Os dedos e seus *joints* no H-Anim

No ponto de articulação, foram definidas as possíveis posições como elementos; a subdivisão, localização, translação e a posição como atributos desses elementos. Por exemplo, no sinal "soldado" (Figura 49), o ponto de articulação se dá na posição: cabeça, subdivisão: testa, localização: lado direito e translação: lateral.

Figura 49: Sinal "soldado" no X-LIBRAS

O elemento Ponto_Articulacao, do documento XML, para gerar este sinal, ficaria como mostra a Figura 50.

```
<Ponto_Articulacao>
        <Cabeca Subdivisao="Testa" Localizacao="Lado_Direito"
                Translacao="Lateral">
        </Cabeca>
</Ponto_Articulacao>
```

Figura 50: Atributo Ponto_Articulacao

Para se chegar a essa articulação, é necessária a rotação do ombro e do cotovelo direito e esquerdo através dos *joints hanim_r_shoulder, hanim_r_elbow*, *hanim_l_shoulde* e *hanim_l_elbow* respectivamente.

Foram utilizadas as classes Java XLibrasRotation e XLibrasVec3f (Anexo C) que de posse das rotações dos eixos X, Y, Z, retorna o valor em ângulos da rotação a ser efetuada.

Para gerar o sinal da Figura 49, o *joint hanim_r_shoulder* foi rotacionado -90 graus no eixo X e -15 graus no eixo Y, o *joint hanim_r_elbow* foi rotacionado -120 graus no eixo X e 28 no eixo Y.

O módulo Gerador de Sinal utiliza essa associação para cada ponto de articulação como mostrado na Tabela 10.

Posicao	**SubDivisao**	**Localizacao**	**Translacao**	**hanim_r_shoulder**			**hanim_r_elbow**		
				X	**Y**	**Z**	**X**	**Y**	**Z**
Cabeca	Testa	Lado_Direito	Lateral	-90	-15	0	-120	28	0

Tabela 10: Parâmetros para definição do ponto de articulação

O movimento do sinal é definido em segmentos do movimento, ou seja, um sinal pode ter mais de um movimento, e este é caracterizado pelos atributos: eixo, velocidade, tipo, direção e freqüência, cada um destes com seus possíveis valores. De igual modo à configuração da articulação, os movimentos são configurados no módulo Gerador de Sinal, através da articulação dos *joints hanim_r_shoulde, hanim_r_elbow*, *hanim_l_shoulde* e *hanim_l_elbow*, mais o tipo do movimento, cada um destes tipos com seu algoritmo próprio.

A expressão facial se dá através de uma configuração para a cabeça ou do rosto ou dos dois, por isso, o elemento “Expressao” foi definido assim:

```
<!ELEMENT Expressao (Exp_Cabeca | Exp_Rosto* | (Exp_Cabeca, Exp_Rosto*))>
```

A expressão da cabeça contém o atributo “opcao" que define a posição da cabeça; para cada opção é definida uma rotação para o *joints hanim_skullbase* (base do crânio). O módulo Gerador de Sinal utiliza a Tabela 11 para essa configuração.

Exp_Cabeca	hanim_skullbase		
	X	Y	Z
Lados			
Frente/Trás			
Inclinação			

Tabela 11: Parâmetros para definição da expressão da cabeça

A Figura 51 mostra o avatar resultante de um sinal que utiliza a expressão da cabeça para representá-lo.

Figura 51: Sinal utilizando a expressão da cabeça

A expressão facial utiliza-se de opções pré-definidas conhecidas atualmente na língua de sinais. Para cada uma delas foi criado um valor para o atributo “opcao” do elemento “Exp_Rosto”; para cada uma dessas expressões, foi modelada uma textura como mostra a Figura 52.

Expressão Normal	Sobrancelhas Levantadas	Sobrancelhas Franzidas
		

Figura 52: Expressões faciais

Na Figura 53 é mostrada a representação do sinal “soldado” num documento XML do vocabulário X-LIBRAS.

```
<?xml version="1.0"?>
<!DOCTYPE Sinal SYSTEM "xlibras.dtd">
<Sinal Nome="Soldado">
  <Config_Mao Lado="Direito">
    <Posicao OX="0" OY="0" OZ="0"></Posicao>
    <Config_Dedo Contato="Nao" Dedo="1" Ang_Contracao="0"></Config_Dedo>
       <Config_Dedo Contato="Nao" Dedo="2"
Ang_Contracao="0"></Config_Dedo>
       <Config_Dedo Contato="Nao" Dedo="3"
Ang_Contracao="0"></Config_Dedo>
       <Config_Dedo Contato="Nao" Dedo="4"
Ang_Contracao="0"></Config_Dedo>
       <Config_Dedo Contato="Nao" Dedo="5"
Ang_Contracao="0"></Config_Dedo>
  </Config_Mao>
  <Ponto_Articulacao>
    <Cabeca Subdivisao="Testa" Localizacao="Lado_Direito"
            Translacao="Lateral"></Cabeca>
  </Ponto_Articulacao>
</Sinal>
```

Figura 53: Documento XML do vocabulário X-LIBRAS

Neste documento há a definição de todos os elementos que formam o sinal, isto é, as unidades mínimas distintivas são descriminadas de modo que outras aplicações possam interpretar esse sinal no sistema específico, como por exemplo o *parser* da aplicação proposta, que utilizará esse documento para gerar o sinal no avatar H-Anim.

A Figura 54 mostra a aplicação sendo executada no *browser* VRML demonstrando o mapeamento do sinal do vocabulário X-LIBRAS para a especificação H-Anim, permitindo ao usuário a visualização dos sinais da LIBRAS num ambiente virtual.

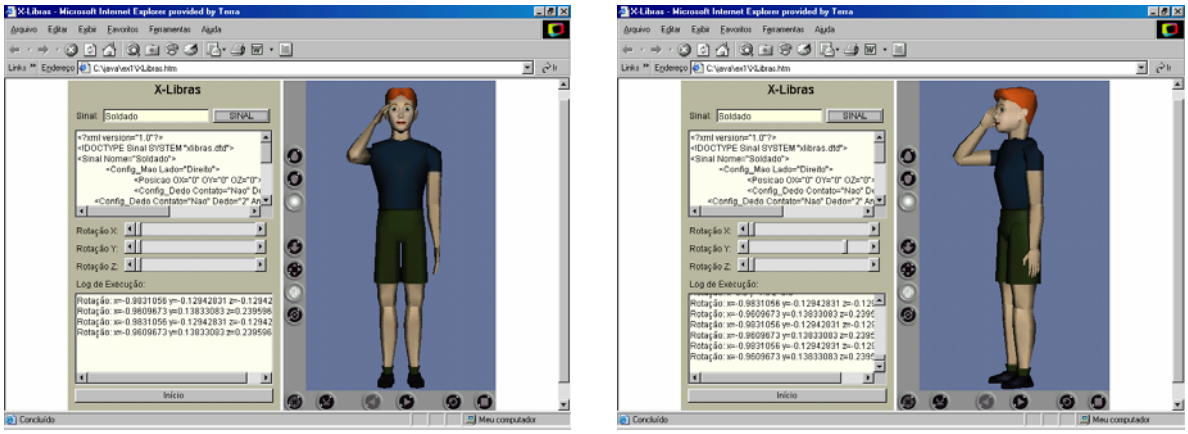

Figura 54: Ambiente Virtual X-LIBRAS

Além da visualização, o usuário pode explorar com mais detalhes a formação do sinal através da rotação do avatar humanóide, podendo assim, ter uma visão tridimensional do sinal.

6.8 Implementação no X3D

Para demonstrar a implementação do sistema proposto no X3D foi especificado o mesmo avatar humanóide, visto anteriormente, utilizando a especificação X3D. A implementação desse avatar utilizou as mesmas características como visto na Figura 55 e foi desenvolvido no *software* de autoria Vizx3D.

O Vizx3D é uma ferramenta que permite o desenvolvimento de ambientes virtuais através de uma interface visual (GUI), além de possibilitar a simulação da visualização do *browser*.

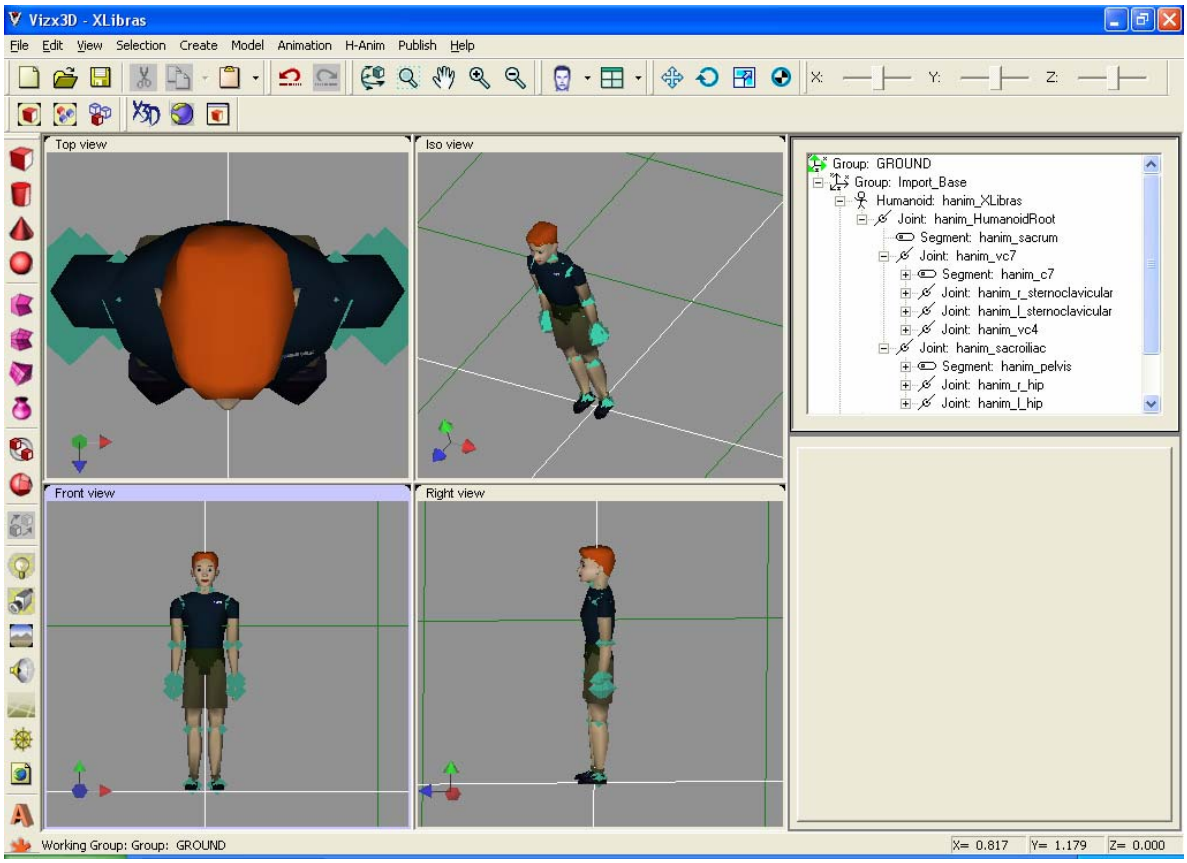

Figura 55: Implementação do X-LIBRAS no Vizx3D

Na Figura 56 pode ser visualizado o código X3D da implementação do avatar humanóide, nesse código está a implementação do avatar até o elemento *hanim_vc7*.

```
<?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='Full'>
<Scene>
<Humanoid DEF='hanim_XLibras' name="XLibras" version="1.1"
     USE='hanim_sacrum'
     USE='hanim_c7'
     USE='hanim_r_clavicle'
     USE='hanim_r_upperarm'
     USE='hanim_r_forearm'
     USE='hanim_r_hand'
     USE='hanim_r_middle_proximal'
     USE='hanim_r_ring_proximal'
     USE='hanim_r_pinky_proximal'
     USE='hanim_r_index_proximal'
     USE='hanim_r_thumb_metacarpal'
     USE='hanim_l_clavicle'
     USE='hanim_l_upperarm'
     USE='hanim_l_forearm'
     USE='hanim_l_hand'
     USE='hanim_l_thumb_metacarpal'
     USE='hanim_l_ring_proximal'
     USE='hanim_l_pinky_proximal'
     USE='hanim_l_index_proximal'
     USE='hanim_l_middle_proximal'
```

```
      USE='hanim_c4'
      USE='hanim_c1'
      USE='hanim_skull'
      USE='hanim_pelvis'
      USE='hanim_r_thigh'
      USE='hanim_r_calf'
      USE='hanim_r_hindfoot'
      USE='hanim_l_thigh'
      USE='hanim_l_calf'
      USE='hanim_l_hindfoot'
      USE='hanim_HumanoidRoot'
      USE='hanim_vc7'
      USE='hanim_r_sternoclavicular'
      USE='hanim_r_shoulder'
      USE='hanim_r_elbow'
      USE='hanim_r_wrist'
      USE='hanim_r_middle1'
      USE='hanim_r_ring1'
      USE='hanim_r_pinky1'
      USE='hanim_r_index1'
      USE='hanim_r_thumb1'
      USE='hanim_l_sternoclavicular'
      USE='hanim_l_shoulder'
      USE='hanim_l_elbow'
      USE='hanim_l_wrist'
      USE='hanim_l_thumb1'
      USE='hanim_l_ring1'
      USE='hanim_l_pinky1'
      USE='hanim_l_index1'
      USE='hanim_l_middle1'
      USE='hanim_vc4'
      USE='hanim_vc1'
      USE='hanim_skullbase'
      USE='hanim_sacroiliac'
      USE='hanim_r_hip'
      USE='hanim_r_knee'
      USE='hanim_r_ankle'
      USE='hanim_l_hip'
      USE='hanim_l_knee'
      USE='hanim_l_ankle'>
      <Joint DEF='hanim_HumanoidRoot' center='0 .707 -.024' name="HumanoidRoot">
       <Group DEF='childof_hanim_HumanoidRoot'>
        <Transform DEF='dad_hanim_sacrum'>
         <Segment DEF='hanim_sacrum' name="sacrum"/>
        </Transform>
        <Joint DEF='hanim_vc7' center='-.00066 .79796 -.00199' name="vc7">
         <Group DEF='childof_hanim_vc7'>
          <Transform DEF='dad_hanim_c7'>
           <Segment DEF='hanim_c7' name="c7">
            <Transform DEF='dad_IndexedFaceSet1'>
             <Shape DEF='IndexedFaceSet1'>
              <Appearance>
               <ImageTexture url='"textures/torax.jpg"'/>
...
```

Figura 56: Implementação do avatar em X3D

Para a implementação do movimento dos sinais no X3D, foram utilizados as articulações mostradas na Tabela 12.

Joint	**Articulação**
hanim_vc4	base do pescoço (movimento da cabeça)
hanim_r_shoulder / hanim_l_shoulder	ombro direito e esquerdo
hanim_r_elbow / hanim_l_elbow	cotovelo direito e esquerdo
hanim_r_wrist / hanim_l_wrist	pulso direito e esquerdo
hanim_r_middle1 / hanim_r_middle1 hanim_r_ring1 / hanim_l_ring1 hanim_r_pinky1 / hanim_l_pinky1 hanim_r_index1 / hanim_l_index1 hanim_r_thumb1 / hanim_l_thumb1	dedos das mãos esquerda e direita

Tabela 12: *Joints* utilizados nos sinais LIBRAS

A implementação dos movimentos do avatar foi feita utilizando recursos do X3D através do elemento *ROUTE*, definindo o nó (*fromnode*) e o campo (*fromfield*) da posição origem, e o nó (*tonode*) e o campo (*tofield*) da posição destino.

Na ferramenta Vizx3D há um editor de implementação de movimentos por *frames*, isto é, a cada intervalo de tempo pode definir-se a posição das articulações e, então, as posições são memorizadas até o movimento finalizar. A Figura 57 mostra o editor de animações do Vizx3D.

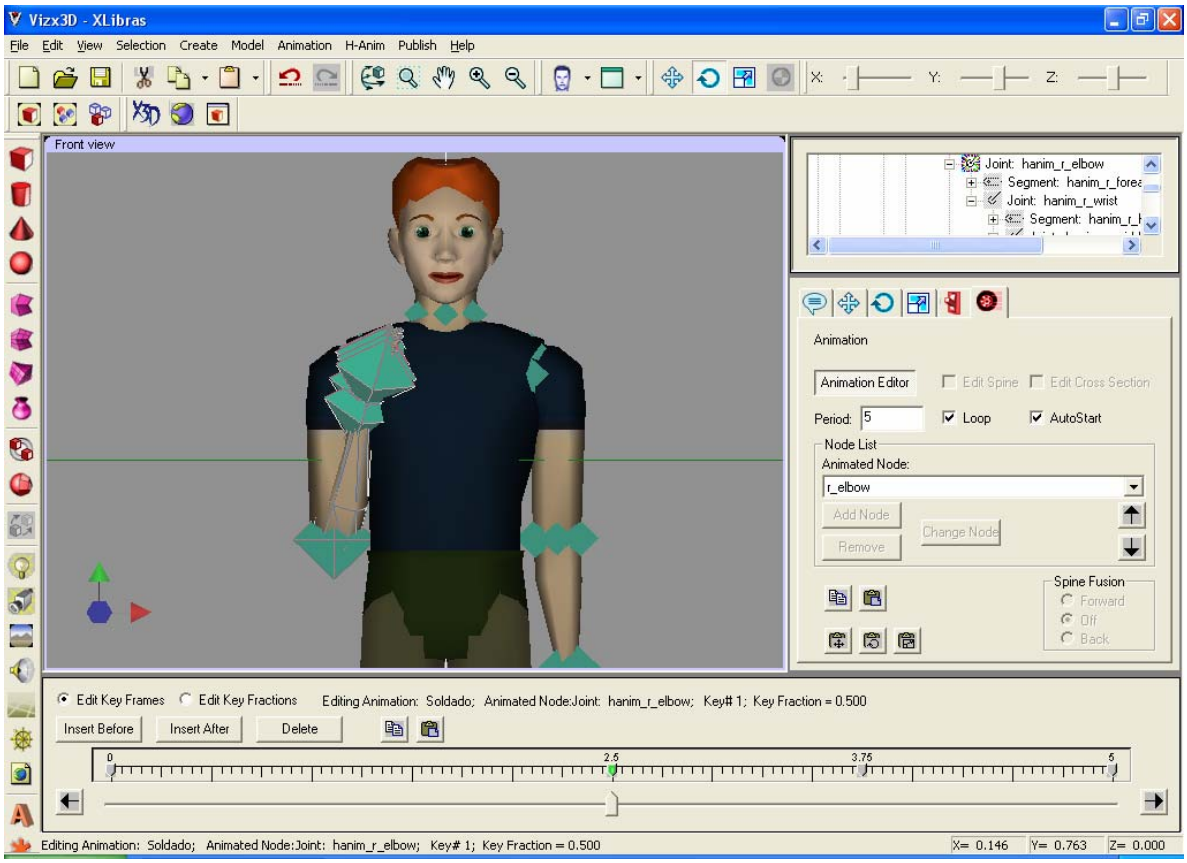

Figura 57: Editor de animação do Vizx3D

O código da implementação do movimento pode ser visto na Figura 58.

```
<TimeSensor DEF='Soldado'
 cycleInterval='5.000'
 loop='false'
 startTime='-1.000'
/>
<OrientationInterpolator DEF='Soldado_rot0' key='0 .5 .75 1'
keyValue='-.952 .133 .275 0 -.989 .15 0 2.752 -.649 .068 .757 1.378 -.989 .151 0
1.941'
/>
<OrientationInterpolator DEF='Soldado_rot1'
 key='0 .5 .75 1'
 keyValue='-.997 -.083 0 0 -.997 -.083 0 0 -.962 -.273 0 1.589  -.994 .11 0 1.276'
/>
<TimeSensor DEF='vizx_init'
 cycleInterval='0.100'
 loop='false'
/>
<ROUTE fromNode='vizx_init' fromField='cycleTime' toNode='Soldado'
toField='startTime'/>
<ROUTE fromNode='vizx_init' fromField='cycleTime' toNode='vizx_init'
       toField='stopTime'/>
<ROUTE fromNode='Soldado' fromField='fraction_changed' toNode='Soldado_rot0'
       toField='set_fraction'/>
<ROUTE fromNode='Soldado_rot0' fromField='value_changed' toNode='hanim_r_elbow'
       toField='set_rotation'/>
<ROUTE fromNode='Soldado'fromField='fraction_changed' toNode='Soldado_rot1'
       toField='set_fraction'/>
<ROUTE fromNode='Soldado_rot1' fromField='value_changed' toNode='hanim_r_shoulder'
       toField='set_rotation'/>
```

Figura 58: Código X3D da implementação da animação

A Figura 59 mostra a seqüência do sinal implementado no X3D. Esta visualização foi feita utilizando a ferramenta de simulação do Vizx3D.

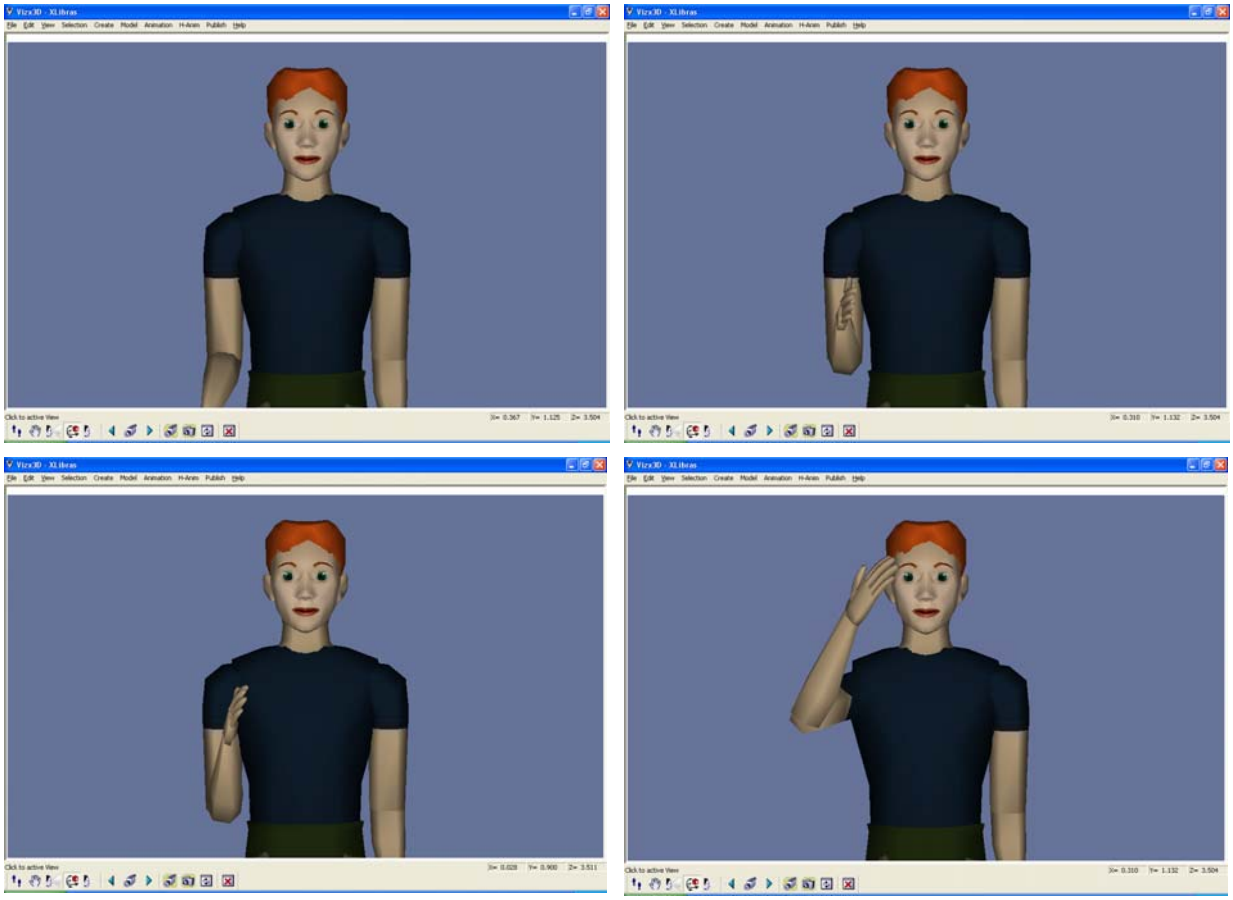

Figura 59: Animação do avatar X3D no Vizx3D

CONCLUSÃO

Este trabalho apresentou uma tecnologia que permite a criação de um mecanismo padronizado de armazenamento e intercâmbio de informações dos sinais da LIBRAS por meio do vocabulário X-LIBRAS, facilitando assim, o desenvolvimento de novas ferramentas e aplicações computacionais na área da comunicação gestual-facial

Vocabulários XML são utilizados nas mais diversas áreas do conhecimento para padronizar a troca de informações de um determinado segmento da ciência e o X-LIBRAS vai de encontro a essa tendência pois fornece meios de armazenar as informações necessárias para representar o sinal da LIBRAS.

Devido a recente adoção da LIBRAS como meio legal de comunicação, e ainda não haver uma padronização das unidades mínimas distintivas que formam o sinal, esse vocabulário deve ir se adaptando às mudanças naturais que ocorrem na língua, porém, esse é um recurso facilitador que a tecnologia XML fornece, pois sua natureza auto-descritiva é flexível a alterações necessárias dos vocabulários, diferentemente de outras formas de armazenamento de informações.

Um dos objetivos do desenvolvimento do vocabulário X-LIBRAS era de se poder construir um ambiente virtual onde pesquisadores, usuários deficientes e qualquer pessoa que desejasse conhecer melhor a natureza dos sinais da LIBRAS, pudesse encontrar nesse sistema um meio amigável e eficiente de interface homem-computador.

O ambiente virtual implementado atendeu aos principais aspectos de um sistema de realidade virtual, isto é, sintético, ou seja, gerado em tempo real por um computador; realismo, o avatar atendeu à necessidade que se tinha que ao realizar o sinal, pudesse-se compreender e identificar perfeitamente o sinal como se uma pessoa real estivesse realizando-

o. Isto foi possível devido à tecnologia pesquisada e utilizada, o padrão H-Anim e a implementação das rotinas de animação no avatar.

Nessa questão, ficou prejudicada a implementação do sistema em sua totalidade na tecnologia X3D, pois até a finalização do projeto ainda não haviam *plug-ins* que fossem compatíveis com todos os recursos necessários para execução do sistema, porém o *software* Vizx3D, cedido gratuitamente pelo seu fabricante (*Virtock Technologies*) para este projeto, mostrou-se como a melhor ferramenta de autoria para desenvolvimento de ambientes virtuais utilizando X3D. Ao entrarmos em contato com o seu fabricante, este pretende lançar um *plug-in* que contemple todos os recursos do H-Anim.

Devido a este fato, a geração dos sinais por meio de uma aplicação externa ao ambiente virtual teve que ser desenvolvida com o VRML, e no X3D foi implementado os sinais através de mecanismo internos da tecnologia. Porém, de acordo com o que foi especificado no projeto, ao existir um *plug-in* que esteja em acordo com os recursos do X3D, o sistema implementado também desempenharia o mesmo papel do que foi feito em VRML.

Outro aspecto que trouxe vantagens no desenvolvimento do projeto, foi a utilização de uma metodologia de engenharia de *software* voltada para ambientes virtuais, pois possibilitou o entendimento do processo de desenvolvimento de sistemas para realidade virtual. Esta metodologia também permitiu a divisão do sistema em várias fases, possibilitando assim, a documentação do desenvolvimento em cada uma das fases, facilitando futuras manutenções e reutilização dos módulos desenvolvidos.

Trabalhos Futuros

Utilizando este vocabulário, outras ferramentas podem ser implementadas na área da LIBRAS, como por exemplo um editor de sinais, no qual o usuário pudesse montar no próprio

ambiente virtual, os sinais através das unidades mínimas distintivas, visualizar o sinal criado e gravar na base XML do vocabulário X-LIBRAS, o novo sinal.

Outra implementação possível, é um ambiente de colaboração em rede, utilizando a Internet, já que todas as tecnologias utilizadas na implementação são voltadas para a Internet.

Além disso, o processo de desenvolvimento pode ser completado, criando uma etapa de homologação do sistema, na qual, usuários com deficiência auditiva pudessem opiniar sobre a aplicação e propor melhorias.

REFERÊNCIAS

AMES, Andrea L.; MORELAND, John L.; NADEAU, David R. **VRML 2.0 Sourcebook**. Nova York: John Wiley & Sons, 1997.

ANDERSON, Richard; et al. **Professional XML**. ed. Ciência Moderna, 2001.

APAYDIN, Ozan. **Networked Humanoid Animation Driven by Human Voice Using Extensible 3D (X3D), H-Anim and Java Speech Open Standards**. Dissertação de Mestrado – Naval Postgraduate School, Monterey, CA, 2002.

ASTHEIMER, Peter. **A Business View of Virtual Reality**. IEEE Computer Graphics and Applications, 1999. vol. 19, pp. 28-29.

AZUMA, Ronald T. **A Survey of Augmented Reality**. Presence, 1997. vol. 6, pp. 355-385.

BARRILLEAUX, Jon. **3D User Interfaces With Java 3D**. Greenwich: Manning, 2001

BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. **The Unified Modeling Language User Guide**. Tradução de Fábio Freitas da Silva. Rio de Janeiro: Campus, 2000.

BOWMAN, Douglas A. **Interaction Techniques For Common Tasks In Immersive Virtual Environments: Design, Evaluation, and Application**. Doctoral dissertation, Georgia Institute of Technology, 1999. Disponível em <http://vtopus.cs.vt.edu/~bowman/thesis>. Acesso em 18 fev 2004.

BRITO, Lucinda Ferreira. **A Língua Brasileira de Sinais**. 2003, Disponível em <http://www.ines.org.br/ines_livros/fasc7_principal.htm>. Acesso em 13 mai. 2003.

BRITO, Lucinda Ferreira. **Por uma gramática de língua de sinais**. Rio de Janeiro, Tempo Brasileiro: UFRJ, Departamento de lingüística e filosofia, 1995

BROWN, Barry, et al. **Lessons From The Lighthouse: Collaboration in A Shared Mixed Reality System**. Ft. Lauderdale, Florida, USA, 2003.

BRUTZMAN, Don. **X3D-Edit for Extensible 3D (X3D) Graphics**, 2003. Disponível em: <www.web3d.org/TaskGroups/x3d/translation/README.X3D-Edit.html>. Acesso em: 17 mar. 2003.

CAMPOS, Márcia de Borba, et. al. **SIGNSIM: uma ferramenta para auxílio à aprendizagem da língua brasileira de sinais**. In: V Congresso Ibero-Americano de informática na Educação – RIBIE – Chile, 2000.

CAMPOS, Márcia de Borba. **Ambiente Telemático de interação e comunicação para suporte à educação bilíngüe de surdos**. Tese apresentada ao Programa de Pós-Graduação em Informática na Educação da UFRGS. Porto Alegre, 2001.

CAMPOS, Márcia de Borba. **SIGNTALK: chat baseado na escrita de língua de sinais**. In: Congresso Iberoamericano de Informática Educativa Especial – CIEE'98. Neuguén, Argentina, 1998. Disponível em CD-ROM.

CAPOVILLA, Fernando C., et al. **Brazilian Sign Language Lexicography and Technology: Dictionary Digital Encyclopedia, Chereme-Based Sign Retrieval, and Quadriplegic Deaf Communication Systems**. Sign Language Studies, 2003. vol. 3, no. 4.

CAPOVILLA, Fernando C, et al. **Sistema de Multimídia para comunicação surdo-surdo e surdo-ouvinte em línguas brasileira e americana de sinais via rede de computador**. Revista Informática em saúde, 1996. Ano 20, Vol. 20, Nº. 03. p. 110 –114.

CASEY, Larijani C. **Realidad Virtual**. McGrawHill, 1994.

CLARK, James. **XSL Transformations (XSLT) Version 1.0**, 1999. Disponível em: <http://www.w3.org/TR/xslt>. Acesso em 12 mar. 2003.

FENCOTT, Clive. **Towards a Design Methodology for Virtual Environments**. Virtual Reality Applications Research Center. University of Teesside, 1999. Consultado em 23 jun. 2003. Disponível em <http://www.cs.york.ac.uk/hci/kings_manor_workshops/UCDIVE/fencott.pdf>.

FISHWICK, Paul. **Object-Oriented Design for Physical Modeling. To Appear in the ACM Transactions on Modeling and Computer Simulation**. 1996. Disponível em <http://www.cise.ufl.edu/fishwick>. Acesso em 12 fev. 2004.

FOLEY, James D., et al. **Computer Graphics, Principles and Practice**. ed. Reading, MA: Addision-Wesley, 1996.

GÓES, Maria Cecília Rafael de. **Linguagem, Surdez e Educação**. Editora Autores Associados. Campinas – São Paulo, 1996.

GOLDBERG, Rick. **Xj3D Architectural Overview**. 1999. Disponível em <http://www.web3d.org/TaskGroups/x3d/sun/xj3d-arch-over.html>. Acesso em 28 nov. 2002.

GRIEPP, Timothy P.; CRUZ-NEIRA, Carolina. **XJL – an XML Schema for the Rapid Development of Advanced Synthetics Environments**. 2002. Disponível em <http://citeseer.nj.nec.com/griepp02xjl.html>. Acesso em 03 fev. 2003.

HAREL, David, et al. **Statemate: a working environment for the development of complex reactive systems**. IEEE Transactions on Software Engineering, 1990. vol. 16, n. 4, pags.403-414.

HUANG, Zhisheng; ELIENS, Antons; VISSER, Cees. **A Platform for Embodied Conversation Agents Based on Distributed Logic Programming**, 2002. Disponível em <http://www.vhml.org/workshops/AAMAS/papers/eliens.pdf>. Acesso em 05 ago. 2003.

HUMANOID ANIMATION WORKING GROUP. **Specification for a Standard Humanoid**, 2002. Disponível em <http://www.h-anim.org/Specifications/H-Anim1.1>. Acesso em 12 mar. 2003.

ISO/IEC FCD 19774:200x, Humanoid animation (H-Anim). **International Standard is: Information technology - Computer graphics and image processing - Humanoid animation (H-Anim)**. Disponível em <http://www.h-anim.org/Specifications/H-Anim200x/ISO_IEC_FCD_19774>. Acesso em 05 jun. 2003.

ISO/IEC FCD 19777-2:200x. **Extensible 3D (X3D) language bindings Part 2: Java**. Disponível em <http://www.web3d.org/x3d/specifications/ISO-IEC-19777-FCD-X3dLanguageBindings/Part2>. Acesso em 28 mar. 2004.

ISO-IEC-19775-FDIS-X3DABSTRACTSPECIFICATION. **Extensible 3D (X3D) Part 1: Architecture and base components - 18 Texturing component**. Disponível em < http://www.web3d.org/x3d/specifications/ISO-IEC-19775-FDIS-X3dAbstractSpecification/Part01/components/texturing.html>. Acesso em 10 mai. 2004.

KALRA, Prem, et al. **Simulation of Facial Actions Based on Rational Free Form Deformations**. Proc. Eurographics'92, Cambridge, U.K., 1992.

KALRA, Prem, et al. **SMILE: A Multilayered Facial Animation System**. Proc. of Conference of Modeling in Computer Graphics, Springer, Tokyo, 1991.

KANG, Kyo C., KO, Kwang-Il, **PARTS: a temporal logic-based real-time software specification and verification method**. Proceedings of the 17th international conference on Software engineering, p.169-176, April 24-28, Seattle, Washington, United States, 1995

KARNOPP, Lodenir Becker. **Aquisição do parâmetro de configuração de mão na língua brasileira de sinais (LIBRAS): estudo sobre quatro crianças surdas, filhas de pais surdos.** Dissertação de mestrado em letras – Instituto de Letras e Artes, Pontifícia Universidade Católica do Rio Grande do Sul – PUCRS. Porto Alegre, 1994.

KIM, Jounghyun G., et al. **Software Engineering of Virtual Worlds**. Proceedings of the ACM symposium on Virtual reality software and technology. Taipei, Taiwan, 1998.

Kimen, Shel. **VRML Is- Java Does- And the EAI Can Help**. 1997. Disponível em <http://vrml.sgi.com/features/java/java.html>. Acesso em 12 fev. 2004.

KIRNER, Cláudio e PINHO, Márcio S. **Introdução à Realidade Virtual**. Minicurso JAI/SBC, Recife, PE, 1996.

KOLEVA, Boriana; BENFORD, Steve; GREENHALGH, Chris. **The Properties of Mixed Reality Boundaries**. In Proc. 6th European Conference on Computer Supported Co-operative Work (ECSCW'99), Copenhagen, 1999.

KULWINDER, Kaur. **Designing Virtual Environments for Usability**. PhD Thesis, City University, London, 1998.

MACEDO, Daniela Remião. ***Sign Dic*: Um ambiente multimídia para a criação e consulta de dicionários bilíngües de línguas de sinais e línguas orais**. Dissertação de mestrado - PUCRS. Porto Alegre, 1999a.

MARCATO, Simone A., et al. **Um Ambiente para a Aprendizagem da Língua de Sinais**. In: SBC 2000 – XX Congresso da Sociedade Brasileira de Computação, PUCPR - Curitiba, agosto de 2000.

MAZUTTI, Caroline et al**. SignHTML Editor HTML para escrita de Língua de Sinais**. Trabalho de conclusão - PUCRS. Porto Alegre, 2001.

MEGGINSON, David. **Simple API for XML**. 2002. Disponível em <http://www.saxproject.org>. Acesso em 19 mar. 2003.

MILGRAM, Paul; KISHINO, Fumio. **A Taxonomy of Mixed Reality Visual Display.** Inst. Of Electronics, Information and Communication Engineers (IEICE) Trans. Information and Systems, vol. E77-D, no. 12, 1994a.

MILGRAM, Paul, et al. **Augmented Reality: A Class of Display on the Reality-Virtuality Continuum**. In: Telemanipulador and Telepresence Technologies. SPIE vol. 2351, 1994b.

PAUSCH, Randy F. **The Software Development Environment**. **Implementing Virtual Reality**. SIGGRAPH 93 Course Nota 43. pages 1.6.1-1.6.6, 1993.

QUADROS, Ronice Müller. **Educação de Surdos – Aquisição da Linguagem**. Artes Médicas. 1ª Edição. Porto Alegre, 1997.

RAJAN, Vivek, et al. **A Realistic Video Avatar System for Networked Virtual Environments**. In: Immersive Projection Technology Conference, 2002.

REITMAYR, Gerhard; SCHMALSTIEG, Dieter. **An Open Software Architecture for Virtual Reality Interaction**, 2001. Disponível em <http://citeseer.nj.nec.com/reitmayr01opentracker.html>. Acesso em 21 nov 2002.

ROCHA, Heloísa Vieira et al. **Um Ambiente para a Aprendizagem da Língua de Sinais**. In: SBC 2000 – XX Congresso da Sociedade Brasileira de Computação, PUCPR - Curitiba, agosto de 2000.

RUMBAUGH, James, et al. **Object Oriented Modeling and Design**. Edglewood Cliffs, New Jersey: Prentice Hall, 1991.

SÁNCHEZ-SEGURA, Maria-Isabel, et al. **The Design and Implementation of a GUI-Based Interaction Authoring Tool for Virtual Environments**. SEKE'01. The 13th International Conference on Software Engineering and Knowledge Engineering. Buenos Aires, Argentina, 2001.

SANTAROSA, Lucila Maria. **Telemática y la inclusión virtual y social de personas con necesidades especiales: un espacio posible en la Internet**. in: V Congresso Ibero-Americano de Informática na Educação- RIBIE: Viñadelmar, 2000.

SHNEIDERMAN, Ben. **Designing the User Interface, Strategies for Effective Human-Computer Interaction**. Ed. Reading, MA: Addison-Wesley, 1998.

SIGNINGAVATAR. Disponível em <http://www.vcom3d.com>. Acesso em 10 set. 2003.

SUTTON, Valerie. ***SignWriting* Site**. Disponível em <http://www.signwriting.org>. Acesso em 12 mar. 2002.

TAMURA, Hideyuki; YAMAMOTO, Hiroyuki; KATAYAMA, Akihiro. **Mixed Reality: Future Dreams Seen at the Border between Real and Virtual Worlds**. In: IEEE. Novembr/December, 2001.

TANRIVERDI, Vildan e JACOB, Robert J. K. **VRID: A Design Model and Methodology for Developing Virtual Reality Interfaces**. Department of Electrical Engineering and Computer Science. Tufts University. Medford, MA, 2001.

THALMANN, Daniel et al. **Avatars in Networked Virtual Environments.** ed. Wiley, 1999.

W3C CONSORTIUM. **Extensible Markup Language (XML)**, 2003. Disponível em: <http://www.w3.org/XML>. Acesso em 14 jan. 2003.

W3C DOM WORKING GROUP. **Document Object Model (DOM)**, 2002. Disponível em: <http://www.w3.org/DOM>. Acesso em 17 mar. 2003.

WANG, F. **Using Live Vídeo to Represent Remote Users in Collaborative Virtual Environment**, University of Illinois at Chicago, 1998.

WEB3D CONSORTIUM. **Extensible 3D (X3D™) Graphics Working Group**, 2002. Disponível em <http://www.web3d.org/x3d>. Acesso em 03 mar. 2002.

WITHERS, John. **Developing Java Entertainment Applets**, New York: John Wiley & Sons, 1997.

YAMAMOTO, H. **Case Studies of Producing Mixed Reality Worlds**. In: Proceedings IEEE, 1999.

YURA, Shunsuke; USAKA, Tomonori; SAKAMURA, Ken. **Video Avatar: Embedded Vídeo for Collaborative Virtual Environment**. In: IEEE International Conference on Multimedia Computing and Systems Volume II, 7 – 11 June, Florence, Italy, 1999.

ANEXO A - LEI Nº 10.436, DE 24 DE ABRIL DE 2002

Dispõe sobre a Língua Brasileira de Sinais – LIBRAS e dá outras providências.

O PRESIDENTE DA REPÚBLICA

Faço saber que o Congresso Nacional decreta e eu sanciono a seguinte Lei:

Art. 1º É reconhecida como meio legal de comunicação e expressão a Língua Brasileira de Sinais – LIBRAS e outros recursos de expressão a ela associados.

Parágrafo único. Entende-se como Língua Brasileira de Sinais – LIBRAS a forma de comunicação e expressão, em que o sistema lingüístico de natureza visual-motora, com estrutura gramatical própria, constituem um sistema lingüístico de transmissão de idéias e fatos, oriundos de comunidades de pessoas surdas do Brasil.

Art. 2º Deve ser garantido, por parte do poder público em geral e empresas concessionárias de serviços públicos, formas institucionalizadas de apoiar o uso e difusão da Língua Brasileira de Sinais – LIBRAS como meio de comunicação objetiva e de utilização corrente das comunidades surdas do Brasil.

Art. 3º As instituições públicas e empresas concessionárias de serviços públicos de assistência à saúde devem garantir atendimento e tratamento adequado aos portadores de deficiência auditiva, de acordo com as normas legais em vigor.

Art. 4º O sistema educacional federal e os sistemas educacionais estaduais, municipais e do Distrito Federal devem garantir a inclusão nos cursos de formação de Educação Especial, de Fonoaudiologia e de Magistério, em seus níveis médio e superior, do ensino da Língua Brasileira de Sinais – LIBRAS, como parte integrante dos Parâmetros Curriculares Nacionais – PCNs, conforme legislação vigente.

Parágrafo único. A Língua Brasileira de Sinais – LIBRAS não poderá substituir a modalidade escrita da língua portuguesa.

Art. 5º Esta Lei entra em vigor na data de sua publicação.

Brasília, 24 de abril de 2002; 181º da Independência e 114º da República.

FERNANDO HENRIQUE CARDOSO

ANEXO B - *SCHEMA* X-LIBRAS

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
	<xsd:element name="Posicao">
		<xsd:complexType>
			<xsd:attribute name="OX" use="required" type="xsd:string"/>
			<xsd:attribute name="OY" use="required" type="xsd:string"/>
			<xsd:attribute name="OZ" use="required" type="xsd:string"/>
		</xsd:complexType>
	</xsd:element>
	<xsd:element name="Sinal">
		<xsd:complexType>
			<xsd:sequence>
				<xsd:element ref="Config_Mao" minOccurs="1"
maxOccurs="unbounded"/>
				<xsd:element ref="Ponto_Articulacao" minOccurs="0"
maxOccurs="unbounded"/>
				<xsd:element ref="Movimento" minOccurs="0"
maxOccurs="unbounded"/>
				<xsd:element ref="Expressao" minOccurs="0"
maxOccurs="unbounded"/>
			</xsd:sequence>
			<xsd:attribute name="Nome" use="required" type="xsd:string"/>
		</xsd:complexType>
	</xsd:element>
	<xsd:element name="Cabeca">
		<xsd:complexType>
			<xsd:attribute name="Translacao" use="required"
type="CabecaTranslacaotype"/>
			<xsd:attribute name="Localizacao" use="required"
type="CabecaLocalizacaotype"/>
			<xsd:attribute name="Subdivisao" use="required"
type="CabecaSubdivisaotype"/>
		</xsd:complexType>
	</xsd:element>
	<xsd:element name="Ponto_Articulacao">
		<xsd:complexType>
			<xsd:choice>
				<xsd:element ref="Cabeca"/>
				<xsd:element ref="Tronco"/>
				<xsd:element ref="Bracos"/>
				<xsd:element ref="Mao"/>
			</xsd:choice>
			<xsd:attribute name="Espaco_Neutro" use="required"
type="Ponto_ArticulacaoEspaco_Neutrotype"/>
		</xsd:complexType>
	</xsd:element>
	<xsd:element name="Exp_Rosto">
		<xsd:complexType>
			<xsd:attribute name="Opcao" use="optional"
type="Exp_RostoOpcaotype"/>
		</xsd:complexType>
	</xsd:element>
	<xsd:element name="Exp_Cabeca">
		<xsd:complexType>
			<xsd:attribute name="Opcao" use="optional"
type="Exp_CabecaOpcaotype"/>
		</xsd:complexType>
	</xsd:element>
	<xsd:element name="Expressao">
		<xsd:complexType>
			<xsd:choice>
				<xsd:element ref="Exp_Cabeca"/>
```

```
                        <xsd:element ref="Exp_Rosto" minOccurs="0"
maxOccurs="unbounded"/>
                        <xsd:sequence>
                              <xsd:element ref="Exp_Cabeca"/>
                              <xsd:element ref="Exp_Rosto" minOccurs="0"
maxOccurs="unbounded"/>
                        </xsd:sequence>
                  </xsd:choice>
            </xsd:complexType>
      </xsd:element>
      <xsd:element name="Tronco">
            <xsd:complexType>
                  <xsd:attribute name="Translacao" use="optional"
type="TroncoTranslacaotype"/>
                  <xsd:attribute name="Localizacao" use="required"
type="TroncoLocalizacaotype"/>
                  <xsd:attribute name="Subdivisao" use="required"
type="TroncoSubdivisaotype"/>
                  <xsd:attribute name="Posicao" use="optional"
type="TroncoPosicaotype"/>
            </xsd:complexType>
      </xsd:element>
      <xsd:element name="Mao">
            <xsd:complexType>
                  <xsd:attribute name="Translacao" use="optional"
type="MaoTranslacaotype"/>
                  <xsd:attribute name="Localizacao" use="required"
type="MaoLocalizacaotype"/>
                  <xsd:attribute name="Subdivisao" use="required"
type="MaoSubdivisaotype"/>
            </xsd:complexType>
      </xsd:element>
      <xsd:element name="Movimento">
            <xsd:complexType>
                  <xsd:attribute name="Frequencia" use="optional"
type="MovimentoFrequenciatype"/>
                  <xsd:attribute name="Direcao" use="optional"
type="MovimentoDirecaotype"/>
                  <xsd:attribute name="Velocidade" use="optional"
type="MovimentoVelocidadetype"/>
                  <xsd:attribute name="Tipo" use="optional"
type="MovimentoTipotype"/>
                  <xsd:attribute name="Eixo" use="optional" type="xsd:string"/>
            </xsd:complexType>
      </xsd:element>
      <xsd:element name="Config_Dedo">
            <xsd:complexType>
                  <xsd:attribute name="Contato" use="required"
type="Config_DedoContatotype"/>
                  <xsd:attribute name="Dedo" use="required"
type="Config_DedoDedotype"/>
                  <xsd:attribute name="Ang_Contracao" use="required"
type="Config_DedoAng_Contracaotype"/>
            </xsd:complexType>
      </xsd:element>
      <xsd:element name="Config_Mao">
            <xsd:complexType>
                  <xsd:sequence>
                        <xsd:element ref="Posicao"/>
                        <xsd:element ref="Config_Dedo" minOccurs="0"
maxOccurs="unbounded"/>
                  </xsd:sequence>
                  <xsd:attribute name="Lado" use="required"
type="Config_MaoLadotype"/>
            </xsd:complexType>
      </xsd:element>
      <xsd:element name="Bracos">
```

```
            <xsd:complexType>
                  <xsd:attribute name="Translacao" use="optional"
type="BracosTranslacaotype"/>
                  <xsd:attribute name="Localizacao" use="required"
type="BracosLocalizacaotype"/>
                  <xsd:attribute name="Subdivisao" use="required"
type="BracosSubdivisaotype"/>
            </xsd:complexType>
      </xsd:element>
      <xsd:simpleType name="BracosTranslacaotype">
            <xsd:restriction base="xsd:string">
                  <xsd:enumeration value="Lateral"/>
                  <xsd:enumeration value="Frente"/>
                  <xsd:enumeration value="Atras"/>
            </xsd:restriction>
      </xsd:simpleType>
      <xsd:simpleType name="BracosLocalizacaotype">
            <xsd:restriction base="xsd:string">
                  <xsd:enumeration value="Lado_Direito"/>
                  <xsd:enumeration value="Lado_Esquerdo"/>
                  <xsd:enumeration value="Medial"/>
                  <xsd:enumeration value="Interna"/>
                  <xsd:enumeration value="Externa"/>
            </xsd:restriction>
      </xsd:simpleType>
      <xsd:simpleType name="BracosSubdivisaotype">
            <xsd:restriction base="xsd:string">
                  <xsd:enumeration value="Braco"/>
                  <xsd:enumeration value="Queixo"/>
                  <xsd:enumeration value="Antebraco"/>
                  <xsd:enumeration value="Cotovelo"/>
                  <xsd:enumeration value="Pulso"/>
            </xsd:restriction>
      </xsd:simpleType>
      <xsd:simpleType name="Config_MaoLadotype">
            <xsd:restriction base="xsd:string">
                  <xsd:enumeration value="Direito"/>
                  <xsd:enumeration value="Esquerdo"/>
            </xsd:restriction>
      </xsd:simpleType>
      <xsd:simpleType name="Config_DedoContatotype">
            <xsd:restriction base="xsd:string">
                  <xsd:enumeration value="Sim"/>
                  <xsd:enumeration value="Nao"/>
            </xsd:restriction>
      </xsd:simpleType>
      <xsd:simpleType name="Config_DedoDedotype">
            <xsd:restriction base="xsd:string">
                  <xsd:enumeration value="1"/>
                  <xsd:enumeration value="2"/>
                  <xsd:enumeration value="3"/>
                  <xsd:enumeration value="4"/>
                  <xsd:enumeration value="5"/>
            </xsd:restriction>
      </xsd:simpleType>
      <xsd:simpleType name="Config_DedoAng_Contracaotype">
            <xsd:restriction base="xsd:string">
                  <xsd:enumeration value="0"/>
                  <xsd:enumeration value="45"/>
                  <xsd:enumeration value="90"/>
            </xsd:restriction>
      </xsd:simpleType>
      <xsd:simpleType name="MovimentoFrequenciatype">
            <xsd:restriction base="xsd:string">
                  <xsd:enumeration value="Simples"/>
                  <xsd:enumeration value="Repetido"/>
            </xsd:restriction>
```

```
	</xsd:simpleType>
	<xsd:simpleType name="MovimentoDirecaotype">
		<xsd:restriction base="xsd:string">
			<xsd:enumeration value="Unidirecional"/>
			<xsd:enumeration value="Bidirecional"/>
			<xsd:enumeration value="Multidirecional"/>
		</xsd:restriction>
	</xsd:simpleType>
	<xsd:simpleType name="MovimentoVelocidadetype">
		<xsd:restriction base="xsd:string">
			<xsd:enumeration value="Tensao"/>
			<xsd:enumeration value="Retencao"/>
			<xsd:enumeration value="Continuidade"/>
			<xsd:enumeration value="Refreamento"/>
			<xsd:enumeration value="Repetido"/>
		</xsd:restriction>
	</xsd:simpleType>
	<xsd:simpleType name="MovimentoTipotype">
		<xsd:restriction base="xsd:string">
			<xsd:enumeration value="Retilineo"/>
			<xsd:enumeration value="Circular"/>
			<xsd:enumeration value="Continuo"/>
		</xsd:restriction>
	</xsd:simpleType>
	<xsd:simpleType name="MaoTranslacaotype">
		<xsd:restriction base="xsd:string">
			<xsd:enumeration value="Lateral"/>
			<xsd:enumeration value="Frente"/>
			<xsd:enumeration value="Atras"/>
		</xsd:restriction>
	</xsd:simpleType>
	<xsd:simpleType name="MaoLocalizacaotype">
		<xsd:restriction base="xsd:string">
			<xsd:enumeration value="Lado_Direito"/>
			<xsd:enumeration value="Lado_Esquerdo"/>
			<xsd:enumeration value="Medial"/>
			<xsd:enumeration value="Interna"/>
			<xsd:enumeration value="Externa"/>
		</xsd:restriction>
	</xsd:simpleType>
	<xsd:simpleType name="MaoSubdivisaotype">
		<xsd:restriction base="xsd:string">
			<xsd:enumeration value="Palma"/>
			<xsd:enumeration value="Costa"/>
			<xsd:enumeration value="Lado_Indicador"/>
			<xsd:enumeration value="Lado_Minimo"/>
			<xsd:enumeration value="Dedos"/>
			<xsd:enumeration value="Ponta_Dedos"/>
			<xsd:enumeration value="Juncao_Mao"/>
			<xsd:enumeration value="Junta_Dedos"/>
			<xsd:enumeration value="Dedo_Minimo"/>
			<xsd:enumeration value="Anular"/>
			<xsd:enumeration value="Dedo_Medio"/>
			<xsd:enumeration value="Indicador"/>
			<xsd:enumeration value="Polegar"/>
			<xsd:enumeration value="Intersticio_Polegar"/>
			<xsd:enumeration value="Intersticio_Indicador"/>
			<xsd:enumeration value="Intersticio_Medio"/>
			<xsd:enumeration value="Intersticio_Anular"/>
		</xsd:restriction>
	</xsd:simpleType>
	<xsd:simpleType name="TroncoTranslacaotype">
		<xsd:restriction base="xsd:string">
			<xsd:enumeration value="Lateral"/>
			<xsd:enumeration value="Frente"/>
			<xsd:enumeration value="Atras"/>
		</xsd:restriction>
```

```
	</xsd:simpleType>
	<xsd:simpleType name="TroncoLocalizacaotype">
		<xsd:restriction base="xsd:string">
			<xsd:enumeration value="Lado_Direito"/>
			<xsd:enumeration value="Lado_Esquerdo"/>
			<xsd:enumeration value="Medial"/>
			<xsd:enumeration value="Interna"/>
			<xsd:enumeration value="Externa"/>
		</xsd:restriction>
	</xsd:simpleType>
	<xsd:simpleType name="TroncoSubdivisaotype">
		<xsd:restriction base="xsd:string">
			<xsd:enumeration value="Pescoco"/>
			<xsd:enumeration value="Ombro"/>
			<xsd:enumeration value="Busto"/>
			<xsd:enumeration value="Estomago"/>
			<xsd:enumeration value="Cintura"/>
		</xsd:restriction>
	</xsd:simpleType>
	<xsd:simpleType name="TroncoPosicaotype">
		<xsd:restriction base="xsd:string">
			<xsd:enumeration value="Proximo"/>
			<xsd:enumeration value="Distancia_Media"/>
			<xsd:enumeration value="Distante"/>
			<xsd:enumeration value="Contato"/>
			<xsd:enumeration value="Contato_Inicial"/>
			<xsd:enumeration value="Contato_Medial"/>
			<xsd:enumeration value="Contato_Final"/>
		</xsd:restriction>
	</xsd:simpleType>
	<xsd:simpleType name="Exp_CabecaOpcaotype">
		<xsd:restriction base="xsd:string">
			<xsd:enumeration value="Frente_Tras"/>
			<xsd:enumeration value="Lados"/>
			<xsd:enumeration value="Frente"/>
			<xsd:enumeration value="Lado"/>
			<xsd:enumeration value="Tras"/>
		</xsd:restriction>
	</xsd:simpleType>
	<xsd:simpleType name="Exp_RostoOpcaotype">
		<xsd:restriction base="xsd:string">
			<xsd:enumeration value="Sobrancelha_Franzida"/>
			<xsd:enumeration value="Olhos_Arregalados"/>
			<xsd:enumeration value="Lance_Olhos"/>
			<xsd:enumeration value="Sobrancelha_Levantada"/>
			<xsd:enumeration value="Bochecha_Inflada"/>
			<xsd:enumeration value="Bochecha_Contraida"/>
			<xsd:enumeration value="Labios_Contraidos"/>
			<xsd:enumeration value="Lingua"/>
			<xsd:enumeration value="Bochecha_Direita_Inflada"/>
			<xsd:enumeration value="Contracao_Labio"/>
			<xsd:enumeration value="Franzir"/>
		</xsd:restriction>
	</xsd:simpleType>
	<xsd:simpleType name="Ponto_ArticulacaoEspaco_Neutrotype">
		<xsd:restriction base="xsd:string">
			<xsd:enumeration value="Sim"/>
			<xsd:enumeration value="Nao"/>
		</xsd:restriction>
	</xsd:simpleType>
	<xsd:simpleType name="CabecaTranslacaotype">
		<xsd:restriction base="xsd:string">
			<xsd:enumeration value="Lateral"/>
			<xsd:enumeration value="Frente"/>
			<xsd:enumeration value="Atras"/>
		</xsd:restriction>
	</xsd:simpleType>
```

```
	<xsd:simpleType name="CabecaLocalizacaotype">
		<xsd:restriction base="xsd:string">
			<xsd:enumeration value="Lado_Direito"/>
			<xsd:enumeration value="Lado_Esquerdo"/>
			<xsd:enumeration value="Medial"/>
			<xsd:enumeration value="Interna"/>
			<xsd:enumeration value="Externa"/>
		</xsd:restriction>
	</xsd:simpleType>
	<xsd:simpleType name="CabecaSubdivisaotype">
		<xsd:restriction base="xsd:string">
			<xsd:enumeration value="Topo"/>
			<xsd:enumeration value="Testa"/>
			<xsd:enumeration value="Rosto"/>
			<xsd:enumeration value="Rosto_Sup"/>
			<xsd:enumeration value="Rosto_Inf"/>
			<xsd:enumeration value="Orelha"/>
			<xsd:enumeration value="Olhos"/>
			<xsd:enumeration value="Nariz"/>
			<xsd:enumeration value="Boca"/>
			<xsd:enumeration value="Bochecha"/>
			<xsd:enumeration value="Queixo"/>
		</xsd:restriction>
	</xsd:simpleType>
</xsd:schema>
```

ANEXO C: CLASSES XLIBRASROTATION E XLIBRASVEC3F

```
import java.lang.*;
import java.util.*;
import XLibrasVec3f;

public class XLibrasRotation {
  XLibrasRotation() {
    quat = new float[4];
  }

  XLibrasRotation(float q0, float q1, float q2, float q3) {
    quat = new float[4];
    quat[0] = q0; quat[1] = q1; quat[2] = q2; quat[3] = q3;
    normalize();
  }

  XLibrasRotation(XLibrasVec3f axis, float radians) {
    quat = new float[4];
    setValue(axis, radians);
  }

  public XLibrasRotation setValue(XLibrasVec3f axis, float radians) {
    XLibrasVec3f q = new XLibrasVec3f(axis);

    q.normalize();
    q.multBy((float) Math.sin(radians / 2.0f));

    float[] val = q.getValue();
    quat[0] = val[0];
    quat[1] = val[1];
    quat[2] = val[2];

    quat[3] = (float) Math.cos(radians / 2.0f);

    return this;
  }

  public float getValue(XLibrasVec3f axis) {
    float   len;
    XLibrasVec3f q = new XLibrasVec3f(quat[0], quat[1], quat[2]);

    if ((len = q.length()) > 0.00001f) {
        q.multBy(1.0f / len);
      float[] val = q.getValue();
      axis.setValue(val[0], val[1], val[2]);
      return (float)(2.0 * Math.acos(quat[3]));
    }

    else {
      axis.setValue(0.0f, 0.0f, 1.0f);
      return 0.0f;
    }
  }

  XLibrasRotation times(XLibrasRotation q2) {
    XLibrasRotation q = new XLibrasRotation(q2.quat[3] * quat[0] +
q2.quat[0] * quat[3] + q2.quat[1] * quat[2] - q2.quat[2] * quat[1],
```

```
                          q2.quat[3] * quat[1] + q2.quat[1] * quat[3] +
                          q2.quat[2] * quat[0] - q2.quat[0] * quat[2],
                          q2.quat[3] * quat[2] + q2.quat[2] * quat[3] +
                          q2.quat[0] * quat[1] - q2.quat[1] * quat[0],
                          q2.quat[3] * quat[3] - q2.quat[0] * quat[0] -
                          q2.quat[1] * quat[1] - q2.quat[2] * quat[2]);
    q.normalize();
    return (q);
  }

  void normalize() {
    float   dist = (float)(1.0 / Math.sqrt(norm()));

    quat[0] *= dist;
    quat[1] *= dist;
    quat[2] *= dist;
    quat[3] *= dist;
  }

  float norm() {
    return (quat[0] * quat[0] +
            quat[1] * quat[1] +
            quat[2] * quat[2] +
            quat[3] * quat[3]);
  }

  float[] quat;
}
```

```
import java.lang.*;

public class XLibrasVec3f {

  XLibrasVec3f() {
    vec = new float[3];
  }

  XLibrasVec3f(XLibrasVec3f arg) {
    vec = new float[3];
    vec[0] = arg.vec[0]; vec[1] = arg.vec[1]; vec[2] = arg.vec[2];
  }

  XLibrasVec3f(float x, float y, float z) {
    vec = new float[3];
    setValue(x, y, z);
  }

  public float length() {
    return (float) Math.sqrt(vec[0] * vec[0] +
                        vec[1] * vec[1] +
                        vec[2] * vec[2]);
  }

  public void normalize() {
    float len = length();

    if (len != 0.0) {
      multBy(len);
    }
```

```
    else setValue(0.0f, 0.0f, 0.0f);
  }

  public void setValue(float x, float y, float z) {
    vec[0] = x; vec[1] = y; vec[2] = z;
  }

  float[] getValue() {
    return vec;
 }
    vec[0] *= arg;
    vec[1] *= arg;
    vec[2] *= arg;
  }

  XLibrasVec3f times(float arg) {
    XLibrasVec3f v = new XLibrasVec3f();
    v.vec[0] = vec[0] * arg;
    v.vec[1] = vec[1] * arg;
    v.vec[2] = vec[2] * arg;
    return v;
  }
   float[] vec;
}
```