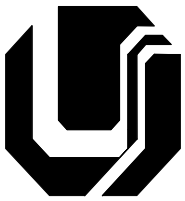
UNIVERSIDADE FEDERAL DE UBERLÂNDIA

FACULDADE DE ENGENHARIA ELÉTRICA

PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA



ARQUITETURA PARA DISTRIBUIÇÃO DE AMBIENTES VIRTUAIS MULTIDISCIPLINARES

ORIENTADOR: EDGARD LAMOUNIER JÚNIOR, PhD

CO-ORIENTADOR: ALEXANDRE CARDOSO, Dr

ORIENTANDO: MARCOS WAGNER DE SOUZA RIBEIRO

JANEIRO
2006

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA ELÉTRICA
PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

ARQUITETURA PARA DISTRIBUIÇÃO DE AMBIENTES VIRTUAIS MULTIDISCIPLINARES

Tese apresentada por Marcos Wagner de Souza Ribeiro à Universidade Federal de Uberlândia para obtenção do título de Doutor em Ciências.

Professor Edgard Lamounier Jr., PhD (Orientador)

Professor Alexandre Cardoso, Dr (Co-Orientador)

Professor Cláudio Kirner, Dr

Professor Luciano Vieira Lima, Dr

Professor Paulo Roberto Guardieiro, Dr

Professor Djamel Sadok, PhD

Uberlândia, 30 de janeiro de 2006.

ARQUITETURA PARA DISTRIBUIÇÃO DE AMBIENTES VIRTUAIS MULTIDISCIPLINARES

MARCOS WAGNER DE SOUZA RIBEIRO

Tese apresentada por Marcos Wagner de Souza Ribeiro à Universidade Federal de Uberlândia como parte dos requisitos para obtenção do título de Doutor em Ciências.

Profº Edgard Lamounier Júnior Orientador	Profº. Darizon Alves Andrade Coordenador do Curso de Pós-Graduação

DEDICATÓRIA

À minha esposa, Eliane

À minha filha, Stefani

Ao meu irmão, Dioni

Aos meus pais José Ribeiro e Odília

AGRADECIMENTOS

Agradeço primeiramente a Deus, que permitiu a finalização de uma mais uma etapa em minha vida.

Ao professor e amigo Edgard Lamounier, orientador deste trabalho, pela valiosa orientação e por me mostrar sempre o caminho a seguir.

Ao professor Alexandre Cardoso pelo apoio e amizade na co-orientação deste trabalho.

E, a todos aqueles que contribuíram de forma direta ou indireta para a realização deste trabalho.

RESUMO

RIBEIRO, Marcos Wagner de Souza. *Arquitetura para Distribuição de Ambientes Virtuais Multidisciplinares*, Uberlândia, Faculdade de Engenharia Elétrica - UFU, 2005, 176p.

Esta tese apresenta uma arquitetura para distribuição de ambientes virtuais como ferramenta de apoio a projetos multidisciplinares de ensino. Para tanto, diferentes plataformas de distribuição foram avaliadas com o objetivo de identificar aquela que com mais eficiência permita que interações ocorridas em um ambiente altere o comportamento de outros, mesmo que estes sejam relacionados a outras áreas do conhecimento. Protótipos construídos sobre a plataforma escolhida para a distribuição, seguindo uma mesma metodologia (onde aspectos do modelo de dados foram alterados) e ainda, tendo a latência, escalabilidade e extensibilidade como parâmetros de comparação demonstraram qual a melhor abordagem para construção de ambientes virtuais multidisciplinares. Cada protótipo foi construído com base em algoritmos de distribuição que permitiram ao sistema funcionar corretamente em situações passíveis de erros. Ambientes Virtuais de Biologia (paisagem com plantas, água, luz e terra) e Química (membrana de uma folha) foram utilizados tendo o fenômeno da fotossíntese como estudo de caso e relação entre os dois ambientes. O sistema foi avaliado por professores e alunos e os resultados alcançados permitiram concluir que o mesmo é eficaz e aplicável.

Palavras-Chave: Ambientes Virtuais Distribuídos, CORBA, Multidisciplinaridade, Realidade Virtual

ABSTRACT

RIBEIRO, Marcos Wagner de Souza. *Architecture To Support Multidisciplinary Learning Through Virtual Environment Distribution*, Uberlândia, Faculty of Electrical Engineering - UFU, 2005, 176p.

This thesis presents a study of architecture for virtual environment distribution as support tool for learning multidisciplinary projects. Different distribution platforms have been evaluated with the objective to identify the one that with more efficiency allowed exactly that occurred alterations in an environment modify the behavior of others, that these are related to other areas of the knowledge. Building prototypes under the platform chosen for the distribution, following same methodology (modifying aspects of the data model) and having the latency, scalability and extensibility as method of comparison had demonstrated to which the best choice for building multidisciplinary virtual environment. Each prototype was constructed on the basis of distribution algorithms that had allowed the system to function correctly in possible situations of errors. Virtual environments of Biology (landscape with plants, water, light and land) and Chemistry (membrane and molecules) had been used having the phenomenon of the photosynthesis as study of case and relation between two environments. The system was evaluated by professors and students and the reached results had allowed concluding that it is efficient and applicable.

Keywords: CORBA, Distributed Virtual Environments, Multidisciplinary, Virtual Reality

PUBLICAÇÕES

A seguir são apresentadas as publicações resultantes desse trabalho:

RIBEIRO, Marcos Wagner de Souza; LAMOUNIER, Edgard Jr.; CARDOSO, Alexandre. Uso de CORBA na Distribuição de Ambientes Virtuais para Suportar Multidisciplinaridade no Processo da Educação. SVR 2004 VII Symposium on Virtual Reality, 7, 2004, São Paulo. *Proceedings...* São Paulo: SENAC, 2004, p.356-358.

RIBEIRO, Marcos Wagner de Souza; LAMOUNIER, Edgard Jr.; CARDOSO, Alexandre. Uso de CORBA na Distribuição de Ambientes Virtuais. *Práxis*, Canoas-RS, n. 5, p43-52, ago/dez, 2004.

RIBEIRO, Marcos Wagner de Souza; LAMOUNIER, Edgard Jr.; CARDOSO, Alexandre. A Study of Distinct Virtual Environments Distribution. *Práxis*, Canoas-RS, n. 6, p49-56, jan/jul, 2005.

RIBEIRO, Marcos Wagner de Souza; LAMOUNIER, Edgard Jr.; CARDOSO, Alexandre. A Proposed Architecture To Support Multidisciplinary Learning Through Virtual Environment Distribution. IASTED International Conference, 17, 2005, Phoenix, AZ, USA. *Proceedings on Parallel and Distributed Computing And Systems – PDCS 2005*, Phoenix: IASTED, 2005. p.56-61.

RIBEIRO, Marcos Wagner de Souza; LAMOUNIER, Edgard Jr.; CARDOSO, Alexandre. Uma Proposta de Arquitetura na Distribuição de Ambientes Virtuais Muldisciplinares. Workshop de Aplicações de Realidade Virtual, 1, 2005, Uberlândia. *Anais...* Uberlândia-MG: UFU, 2005. 1 CD-ROM.

SIQUEIRA, Luiz Leonardo, RIBEIRO, Marcos Wagner de Souza; LAMOUNIER, Edgard Jr.; CARDOSO, Alexandre. Estudo Comparativo Entre Plataformas de Suporte a Ambientes Virtuais Distribuídos. Workshop de Aplicações de Realidade Virtual, 1, 2005, Uberlândia. *Anais...* Uberlândia-MG: UFU, 2005. 1 CD-ROM.

RIBEIRO, Marcos Wagner de Souza; LAMOUNIER, Edgard Jr.; CARDOSO, Alexandre. A Proposed Architecture to Support Multidisciplinary Learning through Virtual Environment Distribution. *IEEE Transactions Parallel and Distributed Systems*. ***Submetido em novembro de 2005***.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

2D – Bidimensional

3D – Tridimensional

API – ***A**pplication **P**rogramming **I**nterface*

AV – **A**mbiente **V**irtual

AVD – **A**mbiente **V**irtual **D**istribuído

BOA – ***B**asic **O**bject **A**dapter*

COM – ***C**ommon **O**bject **M**odel*

CORBA – ***C**ommon **O**bject **R**equest **B**roker **A**rchitecture*

COSS – ***C**ommon **O**bject **S**ervice **S**pecifications*

DCE – ***D**istributed **C**omputing **E**nvironment*

DIVE – ***D**istributed **I**nteractive **V**irtual **E**nvironment*

GUI – ***G**raphics **U**ser **I**nterface*

HMD – ***H**ead **M**ounted **D**isplay*

IDL – ***I**nterface **D**efinition **L**anguage*

IIOP – ***I**nternet **I**nter-**O**BR **P**rotocol*

ILU – ***I**nter-**L**anguage **U**nification*

ISO – ***I**nternational **S**tandards **O**rganization*

ms – Milissegundo

OMA – ***O**bject **M**anagement **A**rchitecture*

OMG – ***O**bject **M**anagement **G**roup*

OpenGl – ***Open G**raphics **L**ibrary*

ORB – ***O**bject **R**equest **B**roker*

OSF – ***O**pen **S**oftware **F**oundation*

OSI – ***O**pen **S**ystem **I**nterconnect*

RPC – ***R**emote **P**rocedure **C**all*

TCP – ***T**ransfer **C**ommunication **P**rotocol*

UDP – ***U**ser **D**atagram **P**rotocol*

VR – ***V**irtual **R**eality*

VRML – ***V**irtual **R**eality **M**odeling **L**anguage*

CAPÍTULO I

1. INTRODUÇÃO

Realidade Virtual (RV) é considerada uma tecnologia revolucionária, pois possibilita a simulação de mundos reais e imaginários na tela do computador ou em outros dispositivos, criando no usuário a sensação de presença em um "mundo" virtual (CRUZ-NEIRA, 1992; CARLSON, 1993). As pesquisas em RV vêm crescendo consideravelmente, por meio de vários grupos de pesquisa, ligados à indústria, ao entretenimento e à educação (AUKSTAKALNIS 1992; CODDELA, 1993 apud NUNES, 2002). Além disso, existem muitos estudos, soluções e implementações para possibilitar que mais de uma pessoa faça parte de um Ambiente Virtual (AV) (SNOWDON, 1994; GARCIA, 2002; KOTZIAMPASIS, 2003; ALIMA, 2004; YARDI, 2005, JING, 2005). Os sistemas distribuídos em parceria com sistemas RV possibilitaram a criação dos Ambientes Virtuais Distribuídos (AVDs), definidos como ambientes virtuais que permitem a participação de diversos usuários ao mesmo tempo.

Nesse contexto, as principais pesquisas concentram-se na melhoria do processo de comunicação entre as cópias de um ambiente (SEMENTILLE, 1999). Porém, dependendo da aplicação, nem sempre a preocupação será com a comunicação entre cópias de um ambiente, mas também com a comunicação entre ambientes virtuais distintos, que não fazem parte do mesmo contexto, embora possuam relação de dependência (SIQUEIRA, 2005).

Na área da Educação, é comum, pesquisadores estudarem detalhadamente as relações entre conteúdos ou disciplinas com a finalidade de demonstrar que não é possível entender a

complexidade[1] do todo, sem entender separadamente os conteúdos envolvidos. Pode-se denominar esse tipo de pesquisa, de acordo com a especificidade de cada um, como Interdisciplinaridade[2], Transdisciplinaridade[3] ou Multidisciplinaridade. Piaget (1972) define Multidisciplinaridade quando há necessidade de obter informações de duas ou mais ciências ou setores do conhecimento sem que as disciplinas envolvidas no processo sejam elas mesmas modificadas ou enriquecidas.

Sendo assim, técnicas de distribuição de ambientes virtuais multidisciplinares devem ser pesquisadas para possibilitar novas formas de interação (influenciar uma realidade participando de outra), quando existem conteúdos diferentes que se relacionam.

1.1. Objetivo

Esta tese tem por objetivo identificar uma abordagem computacional/algorítmica que seja suficiente para suportar uma distribuição de ambientes virtuais multidisciplinares. Para atingir este objetivo, as seguintes **metas** foram definidas:

a) Identificar uma arquitetura de distribuição que apresente eficácia na distribuição de ambientes virtuais.

b) Escolher uma metodologia para o desenvolvimento de ambientes virtuais distribuídos, associada a um conjunto de algoritmos que aperfeiçoem esta distribuição.

c) Realizar a implementação de protótipos, segundo a metodologia elaborada no

[1] Do latim "plexus" que significa entrelaçado ou que possuem inter-relações.
[2] Interdisciplinaridade: é a interação entre duas ou mais disciplinas, transferindo métodos de uma disciplina à outra. Por exemplo, quando os métodos da física nuclear são transferidos para a medicina, resultam no aparecimento de novos tratamentos de câncer. Outro exemplo de interdisciplinaridade se, ao estudar a pintura, relacionássemos o contexto histórico do Renascimento com os temas usados pelos artistas de então e sobre as técnicas empregadas por eles (GIRARDELLI, 2005).
[3] Transdisciplinaridade: é a interação entre duas ou várias disciplinas proporcionando a criação de um corpo de elementos que compõem uma disciplina original, pois engloba e transcende o que passa por todas as disciplinas, reconhecendo o desconhecido e o inesgotável que estão presentes em todas elas, buscando encontrar seus pontos de interseção. Um bom exemplo de transdisciplinaridade são as grandes teorias explicativas do funcionamento das sociedades (GIRARDELLI, 2005).

item anterior, visando a sua validação.

d) Comparar os protótipos, consolidando o melhor modelo para criação de um AVD multidisciplinar.

e) Verificar a aplicabilidade do AVD desenvolvido.

Durante o desenvolvimento deste trabalho, as seguintes **estratégias** foram adotadas, norteando a consolidação do objeto e metas propostas:

a) Modelar dois ambientes virtuais relacionados a duas áreas distintas do conhecimento. Propõe-se como estudo de caso o Processo da Fotossíntese (Biologia e Química).

b) Implementar protótipo com comunicação entre os dois ambientes, seguindo as etapas:

1) Comunicação unidirecional - apenas um ambiente interferindo no outro, com uma cópia de cada ambiente.
2) Comunicação bidirecional - cada ambiente interfere no outro, com uma cópia de cada ambiente.
3) Comunicação bidirecional - com um ambiente e *n* cópias de outro, possibilitando além da interferência de um em outro, também a replicação das informações nas cópias existentes.
4) Comunicação bidirecional - com *n* cópias de um ambiente e *m* cópias de outro.

c) Por meio de comparação, justificar a escolha pela arquitetura de distribuição.

d) Pesquisar as principais características dos AVDs (Ambientes Virtuais Distribuídos) avaliados para este trabalho.

e) Implementar outros protótipos, comparando-os por meio de aspectos como a

latência de comunicação, escalabilidade e extensibilidade para medir o desempenho de cada um.

f) Aplicar o sistema no ensino (estudantes, professores e pesquisadores), avaliando suas principais características de acordo com normas de qualidade de software educacional.

1.2. Ambientes Virtuais Distribuídos

A Realidade Virtual pode empregar várias técnicas para reproduzir o mundo real e imaginário e possibilitar a manipulação e visualização de informações no computador como se fosse no mundo real. A complexidade desses ambientes virtuais aumenta na medida em que essas informações tornam-se comuns a uma série de usuários, ou seja, esses ambientes são distribuídos. Dessa forma, a Realidade Virtual possibilita interação entre usuários dispersos por meio do uso de ambientes virtuais distribuídos.

Existem diversas plataformas que proporcionam essa distribuição como CORBA, RPC, Java/RMI (RAJ, 2004) e, para a modelagem do ambiente, também existem diversas linguagens em que se podem construir modelos virtuais como VRML, JAVA3D, OPENGL[4].

Um AVD pode ser definido de forma simplificada como um sistema que permite vários usuários interagirem tanto com o ambiente quanto entre "eles" em tempo real, mesmo que estes estejam em diferentes localidades geográficas. Existem bibliografias que classificam os AVDs em dois tipos, considerando a dimensão do ambiente (OST, 2004):

- Espaços amplos e abertos, que simulam áreas relativamente grandes como cidades e desertos;
- Espaços pequenos, que se concentram na criação de modelos como salas ou edifícios.

[4] OPENGL é uma biblioteca de rotinas gráficas e de modelagem 2D e 3D (WOO, 1999).

As primeiras experiências com ambientes virtuais distribuídos estão associadas a aplicações de simulações de batalhas militares feitas por universidades e órgãos governamentais dos Estados Unidos, como o Exército, a Marinha e o Departamento de Defesa (MACEDONIA, 2005).

A área de aplicação dos AVD's é muito abrangente - treinamento (pilotos, militar), pesquisa, ensino à distância, comércio, cultura, engenharia. O computador apresenta um grande potencial como ferramenta de apoio ao ensino, que aliada às técnicas de Realidade Virtual e colaboração, pode enriquecer e valorizar a informação transmitida, simulando realidades, muitas vezes, fora do alcance dos alunos. Existe até mesmo um consenso entre os educadores/pesquisadores de que a RV oferece aos educadores uma nova maneira de ensinar e, aos alunos, uma forte motivação (DIZERO, 2004). A razão mais óbvia é que a RV é uma maneira diferente que habilita as pessoas a fazerem coisas, às quais não teriam acesso no mundo físico (KALAWSKY, 1993).

1.3. Contribuição do Trabalho

Existem várias pesquisas na área de distribuição de ambientes virtuais, que vão desde pequenos ambientes que usam de simples replicação a técnicas de particionamento, métodos conhecidos como *dead-reckoning* que analisam o comportamento do objeto, a ambientes virtuais de grande escala, como é o caso do sistema de treinamento de guerra NPSNET (MACEDONIA, 2005), em que técnicas de gerenciamento de recursos apresentam maior importância.

Todos estes ambientes e outros que serão apresentados neste estudo, simulam ou oferecem apenas um tipo de mundo (unidisciplinar) ou apenas uma área do conhecimento é explorada. Em alguns casos, há a existência de ambientes diferentes, porém dentro de um mesmo contexto.

Este trabalho pretende contribuir com a descrição de uma abordagem (arquitetura) que possibilite a distribuição de ambientes virtuais que simulem diferentes realidades. Além disso, essa arquitetura deve permitir que alterações feitas na realidade de um ambiente virtual possam ser propagadas, de tal maneira que altere a realidade de outros ambientes relacionados e vice-versa. No entanto, sistemas com estas características podem ser construídos de diversas formas ou metodologias (algoritmos). Para tanto, este trabalho contribuirá também com outras abordagens. No entanto, um AVD tem como requisito principal na avaliação de sua qualidade, a quantidade de clientes interagindo entre si ao mesmo tempo (escalabilidade). Portanto, este trabalho destacará a abordagem que melhor atenda a este requisito.

1.4. Organização da Tese

O trabalho está dividido em oito capítulos, descritos resumidamente a seguir:

O Capítulo 2 apresenta a caracterização dos Sistemas Distribuídos (software distribuído), plataformas mais conhecidas, visão geral e comparação de cada uma delas e definição da plataforma escolhida para uso neste trabalho. Ao final do capítulo, são abordadas características da arquitetura escolhida e a implementação usada, comparando-a com a especificação original.

O Capítulo 3 apresenta a caracterização e metodologia de classificação de Ambientes Virtuais Distribuídos, relatando os AVDs mais importantes atualmente.

No Capítulo 4, é descrita a arquitetura usada na distribuição dos Ambientes Virtuais.

A implementação dos protótipos, criados para validar a arquitetura, é apresentada no Capítulo 5.

O Capítulo 6 descreve o funcionamento do sistema, detalhando o estudo de caso multidisciplinar (Processo da Fotossíntese)

O Capítulo 7 mostra uma análise dos protótipos desenvolvidos, avaliando-se os

aspectos representativos em seu desempenho.

O Capítulo 8 apresenta as conclusões obtidas neste trabalho e as sugestões para trabalhos futuros que poderão ser desenvolvidos.

Por último têm-se as referências bibliográficas aqui utilizadas.

CAPÍTULO II

2. SISTEMAS DISTRIBUÍDOS

2.1. Introdução

Este capítulo apresenta as principais características dos sistemas computacionais distribuídos e detalha as plataformas de distribuição de software existentes no mercado e as diferenças entre elas.

Neste capítulo, também é discutida a metodologia proposta nesta tese para utilização de uma arquitetura como suporte para as aplicações distribuídas e detalha a implementação escolhida.

2.2. O que é distribuição?

Segundo Tanenbaum (1990), um sistema distribuído é uma coleção de computadores independentes que aparecem para os usuários do sistema como um único computador.

Outra definição seria um sistema em que componentes de *hardware* e *software* localizados em computadores de uma rede se comunicam por meio de mensagens (COULOURIS, 2005). Portanto, um "Sistema Distribuído é aquele que “roda” em um conjunto de máquinas sem memória compartilhada, aparecendo como um único computador para seus usuários" (LAMPORT, 2003).

Havendo essa interação entre computadores, uma aplicação pode ser dividida em diferentes partes que se comunicam e cada parte pode ser processada em um sistema independente.

Segundo Jalote (1994), sistema distribuído cria a sensação de que toda rede de computadores é um único sistema de tempo compartilhado (*time-sharing*), em vez de um grupo de máquinas independentes.

Um sistema distribuído, segundo Eckhouse Jr. (1978), é formado por um conjunto de módulos, compostos pelo menos por processador-memória, interligados frouxamente por meio de um subsistema de comunicação de topologia arbitrária. Esse *hardware* deve oferecer facilidades de comunicação entre processadores e entre processos, os quais, cooperando sob um controle descentralizado, possibilitam a execução de programas de aplicação.

A definição de sistema distribuído não implica a distribuição geográfica dos computadores que o compõem, pois a conexão fraca pode existir em ambientes confinados ou espalhados (KIRNER, 2005).

Características de um sistema distribuído (SPECTOR, 1981 apud KIRNER, 1988):

a) Multiplicidade de nós de processamento para permitir aumento no desempenho, tolerância a falhas (presença de defeitos ou interferências indevidas, que aparecem tanto em nível de *hardware* quanto de *software*), disponibilidade do sistema e/ou uso de dados geograficamente dispersos.

b) Mecanismos de comunicação para suportar comunicações entre os componentes do sistema distribuído.

c) Isolação entre componentes para suportar controle de diferentes entidades administrativas, tolerância a falhas e disponibilidade.

d) Possibilidade de expansão para permitir crescimento incremental.

e) Mecanismos de detecção de erros para obter disponibilidade e tolerância a falhas.

f) Redundância para obter disponibilidade e tolerância a falhas, pois, uma vez que os erros tenham sidos detectados, a redundância é que permitirá a recuperação.

g) Dispersão geográfica para permitir que os recursos sejam separados geograficamente.

O sistema deve possuir, obrigatoriamente, as características *a* e *b* mencionadas acima, e excepcionalmente, algumas das outras características.

Outra caracterização de sistema distribuído define outros ingredientes básicos (ENSLOW JR, 1974 apud KIRNER, 1988):

a) Multiplicidade de recursos de uso geral (físicos e lógicos), cuja concessão a tarefas específicas deve ocorrer de forma dinâmica.

b) Distribuição física desses recursos, interagindo por meio de uma rede de comunicação.

c) Existência de um sistema operacional de alto nível que unifica e integra o controle dos componentes distribuídos.

Uma outra forma de caracterizar os sistemas distribuídos baseia-se em três itens (PARDO, 1979 apud KIRNER, 1988):

a) O hardware básico de um sistema distribuído consistirá de uma rede de computadores.

b) A arquitetura física do sistema de comunicação apresenta pouca importância do ponto de vista lógico, podendo impor maior impacto em questões como desempenho.

c) *Software* é a palavra-chave em sistemas distribuídos. Em um sistema distribuído, a cooperação entre sistemas remotos é bastante elaborada, uma vez que envolve a implementação de compartilhamento implícito de múltiplos recursos remotos.

Aspectos de Tolerância a Falhas

Pelo fato de sistemas distribuídos possuírem múltiplas partes de *hardware* e de *software* funcionando conjuntamente, as chances de alguma dessas partes falhar é bem maior do que num sistema simples (KIRNER, 1988).

Um sistema distribuído pode apresentar diversos tipos de falhas provenientes de vários fatores. Para propiciar confiabilidade a um sistema distribuído é necessário que apresente tolerância a falhas.

Em sistemas tolerantes a falhas, as interferências indevidas podem ser superadas por meio de redundâncias (SIEWIOREK, 1984).

A redundância pode ser temporal e/ou física. A redundância física consiste em ter-se elementos repetidos capazes de executar a mesma operação. A redundância temporal corresponde à determinação de um resultado por meio de execuções repetidas de uma mesma operação pelo mesmo elemento, usando eventualmente métodos diferentes (KIRNER, 1988).

2.2.1 *Middleware*

Middleware é um termo que implica uma camada de software que proporciona uma abstração e assim esconde a heterogeneidade da rede, hardware, sistema operacional e linguagens de programação usadas.

Seu principal objetivo é facilitar o desenvolvimento de aplicações e sistemas distribuídos e o seu beneficio é ocultar do programador diferenças entre:

- Plataformas de hardware de sistemas operacionais.

- Bibliotecas de comunicação.

- Protocolos de comunicação.

- Formatação de dados.

- Linguagens de programação.

- Modelos de programação.

O *Middleware* possui a seguintes características:

Tem de estar disponível em diversas máquinas. As transferências têm de ser fiáveis, ou seja, tem de existir garantia de entrega e recebimento.

A diversidade das estruturas de comunicação, ou seja, uma aplicação pode se comunicar com outra aplicação ou enviar uma única mensagem para vários destinos.

2.3. Plataformas de distribuição de software

Três dos mais populares paradigmas de objetos distribuídos são (RAJ, 2004):

- *Common Object Request Broker Architecture* – CORBA da OMG.

- Java/*Remote Method Invocation* - Java/RMI da JavaSoft.

- *Distributed Object Component Model* – DCOM da Microsoft.

2.3.1 CORBA

CORBA é a especificação de uma arquitetura para objetos distribuídos e heterogêneos. O padrão CORBA é uma solução aberta de objetos distribuídos desenvolvido pela OMG (*Object Management Group*) para se tornar um padrão no mercado. Recebe destaque por ser independente de linguagem e fabricante, possibilitando que objetos de sistemas distribuídos troquem mensagens entre si de forma transparente, não importando onde eles estejam, em que plataforma ou sistema operacional estejam rodando, em que linguagem

de programação foram implementados e até mesmo qual protocolo de comunicação utilizam (LIMA, 2004).

2.3.1.1. Histórico

CORBA começou a ser desenvolvido em 1989, quando um grupo de empresas reuniu-se em uma organização (OMG), com o objetivo de especificar uma arquitetura global e normas para permitir o trabalho conjunto de componentes de diferentes origens. A idéia era aproveitar os benefícios do paradigma da orientação a objeto, principalmente a noção de encapsulamento dos dados e da implementação de um objeto por meio da sua interface.

O CORBA 1.1 foi inicialmente implementado em 1991, como um modelo que permite a ativação de métodos de objetos por meio de um elemento intermediário chamado ORB (*Object Request Broker*), situado entre o objeto (que se encontra na camada de aplicação do modelo *Open System Interconnection* – OSI) e o sistema operacional, com a possibilidade de comunicar-se em uma rede. Em 1994, foi desenvolvido o CORBA 2.0, ao qual foi adicionado o *Internet Inter-ORB Protocol* (IIOP). O IIOP permitiu que objetos fossem desenvolvidos em implementações diferentes, e assim, CORBA se tornou uma solução para adquirir a interoperabilidade entre objetos que não ficam presos a um padrão específico (OMG, 2004).

2.3.1.2. Características

A arquitetura definida para CORBA possui um alto nível de abstração, permitindo que a implementação do cliente e do servidor possa ser feita em qualquer linguagem e que os objetos se comuniquem de forma totalmente transparente por meio de um "barramento de *software*". Isso só é possível porque os objetos têm suas interfaces descritas em uma

linguagem padrão, chamada de *Interface Definition Language* (IDL). A função da IDL é descrever as interfaces das implementações de objetos, que são acessadas por seus clientes.

Foi definida pela OMG uma arquitetura denominada OMA (*Object Management Architecture*) para realizar a integração entre aplicações. Enquanto CORBA permite a interoperabilidade entre objetos, a OMA agrupa um conjunto de objetos CORBA em serviços e facilidades, que oferecem suporte para o desenvolvimento de aplicações que usam objetos CORBA (PAULOVICH, 2004).

Tudo na arquitetura CORBA depende de um *Object Request Broker* (ORB). O ORB atua como um *Object Bus* central sobre cada objeto CORBA, interagindo transparentemente com outros objetos localizados no mesmo computador ou remotamente. Cada objeto servidor CORBA tem uma interface e expõe um grupo de métodos. Para solicitar um serviço, um cliente adquire uma referência de objeto para o objeto servidor CORBA. O cliente pode fazer pedidos de métodos na referência de objeto como se o objeto servidor residisse no espaço de endereço do cliente. O ORB é responsável por achar a implementação de objetos CORBA, preparando-a para receber e enviar pedidos e carregar a resposta de volta ao cliente. Uma vez que CORBA é somente uma especificação, pode ser usada em diversas plataformas de sistemas operacionais de *mainframes* a UNIX, de máquinas Windows a aparelhos *handheld*, desde que haja uma implementação ORB para aquela plataforma (NEVES JÚNIOR, 2004).

2.3.2 DCOM

DCOM, freqüentemente chamado *'COM no fio',* suporta objetos remotos rodando em um protocolo chamado *Object Remote Procedure Call* (ORPC). Essa camada ORPC é

construída no topo do DCEs RPC[5] e interage com serviços em tempo de execução do COM. Um servidor DCOM é um corpo de códigos que é capaz de servir objetos de um tipo particular em tempo de execução. Cada objeto servidor DCOM pode suportar múltiplas interfaces, cada uma representando um comportamento diferente do objeto. Um cliente DCOM faz requisições por meio dos métodos expostos em um servidor DCOM, adquirindo um ponteiro para uma das interfaces do objeto servidor. O objeto cliente então começa a chamar os métodos expostos do objeto servidor, por meio de um ponteiro de interface adquirida como se o objeto servidor residisse no espaço de endereço do cliente. Já que a especificação COM está no nível binário, permite que componentes servidores DCOM sejam escritos em diversas linguagens de programação como C++, Java, Object Pascal (Delphi), Visual Basic e até COBOL. Desde que a plataforma sustente serviços COM, DCOM pode ser usado nesta plataforma.

2.3.3 JAVA/RMI

Java/RMI depende de um protocolo chamado *Java Remote Method Protocol* (JRMP)**.** Java depende muito da Serialização de Objetos, que permite aos objetos serem reunidos (ou transmitidos) como uma fila. Como *Java Object Serialization* é específica para Java, ambos os objetos servidores Java/RMI e os objetos cliente têm que ser escritos em Java. Cada objeto servidor Java/RMI define uma interface, que pode ser usada para acessar os objetos servidores de fora da atual *Java Virtual Machine* (JVM) e em outras máquinas JVM. A interface expõe um grupo de métodos, que são indício dos serviços oferecidos pelo objeto servidor. Para um cliente localizar um objeto servidor pela primeira vez, RMI depende de um mecanismo chamado *RMIRegistry*, que roda na máquina servidora e armazena informações

[5] DCE (*Distributed Computing Environment*) e RPC (*Remote Procedure Call*): são tecnologias de computação distribuída (RICCIONI, 2000).

sobre objetos servidores disponíveis. Um cliente Java/RMI adquire uma referência de objeto para um objeto servidor Java/RMI, fazendo um *lookup* para uma referência de Objeto Servidor e invoca métodos no Objeto Servidor como se o objeto servidor Java/RMI residisse no espaço de endereço do cliente. Objetos servidores Java/RMI são chamados usando URLs, e para um cliente adquirir uma referência de objeto servidor deveria especificar a URL do objeto servidor, como é feito com a URL para uma página HTML. Já que Java/RMI depende de Java, pode ser usada em diversas plataformas de sistemas operacionais de *mainframes* a UNIX, de máquinas Windows a aparelhos *handheld,* desde que haja uma implementação de *Java Virtual Machine* (JVM) para aquela plataforma. Além de JavaSoft e Microsoft, muitas outras companhias têm anunciado portas de *Java Virtual Machine*.

2.4. Comparação entre plataformas de distribuição

De acordo com Raj (2004), é necessário implementar alguma aplicação para comparar plataformas de distribuição, pois as principais diferenças estão no desempenho, na portabilidade, na facilidade de construção dos objetos e nos resultados obtidos por meio de testes.

2.4.1 Uma Aplicação

Para construir uma aplicação distribuída é necessário fazer um planejamento que segue o modelo tradicional de projeto definido dentro da Engenharia de Software[6]. Porém, a partir da extração de dados (engenharia de requisitos), alguns aspectos são agregados. Na camada Arquitetura é necessária uma definição do meio em que o sistema irá funcionar, apesar de o meio suportar heterogeneidade de *software* e *hardware*. Na camada Projeto, os itens mais importantes a serem adicionados são:

[6] Métodos e técnicas para construção de um *software* (LOPES, 2002).

- Função da aplicação cliente.

- Função da aplicação servidora.

- Definição da interface a ser usada entre cliente e servidor.

Para prover resultados na avaliação das três plataformas citadas anteriormente, serão considerados os mecanismos de implementação de Interfaces (IDL), modelo de cliente e modelo de servidor.

2.4.1.1. A interface IDL

Toda vez que o cliente precisa de algum serviço de um objeto distribuído remoto, ele invoca um método implementado pelo objeto remoto. O serviço que o objeto distribuído remoto (servidor) oferece é encapsulado como um objeto e a interface do objeto remoto são escritos em uma Linguagem de Definição de Interface (IDL). As interfaces especificadas no arquivo IDL servem como um contrato entre um objeto servidor remoto e seus clientes. O cliente pode, dessa forma, interagir com esses objetos servidores remotos, invocando métodos definidos na IDL.

DCOM – O arquivo IDL DCOM mostra que o servidor DCOM implementa interface dupla. COM sustenta invocação estática e dinâmica de objetos. É um pouco diferente de como CORBA faz por meio da sua Interface de Invocação Dinâmica (DII) ou Java faz com *Reflection*. Para a invocação estática funcionar, o compilador Microsoft IDL (MIDL) cria a procuração e o código *stub,* quando roda no arquivo IDL. Eles são gravados no registro do sistema para permitir melhor flexibilidade do seu uso. Esse é o método fundamental (*vtable*) de invocação de objetos. Para a invocação dinâmica funcionar, objetos COM implementam uma interface chamada *IDispatch*. Assim como CORBA ou Java/RMI, para permitir invocação dinâmica é necessário existir algum caminho para descrever os métodos de objetos e seus parâmetros. *Type Libraries* são arquivos que descrevem o objeto, e

COM fornece interfaces, obtidas por meio da interface *IDispatch* para questionar uma *type library* do Objeto.

Tanto CORBA como Java/RMI sustentam herança múltipla em nível de IDL ou de interface. Uma diferença entre IDLs CORBA (e Java/RMI) e IDLs COM é que CORBA (e Java/RMI) podem especificar exceções nas IDLs, enquanto que DCOM não pode. Em CORBA, o compilador IDL gera tipo de informação (*type information*) para cada método em uma interface e a armazena no Repositório de Interface – *Interface Repository* (IR). Um cliente pode, dessa forma, questionar o IR para pegar informação em tempo de execução sobre uma interface particular e, então, usar aquela informação para criar e invocar um método no objeto servidor remoto CORBA dinamicamente, por meio da Interface de Invocação Dinâmica (DII). Similarmente, do lado do servidor, a Interface Reduzida Dinâmica (DSI) permite a um cliente invocar uma operação de um objeto servidor remoto CORBA, que não tem nenhum conhecimento em tempo de compilação do tipo de objeto que está implementando.

Java/RMI – Note que, ao contrário dos outros dois, Java/RMI usa um arquivo *.java* para definir sua interface remota. Essa interface vai garantir consistência de tipo entre o cliente Java/RMI e o objeto servidor Java/RMI. Todo objeto servidor remoto em Java/RMI tem que estender a classe *java.rmi.Remote*. Similarmente, qualquer método que possa ser remotamente invocado em Java/RMI projeta uma *java.rmi.RemoteException*, que é uma superclasse das muitas classes específicas de exceções RMI. Para invocar um método remoto, o cliente faz uma chamada para o cliente *proxy* (procurador). O cliente do lado *proxy* coloca os parâmetros da chamada dentro de uma mensagem de pedido e invoca um protocolo conectado (*wire*) como IIOP (em CORBA) ou ORPC (em DCOM) ou JRMP (em Java/RMI) para despachar a mensagem para o servidor. No lado servidor, o protocolo *wire* entrega a

mensagem para o lado servidor *stub*. O lado servidor *stub* então desfaz a mensagem e chama o método atual no objeto. Tanto em CORBA como em Java/RMI, o cliente *stub* é chamado de *stub* ou *proxy* e o servidor *stub* é chamado *skeleton*. Em DCOM, o cliente *stub* é referido como *proxy* e o servidor *stub* é referido como *stub*.

2.4.1.2. Objeto Cliente

Cliente DCOM – O cliente DCOM primeiramente cria um ponteiro para o objeto servidor. A palavra-chave *new* aqui instancia o objeto servidor DCOM. Isto leva o Microsoft JVM a usar o CLSID para fazer a chamada *CoCreateInstance ( ).*

Cliente CORBA – O cliente CORBA vai primeiramente ter que iniciar o CORBA ORB fazendo uma chamada para *ORB.init( )*. Ele então instancia um objeto servidor CORBA, unindo com uma referência remota ao objeto servidor. Tanto o Visibroker da Inprise quanto o Orbix da Iona, têm um método *bind( )* para unir e obter uma referência de objeto servidor.

Cliente Java/RMI – O cliente Java/RMI primeiro instala um administrador de proteção, antes de fazer qualquer chamada remota. Isto é feito por uma chamada para *System.setSecurityManager( )*. O *RMISecurityManager* provido pelo JavaSoft é uma tentativa do JavaSoft de escrever sua própria implementação. Entretanto, JavaSoft não força o uso do *RMISecurityManager* – pode-se escrever um próprio administrador de proteção e instalá-lo quando quiser.

2.4.1.3. Objeto Servidor

Servidor DCOM – Todas as classes, que são requeridas para Java/COM, são definidas no pacote *com.ms.com*. As classes e os métodos são declarados como *public* para que eles possam ser acessíveis de fora do pacote.

Também note que o CLSID é especificado e declarado como *private*. É usado pelo COM para instanciar o objeto por meio do *CoCreateInstance ( ),* quando o cliente DCOM faz um novo remotamente.

Servidor CORBA – Todas as classes que são requeridas para CORBA são definidas no pacote *org.omg.CORBA*. O objeto servidor CORBA estende uma classe *skeleton* gerada pelo nosso compilador IDL CORBA. As classes e os métodos são declarados como *public* para que eles possam ser acessíveis de fora do pacote.

Servidor Java/RMI – Todas as classes que são requeridas para Java/RMI são definidas no pacote ***java.rmi***. O objeto servidor Java/RMI mostrado estende a classe *UnicastRemoteObject,* que tem todos os métodos remotos definidos do Java/RMI.

2.4.1.4. Principais programas do servidor

Servidor Principal CORBA – A primeira coisa a ser feita pelo programa principal é iniciar o ORB CORBA usando *ORB.init( ).* Um *Object Adapter* (OA) está no topo do ORB e é responsável por conectar a implementação do objeto servidor CORBA com o ORB CORBA. Adaptadores de Objeto fornecem serviços como geração e interpretação das referências de objeto, invocação de método, ativação e desativação de objeto, e mapeamento de referências de objetos para implementações. É necessário iniciar *Basic Object Adapter* (BOA – Adaptador de Objeto Básico) ou o *Portable Object Adapter* (POA – Adaptador de Objeto Portável), dependendo do que o ORB sustenta.

Servidor Principal Java/RMI – O cliente Java/RMI terá primeiramente que instalar um administrador de segurança, antes de fazer qualquer chamada remota. Isso é feito com uma chamada para *System.setSecurityManager( ).* Então, o objeto servidor Java/RMI com o código a seguir permanece até parar de funcionar (desligar).

Servidor Principal DCOM – Não é fornecido um programa principal para a implementação do servidor DCOM. O apoio sobre Java no Internet Explorer "roda" como um servidor *in-process* (em processo) e servidores *in-process* não podem normalmente ser remotos, usando o Windows NT 4.0 COM Distribuído (DCOM).

2.4.1.5. Conclusão

As arquiteturas CORBA, DCOM e JAVA/RMI oferecem mecanismos para uma transparente invocação e acesso a objetos remotos distribuídos. Embora os mecanismos que eles empregam para conseguir possam ser diferentes, a aproximação que cada um deles leva é mais ou menos similar. As tabelas, a seguir (RAJ, 2004), demonstram que muitas características são similares ou se equivalem. Isso retrata que vários aspectos da comparação são iguais e, portanto, a escolha de uma plataforma para distribuição restringe muitas vezes a especificidade do projeto.

As Tabelas 2.1.(a), 2.1.(b), 2.1.(c), 2.1.(d) e 2.1.(e) apresenta as principais diferenças entre DCOM, CORBA e JAVA/RMI. Cabe novamente ressaltar que, muitas características são iguais.

Tabela 2.1.(a) Diferenças entre DCOM, CORBA e JAVA/RMI (RAJ, 2004).

Características	**Plataformas de Distribuição**		
	DCOM	**CORBA**	**JAVA/RMI**
Suporte a múltiplas heranças	Suporta múltiplas interfaces para objetos e usa o método *QueryInterface* ( ) para navegar entre interfaces. Isto significa que um Cliente *Proxy* carrega dinamicamente *stubs* de servidor múltiplo na camada remota dependendo do número de interfaces.	Suporta múltiplas heranças no nível de interface (não é necessário recompilar um projeto para acrescentar alguma funcionalidade).	Suporta múltiplas heranças no nível de interface.

Tabela 2.1.(b) Diferenças entre DCOM, CORBA e JAVA/RMI (RAJ, 2004).

Características	**Plataformas de Distribuição**		
	DCOM	**CORBA**	**JAVA/RMI**
Objeto Global	Todo objeto implementa *IUnknown.*	Toda interface é herdada de *CORBA.Object.*	Toda objeto servidor implementa *java.rmi.Remote*
Localizador de objetos	Exclusivamente identifica um objeto servidor remoto por meio de seu ponteiro de interface, que serve como alça do objeto em tempo de execução.	Exclusivamente identifica objetos servidores remotos por meio de referências de objeto (*objref*), que servem como alça do objeto em tempo de execução.	Exclusivamente identifica objetos servidores remotos com o *ObjID*, que serve como alça do objeto em tempo de execução.
Localizador de Interfaces	Exclusivamente identifica uma interface usando o conceito de Interface IDs (IID) e identifica uma implementação nomeada do objeto servidor usando o conceito de Classe IDs (CLSID), o mapa do qual é achado no registro.	Exclusivamente identifica uma interface usando o nome de interface e identifica uma implementação nomeada do objeto de servidor pelo seu mapeamento para um nome no Repositório de Implementação (*Implementation Repository*).	Exclusivamente identifica uma interface usando o nome de interface e identifica uma implementação nomeada do objeto servidor pelo seu mapa para uma URL no Registro.
Referência ao Objeto Servidor	A geração da referência do objeto servidor remoto é executada no protocolo *wire* pelo Exportador de Objeto.	A geração da referência do objeto servidor remoto é executada no protocolo *wire* pelo Exportador de Objeto.	A geração da referência do objeto servidor remoto é executada pela chamada ao método *UnicastRemoteObject. ExportObject*.
Registro de Objetos	Tarefas como registro de objeto, *skeleton instantiation* etc, são executadas explicitamente pelo programa servidor ou dinamicamente dirigidas pelo sistema COM em tempo de execução.	O construtor implicitamente executa tarefas comuns como registro de objeto, *skeleton instantiation,* etc.	O RMIRegistry executa tarefas comuns como registro de objeto por meio da classe *Naming*.

Tabela 2.1.(c) Diferenças entre DCOM, CORBA e JAVA/RMI (RAJ, 2004).

Características	**Plataformas de Distribuição**		
	DCOM	**CORBA**	**JAVA/RMI**
Protocolo Remoto	Usa a *Object Remote Procedure Call* (ORPC) como seu protocolo remoto latente.	Usa o Protocolo Internet Inter-ORB (IIOP) como seu protocolo remoto latente.	Usa Java *Remote Method Protocol (JRMP)* como seu protocolo remoto latente.
Requisição a um Servidor	Quando um objeto cliente precisa ativar um objeto servidor, ele pode fazer um *CoCreateInstance( )* - (Nota: Há outros modos que o cliente pode adquirir o ponteiro de interface de um servidor).	Quando um objeto cliente precisa ativar um objeto servidor, este usa um *naming* ou um serviço *trader*. (Nota: Há outros modos que o cliente pode adquirir uma referência de servidor).	Quando um objeto cliente precisa de uma referência de objeto servidor, tem que fazer um *lookup( )* no nome de URL do objeto servidor remoto.
Referência do Objeto	A alça de objeto que o cliente usa é o ponteiro da interface.	A alça de objeto que o cliente usa é a Referência de Objeto.	A alça de objeto que o cliente usa é a Referência de Objeto.
Nome do Objeto	O mapeamento do nome do objeto na sua implementação é dirigido pelo Registro.	O mapeamento do nome do objeto na sua implementação é dirigido pelo Repositório de Implementação	O mapeamento do nome do objeto na sua implementação é dirigido pelo *RMIRegistry*.
Tipo de informação	O tipo de informação para métodos é assegurado na Biblioteca de Tipo (*Type Library*).	O tipo de informação para métodos é assegurado no Repositório de Interface.	Qualquer tipo de informação é assegurado pelo próprio objeto.
Localizador de implementação	A responsabilidade de localizar uma implementação de objeto cai sobre o *Service Control Manager* (SCM).	A responsabilidade de localizar uma implementação de objeto cai sobre o *Object Request Broker* (ORB).	A responsabilidade de localizar uma implementação de objeto cai sobre *Java Virtual Machine (JVM).*

Tabela 2.1.(d) Diferenças entre DCOM, CORBA e JAVA/RMI (RAJ, 2004).

Características	**Plataformas de Distribuição**		
	DCOM	**CORBA**	**JAVA/RMI**
Ativação de implementação	A responsabilidade de ativar uma implementação de objeto cai sobre o *Service Control Manager* (SCM).	O responsável de localizar uma implementação de objeto cai sobre o Adaptador de Objetos.	A responsabilidade de ativar uma implementação de objeto cai sobre *Java Virtual Machine* (JVM).
Stub cliente	O lado *stub* do cliente é chamado de *Proxy*.	O lado *stub* do cliente é chamado de *Proxy* ou *stub*.	O lado *stub* do cliente é chamado de *Proxy* ou *stub*.
Stub servidor	O lado *stub* do servidor é chamado de *stub*.	O lado *stub* do servidor é chamado de *skeleton*.	O lado *stub* do servidor é chamado de *skeleton*.
Passagem de parâmetros	Todos os parâmetros que passam entre os objetos cliente e servidor são definidos no arquivo de Definição de Interface. Daí, dependendo da IDL específica, parâmetros são passados ou por valor ou por referência.	Quando passam parâmetros entre o cliente e o objeto servidor remoto, todos os tipos de interface são passados por referência. Todos os outros objetos são passados por valor incluindo tipos de dados altamente complexos.	Quando passam parâmetros entre o cliente e o objeto servidor remoto, são passados por referência remota. Todos os outros objetos são passados por valor.
Coleta de lixo	Tentativas para executar a coleta do lixo distribuído no *wire* através de *pinging*. O protocolo *wire* DCOM usa um mecanismo PINGING para referências de coleta de lixo de objeto servidor remoto.	Não tenta executar a coleta de lixo distribuído como objetivo geral.	Tentativas para executar coleta de lixo distribuído dos objetos servidores remotos usando os mecanismos colocados no JVM.
Tipos complexos	Permite-lhe definir estruturas arbitrariamente complexas, uniões descriminadas e coleções *conformant* na IDL	Tipos complexos que atravessam as divisas de interface devem ser declarados na IDL.	Qualquer objeto Java *Serializable* pode ser passado como um parâmetro através de processos.

Tabela 2.1.(e) Diferenças entre DCOM, CORBA e JAVA/RMI (RAJ, 2004).

Características	**Plataformas de Distribuição**		
	DCOM	**CORBA**	**JAVA/RMI**
Portabilidade	Roda em qualquer plataforma contanto que haja uma implementação de Serviço COM para aquela plataforma (como o software EntireX da AG).	Roda em qualquer plataforma contanto que haja uma implementação de ORB CORBA para aquela plataforma (como o VisiBroker da Inprise).	Roda em qualquer plataforma contanto que haja uma implementação de *Java Virtual Machine* para aquela plataforma (fornecido por companhias como a JavaSoft e Microsoft)
Linguagens de Programação	Já que a especificação é em nível binário, diversas linguagens de programação como C++, Java, Object Pascal (Delphi), Visual Basic e até COBOL podem ser usadas para codificar estes objetos.	Já que esta é somente uma especificação, diversas linguagens de programação podem ser usadas para codificar estes objetos contanto que haja bibliotecas ORB para codificar naquela linguagem.	Já que depende fortemente da Java Object Serialization, estes objetos só podem ser codificados na linguagem Java.
Tratamento de exceção	Cada chamada de método retorna uma estrutura bem definida de tipo HRESULT, cujos locais de *bit* codificam o status de retorno. Para uma exceção mais complexa usa-se o *Error Objects*, e o objeto servidor tem que implementar a interface *ISupportErrorInfo.*	Manipulação de exceção é manipulada através de Objetos de Exceção. Quando um objeto distribuído lança um objeto de exceção, o ORB transparentemente o serializa e o reúne pelo *wire.*	Permite lançar exceções que são então serializadas e reunidas pelo *wire.*

A Tabela 2.2 mostra diferenças entres as plataformas, de acordo com os pontos mais relevantes numa avaliação de plataformas de distribuição, incluindo especificamente a tecnologia *.NET* que incorporou DCOM.

Tabela 2.2. Comparação entre plataformas de distribuição.

Características	Plataformas de Distribuição		
	.NET	**CORBA**	**JAVA/RMI**
Linguagens	De Cobol a JAVA, são mais de 20 linguagens no mesmo ambiente com integração total entre as linguagens.	Diversas linguagens de programação podem ser usadas para codificar objetos CORBA, desde que haja bibliotecas ORB.	Depende fortemente da *Java Object Serialization*, portanto estes objetos só podem ser codificados na linguagem Java.
Protocolo Remoto/ Padrões	Usa o XML como chave para a programação. As aplicações acessam os serviços XML via protocolos padronizados e dados em formatos como HTTP, XML e SOAP.	Faz uso do IIOP Inter ORB como protocolo remoto básico.	Usa o JRMP *Java Remote Method Protocol* como protocolo remoto básico.
Compilação	Todo código executado no ambiente .NET é compilado.	No código criado em CORBA é feita a geração de códigos IDL que é compilada na linguagem escolhida para implementação.	O ambiente Java foi projetado para executar código interpretado.
Performance	Desempenho melhor para ambientes distribuídos em Internet.	Desempenho pode ser baixo à medida que você aumentar os acessos.	Desempenho pode ser baixo à medida que você aumentar os acessos.
Ativação de Implementação	A responsabilidade de ativação de uma implementação recai sobre o *FrameWork.*	A responsabilidade da ativação de uma implementação de um objeto recai sobre o Object Adapter (AO) – da mesma maneira sobre o Basic Object Adapter (BOA) ou Portable Object Adapter (POA).	A responsabilidade da ativação de uma implementação de um objeto é da Máquina virtual JAVA (JVM).
Coleta de Lixo	Não realiza a coleta de lixo distribuída.	Não realiza a coleta de lixo distribuída.	Usa a Máquina Virtual Java para coleta de lixo.
Aplicações	Com .NET é possível criar aplicações para vários tipos de dispositivos (PCs, PocketPCs, Celulares, PDAs, etc).	Suporta ambas as plataformas PC e não-PC.	Suporta vários dispositivos (PCs, PDAs, etc).

2.5. Escolha da arquitetura

De acordo com a comparação no item anterior e outros estudos comparativos, não existem grandes diferenças funcionais entre as plataformas estudadas. Em um estudo comparativo, Siqueira (2005) ratifica a pequena diferença entre plataformas de distribuição, como mostra a Figura 2.1.

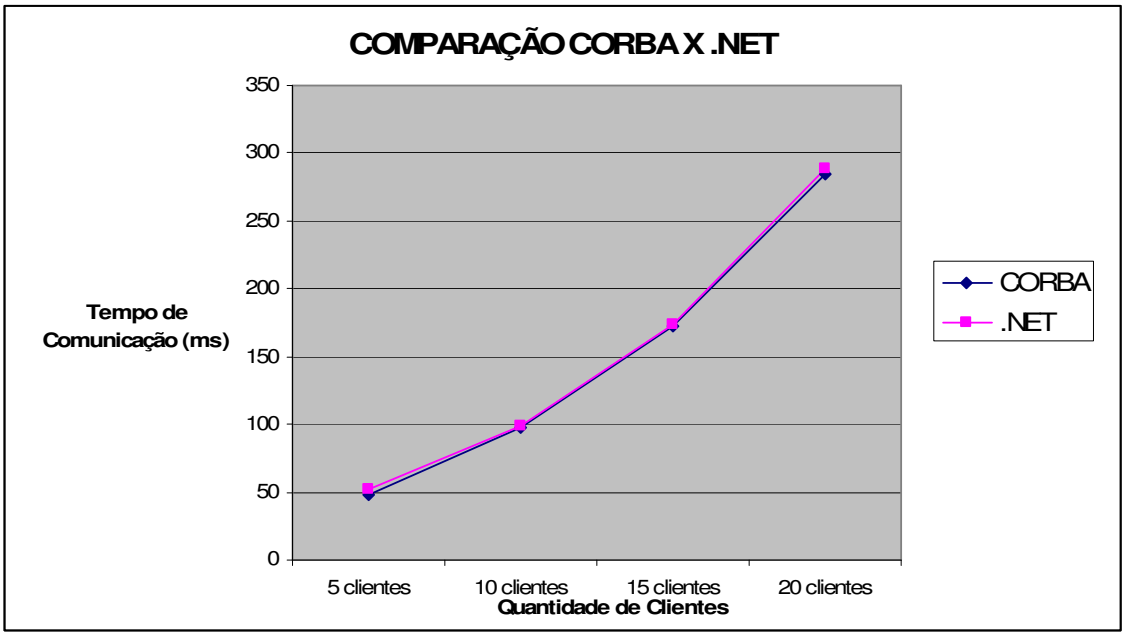


Figura 2.1. Latência dos protótipos (SIQUEIRA, 2005).

De acordo com a Figura 2.1, foram comparados o desempenho de dois protótipos, sendo cada um implementado em uma plataforma de distribuição, mostrando a proximidade dos resultados obtidos. Siqueira (2005) sugere, porém que, com o aumento de clientes (escalabilidade), a arquitetura CORBA se porte melhor. Entre CORBA, JAVA/RMI e DCOM/.NET, além da análise de outros estudos (DERIGGI JR, 1997; DERIGGI JR, 1999), CORBA foi escolhida pelos motivos a seguir:

a) É independente de plataforma.

b) Pode ser implementada na maioria das linguagens existentes.

c) Possui uma implementação menos complexa e com quantidade de linhas de código também inferior às demais.

d) Possui várias implementações, com destaque ao *Visibroker 4.0,* que foi usado neste trabalho por estar presente no ambiente *Delphi 6.0 Enterprise*.

A Tabela 2.3 mostra as principais diferenças entre as plataformas estudadas, ressaltando a escolha por CORBA.

Tabela 2.3. Comparação entre plataformas de distribuição.

CARACTERÍSTICAS	**PLATAFORMAS**		
	CORBA	**DCOM**	**JAVA/RMI**
Independência de plataforma	***SIM***	***NÃO***	***SIM***
Implementação em diversas linguagens	***SIM***	***SIM***	***NÃO***
Implementações diferentes	***SIM***	***NÃO***	***NÃO***

2.5.1. Objetos Distribuídos e Processamento Distribuído

Há alguns termos que são usados, quando se está trabalhando com objetos distribuídos. Portanto, o esclarecimento destes se torna necessário, devido à sua importância.

As *requisições* são pedidos que clientes fazem a um servidor para que um método ou serviço de um objeto seja executado. Uma *referência* indica um objeto particular no sistema e é utilizada nas requisições para identificar um objeto que execute o serviço.

Stub cliente e ***Stub*** servidor é a operação que a chamada remota a procedimentos faz, em empacotar (*marshal*) e desempacotar (*unmarshal*) os parâmetros (mensagens) entre cliente e servidor (RICCIONI, 2000).

O processamento distribuído prevê o compartilhamento de aplicações e funcionalidade, do mesmo modo que os dados são compartilhados. Porém, ao invés de se ter

aplicações monolíticas[7], as aplicações locais em um ambiente de processamento distribuído exercem o papel de controladoras: coordenam as atividades das funções que produzem não somente dados, mas também o código para manipulá-los. O código e os dados são produzidos de forma que possam ser acessados por múltiplos processadores em um ambiente heterogêneo, sem se preocupar com sua localização física.

Assim, têm-se como requisitos essenciais para um ambiente de computação distribuída (SEMENTILLE, 1999):

a) A aplicação deve ser capaz de estabelecer as capacidades de processamento que ela requer.

b) A aplicação deve ser capaz de enviar parâmetros e dados para o processo, bem como receber resultados deste.

c) Os parâmetros e resultados dos processos devem ser expressivos para as várias arquiteturas de máquinas dentro do ambiente.

d) O processo deve ser capaz de executar em diferentes implementações de ambientes e linguagens.

Cabe notar que existem ainda outros requisitos, tais como segurança, descrição de interface, como a informação é transmitida, etc.

Comparando-se a idéia dos objetos distribuídos com as idéias básicas do processamento distribuído, nota-se que os primeiros consistem em uma tecnologia disponível para os sistemas distribuídos, da mesma forma que tem sido para o modelo cliente/servidor a multimídia, o processamento de documento e outras aplicações. No caso dos sistemas distribuídos, preocupações como nomeação, conversão de espaço de endereçamento,

[7] Aplicações monolíticas são totalmente centralizadas. Modelo estações "burras" – mainframes.

protocolos de transporte e descrição de interface são maiores. Cada uma destas áreas é complexa e os objetos podem auxiliar a abstrair e lidar com esta complexidade.

Um aspecto que causa confusão é a diferença entre as tecnologias de componentes ao nível de aplicação e os modelos de objetos que a suportam. No nível mais baixo, existem os modelos de objetos tais como o *System Object Model* (SOM/DSOM) da IBM e o *Component Object Model* (COM) da Microsoft (RICCIONI, 2000). Estas tecnologias em nível de sistema são basicamente endereçadas para resolver o problema de acoplamento binário entre uma aplicação e os objetos que a utilizam. Pode-se tomar como exemplo uma aplicação escrita na linguagem C++, que usa diversas classes, cujos métodos estão implementados em Biblioteca de Ligação Dinâmicas (DLL – *Dynamic Linking Language*). Convenientemente, uma DLL não faz parte do código da aplicação e então pode ser atualizada ou alterada, sem afetar a aplicação, uma vez que sua interface permanece a mesma. Infelizmente, esta idéia falha se as mudanças são feitas em uma classe baseada em DLL, de forma a alterar o tamanho do objeto. Neste caso, a aplicação que executa a chamada pode necessitar ser recompilada, mesmo se seu código fonte não foi mudado textualmente.

Para servir de ligação binária entre as implementações do cliente e do objeto servidor, um modelo de objeto é definido em um nível de abstração, que assume que a linguagem do modelo do objeto e sua tradução sejam transparentes. O modelo é, geralmente, expresso em termos de uma Linguagem de Definição de Interface (IDL), a qual é processada de forma independente da linguagem de implementação. Uma IDL é o caminho para se definir uma interface para um serviço; freqüentemente, o mecanismo gera um código *stub* (procedimento local), que pode ser chamado pela aplicação. Na implementação final, ela descreve a interface para o código, que executa a função e gera *stubs* que podem ser usados para executar as operações (LEAVENS, 1993). O mecanismo do RPC (*Remote Procedure*

Call – Chamadas de Procedimentos Remotos) (NELSON, 1981) pode preencher a lacuna entre os dois. Desde que a IDL possa ser traduzida para a linguagem de implementação, esta abordagem preenche o requisito de independência de linguagem: os *stubs* podem ser gerados em qualquer linguagem para a qual a IDL tenha mapeamento.

A principal diferença entre as definições de interfaces para os mecanismos procedimentais e para os modelos de objetos é que no modelo de objetos a interface é parte de uma construção semântica que representa o objeto. Dependendo do modelo específico, esta construção pode possuir algumas ou todas as características e vantagens esperadas dos objetos, incluindo o encapsulamento, herança e polimorfismo (SEMENTILLE, 2000).

2.5.2. Objetos Interoperáveis

A interoperabilidade pode ser definida como sendo a habilidade de trocar funcionalidade e dados interpretáveis entre duas entidades de software. Pode ser definida em termos de quatro requisitos: comunicação, geração de pedidos, formato de dados e semântica. As entidades de software requerem um canal de comunicação com um protocolo de comunicação comum. Através deste canal, as entidades necessitam ser capazes de formular e transmitir um pedido interpretável para funções ou dados. O resultado do pedido deve retornar para a entidade solicitante. O intercâmbio de dados também implica em um requisito para um formato de dados que possa ser analisado pelas entidades. As entidades devem entender o pedido por meio de alguma forma de tradução semântica.

A principal idéia por trás dos objetos interoperáveis é transpor os limites existentes. Nos modelos de programação orientada a objetos atuais, existe um forte acoplamento binário entre uma aplicação e as classes de objetos que ela usa.

A primeira fronteira a ser transposta é a do espaço de endereçamento. Um "modelo de objeto interprocesso" permite a um processo em um espaço de endereçamento pedir serviço de um objeto em outro, ou dois objetos compartilharem um objeto em um terceiro espaço de endereçamento.

A próxima fronteira é a máquina. Qualquer modelo de objeto interprocesso deve ser capaz de traduzir os dados associados com os pedidos entre modelos de memória. Deve ainda ser capaz de localizar o objeto servidor, estabelecer comunicação com ele, empacotar os pedidos e parâmetros e enviá-los, esperar pelos resultados, desempacotá-los e traduzi-los e devolvê-los à aplicação. Somente as tecnologias, que atravessarem a barreira entre o modelo interprocesso/interprocessador, podem ser chamadas de "tecnologias de objetos distribuídos".

As outras duas fronteiras são: linguagem de programação e o sistema operacional. Estas fronteiras, por sua vez, não têm muito haver com o fato de uma tecnologia ser distribuída ou não.

O objetivo final a ser alcançado é um modelo que permita que vários objetos, ser escritos em qualquer linguagem, possam ser compartilhados por aplicações escritas em outra linguagem, executando em qualquer máquina de uma rede e sobre qualquer sistema operacional (BETZ, 1994; SCHMIDT, 1995).

2.5.3. ORB – *Object Request Broker*

Um ORB é um mecanismo básico pelo qual objetos transparentemente fazem requisições para – e recebem respostas de – cada outro objeto na mesma máquina ou através de uma rede. O cliente não precisa estar ciente do mecanismo usado para comunicar ou ativar um objeto, ou como o objeto é implementado, nem onde o objeto está localizado. Assim, o ORB forma a base para a construção de aplicações com objetos distribuídos e para a

interoperabilidade entre aplicações em ambientes homogêneos e heterogêneos. (PAULOVICH, 2000).

O ORB é responsável pela localização de um objeto ao qual se destina uma requisição, assim como o encaminhamento dos parâmetros dessa requisição em um formato que este objeto aceite. Se houver parâmetros de saída da requisição para o cliente, o retorno de tais parâmetros também é função do ORB.

Um objeto CORBA interage de várias maneiras: com o ORB; por meio da interface ORB ou por um *Object Adapter* (Adaptador de Objeto)[8] – ou um *Basic Object Adapter* (BOA); por um *Portable Object Adapter* (POA).

A Figura 2.2 mostra uma requisição de um cliente, enviada através do ORB, a uma implementação de objeto. O cliente é qualquer objeto que solicita um serviço e a implementação do objeto contêm o código e os dados que caracterizam o comportamento de um objeto. O ORB é responsável por todos os mecanismos necessários para achar a implementação de objeto de um pedido, por preparar a implementação de objeto para receber um pedido e pela comunicação.

[8] Um adaptador de objetos possibilita a comunicação dinâmica de objetos, mesmo quando o objeto não foi implementado no mesmo momento da construção da aplicação.

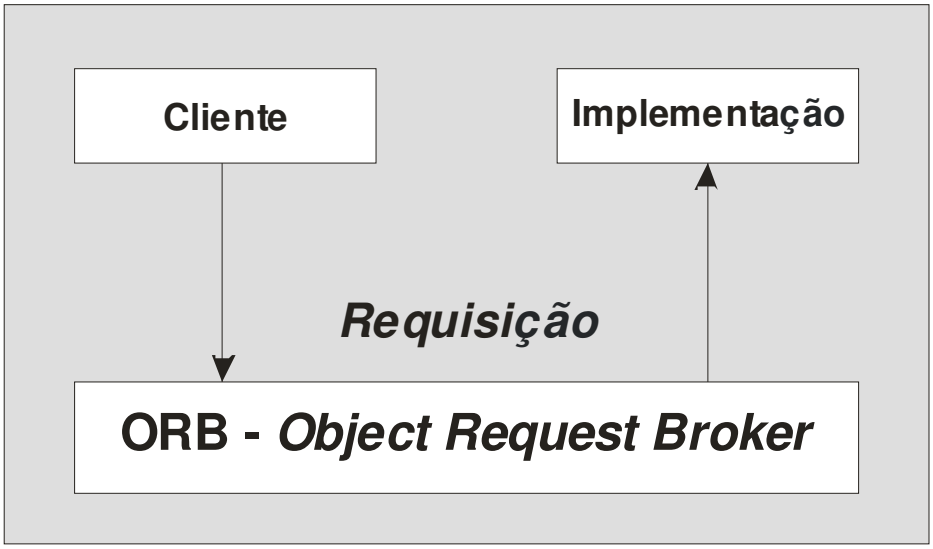


Figura 2.2. Requisição por meio do ORB.

A interface, com a qual o cliente tem contato, é totalmente transparente e independente de qualquer fator relacionado à heterogeneidade do ambiente distribuído no qual se encontra, não importando onde o objeto está localizado, em qual linguagem de programação foi implementado ou a ordenação de bytes da máquina na qual se executa o objeto. Assim, qualquer software que ofereça as interfaces e os serviços especificados pode ser considerado um ORB.

Entretanto, segundo Riccioni (2000), ORBs diferentes podem apresentar características de implementação diferentes, resultando em serviços prestados a objetos e clientes com qualidades e propriedades diferentes.

2.5.4. O Cliente ORB

A arquitetura CORBA permite interfaces estáticas e dinâmicas. Os *stubs* IDL do cliente fornecem as interfaces estáticas para os serviços dos objetos. Estes *stubs* pré-compilados, gerados pelo compilador IDL, definem como os clientes invocam os serviços correspondentes nos servidores.

Com as APIs de invocação dinâmicas, por outro lado, é possível (em tempo de execução) descobrir um serviço que se quer invocar, obter sua definição, emitir uma chamada parametrizada a este e receber uma resposta dele.

Um repositório de interfaces APIs permite ao cliente obter e modificar as descrições de todos os componentes de interface armazenados, os métodos que eles suportam, e os parâmetros que eles requerem. O repositório de interface armazena, atualiza, e gerencia as definições de interface de objetos, e seus programas usam suas APIs para acessar e atualizar esta informação.

Os ORBs fornecem identificadores globais chamados "IDs de repositório", que única e globalmente identificam um componente e sua interface por meio de múltiplos ORBs e repositórios. Os IDs de repositório são cadeias de caracteres (*strings*) únicas geradas pelo sistema que forçam uma consistência de nomes, por meio dos repositórios – nenhuma colisão de nomes é permitida. Eles são gerados, por meio do *Distributed Computing Evironment/Unique User Identifier* (DCE/UUID), ou um prefixo único fornecido pelo usuário acoplado aos nomes estabelecidos pela IDL.

A interface ORB, usada de forma parecida pelo cliente e servidor, oferece uma variedade de APIs úteis. Por exemplo, existem APIs que convertem uma *string* em uma referência a um objeto. Estas chamadas são convenientes no caso de armazenamento ou comunicação de referências de objetos.

Com o suporte para invocações cliente/servidor estáticas e dinâmicas e um repositório de interfaces, o CORBA é mais poderoso e flexível do que a RPC (*Remote Procedure Call*).

As invocações estáticas são fáceis de programar, mais rápidas e autodocumentadas. As invocações dinâmicas, apesar de mais difíceis de programar, oferecem um máximo de

flexibilidade e são essenciais, quando as aplicações devem descobrir serviços em tempo de execução (MAFFEIS, 1997; SCHMIDT, 1997).

2.5.5. O servidor ORB

Os servidores não podem diferenciar as invocações estáticas das dinâmicas. A mesma semântica de mensagem é aplicada em ambos os casos. O ORB localiza um adaptador de objeto, transmite os parâmetros e transfere o controle para a implementação do objeto, através do *stub* IDL do servidor, chamado de "esqueleto" (*skeleton*).

Os esqueletos estáticos oferecem interfaces para cada serviço exportado pelo servidor. Estes *stubs*, como aqueles clientes, são criados usando um compilador IDL. Um esqueleto estático fornece um suporte rígido para os métodos definidos em IDL de uma classe de objeto particular.

As interfaces de esqueletos dinâmicos oferecem um mecanismo de ligação (*binding*) em tempo de execução para os servidores (VINOSKI, 1997). O esqueleto dinâmico inspeciona os parâmetros de uma mensagem recebida para determinar um objeto alvo e um método. Esta técnica é útil para construir pontes entre os ORBs.

O adaptador de objetos situa-se acima dos serviços de comunicação do núcleo do ORB, aceitando pedidos para serviços provenientes dos objetos servidores. Ele fornece o ambiente em tempo de execução para o instanciamento de objetos servidores, passando pedidos e associando IDs de objetos (referência aos objetos). O adaptador de objeto também registra as classes por ele suportadas e suas instâncias de tempo de execução (isto é, objetos) com o repositório de implementação. A arquitetura CORBA especifica que cada ORB deve suportar um adaptador padrão de objetos.

O repositório de implementações lista as classes que um servidor suporta, os objetos por ele instanciados e seus IDs. Ele também é um lugar comum para armazenamento de informação de rastreamento, rastros de auditoria, segurança e outros dados administrativos.

2.5.6. Arquitetura OMA

OMA (*Object Management Arquitecture*) significa Arquitetura de Gerenciamento de Objetos, responsável pelo desenvolvimento das aplicações. Neste caso, o padrão CORBA tem a responsabilidade de implementar a função de localização de objetos solicitados. Por meio do *Object Management Guide* (OMAG), é que são descritos os objetivos técnicos e a terminologia da arquitetura OMA. Neste guia, estão incluídos o Modelo de Referência OMA e o Modelo de Objetos OMG.

O Modelo de Objetos define a semântica comum para a especificação das características externas visíveis dos objetos, e o Modelo de Referência identifica e caracteriza os componentes, interfaces e protocolos que compõem o OMA. Este último está sendo preenchido com especificações detalhadas para cada componente e categoria de interface. Estas especificações incluem: CORBA, CORBAfacilities (Facilidades CORBA) e CORBAservices (Serviços CORBA).

Segundo Riccioni (2000), os principais componentes do OMA são:

- **CORBA e ORB –** Que manipulam requisições entre objetos.
- **Serviços CORBA –** Que definem serviços em nível de sistema que ajudam a gerenciar e manter objetos.
- **Facilidades CORBA –** Que definem facilidades e interfaces no nível de aplicação.

- **Objetos de Aplicações –** São os objetos propriamente ditos.

- **IDL –** A linguagem de definição de interface do CORBA.

O ORB é o componente mais importante da arquitetura OMA. Ele permite que objetos se comuniquem transparentemente em um ambiente distribuído e heterogêneo.

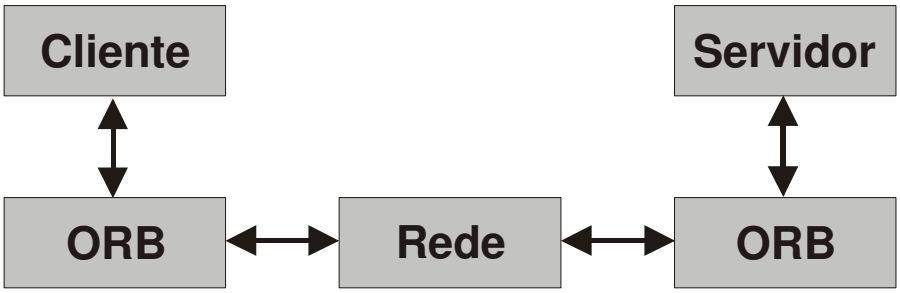


Figura 2.3. Arquitetura CORBA.

A Figura 2.3 mostra o ORB – *Object Request Broker* como elemento principal, pois tem a função de localizar objetos instanciados no servidor, estabelecer conexão com esses objetos, passar os parâmetros necessários e receber os valores resultantes da requisição.

A arquitetura de uma ORB é definida pelas suas interfaces. Qualquer implementação que satisfaça a interface apropriada é aceitável (NEVES JÚNIOR, 2000).

Existem quatro categorias de interfaces de objetos:

- Serviços de Objetos (*Object Services*) são interfaces para serviços gerais, que são provavelmente usados em qualquer programa baseado em objetos distribuídos;

- Recursos Comuns (*Common Facilities*) são interfaces para facilidades horizontais orientadas para usuários finais, aplicáveis à maioria dos domínios de aplicação;

- Interfaces de Domínio (*Domain Interfaces*) são interfaces de aplicações de um domínio específico;

- Interfaces de Aplicação (*Application Interfaces*) são interfaces não padronizadas para aplicações específicas.

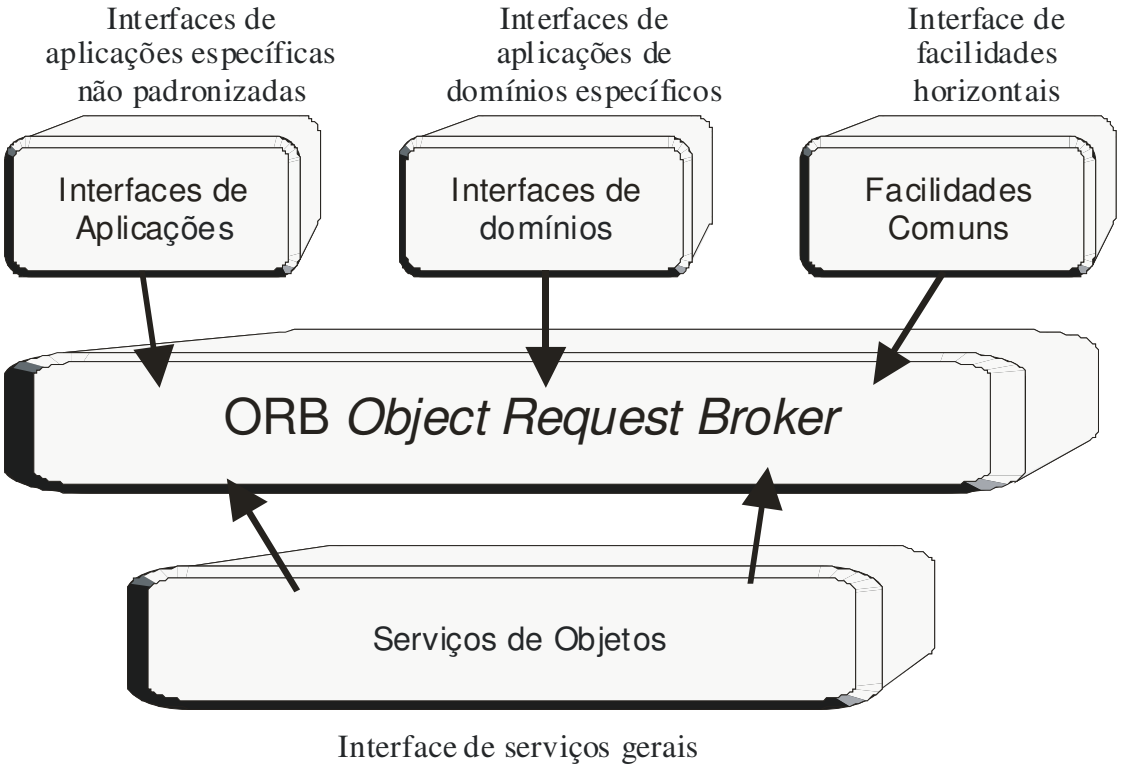


Figura 2.4. Modelo de Referência (OMG, 2004) .

2.6. Implementações CORBA

Existem muitas implementações CORBA disponíveis no mercado (TEIXEIRA, 2002). Cada implementação segue a especificação CORBA, porém, em cada uma dessas implementações, alguns pequenos detalhes diferem da especificação original. Essas alterações podem facilitar o uso da implementação, por agregar facilidades que não estão presentes na especificação original.

2.6.1. O *Inter-Language Unification* (ILU)

O ILU pode ser caracterizado como um sistema de *interfaces* de objetos multi-linguagem. As *interfaces* de objetos fornecidas pelo ILU permitem que as diferenças entre implementações desenvolvidas, em diferentes linguagens e diferentes sistemas operacionais, sejam escondidas. Várias são as utilidades do ILU, podendo ser utilizado, entre outras coisas,

na construção de sistemas distribuídos e bibliotecas multi-linguais orientadas a objetos (JANSSEN, 2005).

2.6.2. ORBIX da IONA

O ORBIX é implementado como um par de bibliotecas – uma para aplicações clientes, e outra para servidores – e um *daemon* de ativação (*orbixd*). O *orbixd* precisa estar presente apenas nos nós que atuam como servidores CORBA, e sua função é redisparar processos servidores dinamicamente, quando solicitado. Um cliente não distribuído e aplicações servidoras em um mesmo espaço de endereçamento, podem ser construídos utilizando-se apenas a biblioteca para o servidor (IONA, 1997 apud SEMENTILLE, 1999 p.84).

2.6.3. Visibroker

O produto CORBA utilizado neste estudo é chamado VisiBroker, um "ORB da Borland, e um dos ORBs mais usados no mundo" (TEIXEIRA, 2002). Ele segue todas as especificações CORBA, com exceção do serviço de nomeação ou localização de objetos. Além de ser o ORB mais popular, a escolha por esta implementação deve-se ao fato de estar presente no ambiente desenvolvido escolhido para este trabalho.

2.6.3.1. *OSAgent*

Objetos CORBA precisam de um meio para localizar um ao outro. O OMG oferece uma solução para isso com o *Naming Service* (serviço de nomeação), que é descrito na especificação CORBA. O *Naming Service* é um programa executado em algum lugar na rede. Os objetos no servidor se registram com o *Naming Service,* de modo que as aplicações cliente tenham um meio de localizar objetos específicos. O *Naming Service* exige um código

adicional, tanto no cliente quanto no servidor. O local do processo precisa ser conhecido previamente, antes da solicitação de uma conexão com um objeto servidor.

Esse é um modo bem complicado de localização de servidores. O Visibroker possui um utilitário, que torna a localização do objeto muito mais fácil que usar o *Naming Service*. Esse programa é o *OSAgent* (TEIXEIRA, 2002). Ele não faz parte da especificação CORBA. *OSAgent* é um utilitário próprio, que só está disponível nesse ORB. Desde que o ORB Visibroker seja usado dentro da implementação CORBA, o *OSAgent* é o método preferido para a localização e conexão a objetos (TEIXEIRA, 2002).

2.7. Considerações Finais

Este capítulo apresentou as principais características dos Sistemas Distribuídos, relatando que a distribuição pode acontecer em nível de *software* como *hardware* e nos dois ao mesmo tempo. Apresentou também, arquiteturas de distribuição existentes no mercado e suas principais diferenças, ressaltando a escolha de CORBA como padrão para objetos distribuídos usado neste trabalho.

No próximo capítulo, apresenta-se a caracterização de ambientes virtuais.

CAPÍTULO III

3. AMBIENTES VIRTUAIS

3.1. Introdução

Para alcançar uma melhor compreensão dos termos “Ambiente Virtual” e “Ambiente Virtual Distribuído”, apresenta-se neste capítulo uma visão geral dos dois tipos de ambientes, objetos de estudo deste trabalho.

Ao final do capítulo, por meio de componentes, requisitos e características, são apresentados os AVDs avaliados neste trabalho.

3.2. Realidade Virtual

Como mencionado anteriormente, Realidade Virtual (RV) é uma tecnologia que vem sendo desenvolvida nos últimos anos com um grande crescimento e uma enorme expectativa. Pode ser definida como sendo a forma mais avançada de interface do usuário de computador até agora disponível (BYRNE, 2005), 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 pode possibilitar a criação/simulação de mundos reais ou imaginários na tela do computador, com aplicação em diversas áreas, assumindo um papel de relevo cada vez maior em campos específicos da vida econômica, social e cultural de muitos países (CAMACHO, 2005).

Uma outra definição um pouco mais refinada de Realidade Virtual é a seguinte: " uma forma das pessoas visualizarem, manipularem e interagirem com computadores e dados extremamente complexos" (AUKSTAKALNIS, 1992). Agrupando algumas outras definições de Realidade Virtual (CARDOSO, 2002; KIRNER, 2005; ANTUNES, 2005), pode-se dizer que Realidade Virtual é uma técnica avançada de interface, em que o usuário pode realizar imersão, navegação e interação em um ambiente sintético tridimensional gerado por computador, utilizando canais multi-sensoriais, funcionando em tempo real.

A idéia de imersão está ligada com o sentimento de se estar dentro do ambiente. Normalmente, um sistema imersivo é obtido com o uso de capacete de visualização, mas existem também sistemas imersivos baseados em salas com projeções das visões nas paredes, teto e piso (CRUZ-NEIRA, 1992). Além do fator visual, os dispositivos ligados com os outros sentidos também são importantes para o sentimento de imersão, como som, posicionamento automático da pessoa e dos movimentos da cabeça, controles reativos, etc. A visualização tridimensional pelo monitor é considerada de imersão subjetiva (ou não-imersiva para alguns).

A idéia de interação está ligada com a capacidade do computador detectar as entradas do usuário e modificar instantaneamente o mundo virtual e as ações sobre ele (capacidade reativa) (KIRNER, 2005).

A idéia de envolvimento está ligada com o grau de motivação para o engajamento de uma pessoa com determinada atividade. O envolvimento pode ser passivo, como ler um livro ou assistir televisão, ou ativo, ao participar de um jogo com algum parceiro (KIRNER, 2005). A Realidade Virtual tem potencial para os dois tipos de envolvimento, ao permitir a exploração de um ambiente virtual e ao propiciar a interação do usuário com um mundo virtual dinâmico.

Do ponto de vista da visualização, a realidade virtual imersiva é baseada no uso de equipamentos que geram realismos multisensoriais, enquanto a realidade virtual não imersiva baseia-se no uso de monitores. De qualquer maneira, os dispositivos baseados nos outros sentidos acabam dando algum grau de imersão à realidade virtual com o uso de monitores, mantendo sua caracterização e importância (ANTUNES, 2005).

Embora a realidade virtual com o uso de capacetes tenha evoluído e seja considerada típica, a realidade virtual com monitor apresenta ainda assim alguns pontos positivos como:

- utilizar plenamente todas as vantagens da evolução da indústria de computadores;
- evitar as limitações técnicas e problemas decorrentes do uso de capacete;
- e facilidade de uso.

Em alguns casos, como visualização, por exemplo, a realidade virtual com monitor é aceitável, mas com a evolução da tecnologia de realidade virtual a tendência será a utilização de capacetes ou salas de projeção para a grande maioria das aplicações (KIRNER, 2005). Porém, no momento com a facilidade de uso de simples computadores, há um crescimento acelerado no desenvolvimento de aplicações e modelos em realidade virtual não imersiva.

3.2.1. Ambientes Virtuais

De acordo com Sementille (2000), um Ambiente Virtual (AV) é um ambiente com o qual há a imersão do usuário em um ambiente tridimensional simulado. Um AV também pode ser entendido como um sistema de *software* que cria a ilusão de um mundo que não existe na realidade. Isto requer a combinação de entrada (interação do usuário), computação (simulação de processos) e saída (estímulos multi-sensoriais).

Os objetos no AV podem corresponder a objetos que existem no mundo real. Quando isso acontece, espera-se que esses objetos reajam como os objetos reais. Portanto, pode haver

a necessidade de se modelar, no mínimo, uma parte da Física (movimento e detecção de colisão entre objetos).

Alguns objetos no AV devem ser capazes de responder às ações dos usuários e às ações de outros objetos: devem existir meios de especificar esses comportamentos e associá-los aos objetos no modelo. Alguns objetos podem ter comportamentos autônomos, como andar, por exemplo. Isto envolverá interações entre os objetos e o tempo simulado dentro do ambiente. Existirá alguma sobreposição entre as técnicas requeridas aqui e as técnicas que estão sendo desenvolvidas em animação por computador e simulação.

Em um AV, o usuário deve poder navegar em três dimensões, ou seja, com seis graus de liberdade. Cada grau se aplica a uma direção ou rotação do movimento. Na verdade, conferem ao usuário a possibilidade de se movimentar em seis direções simultâneas: translação em torno dos três eixos cartesianos (x, y e z) e rotação em torno de cada um. A Figura 3.1 ilustra os seis graus de liberdade.

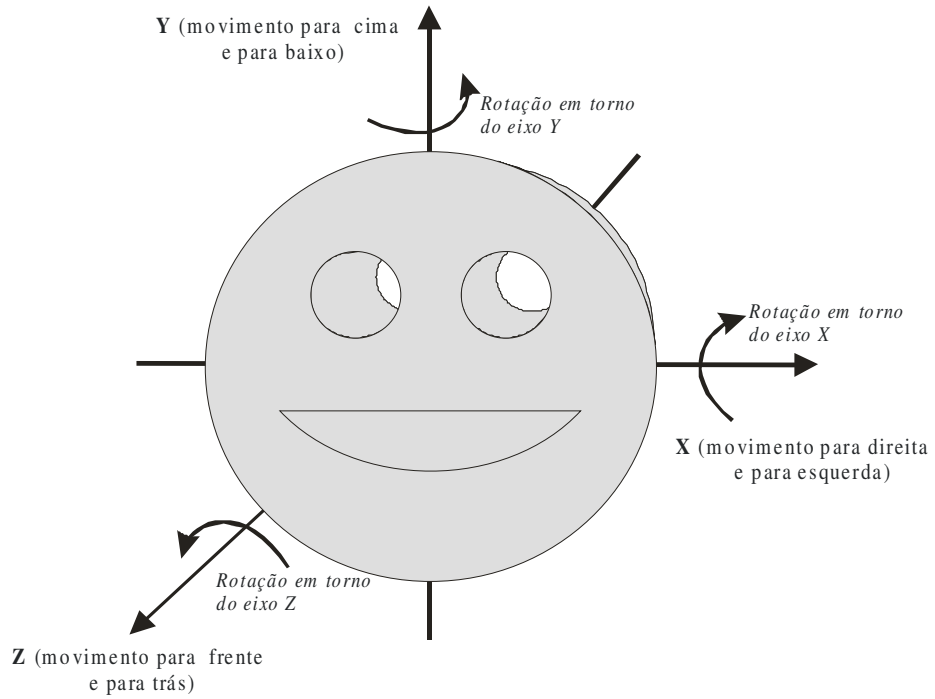


Figura 3.1. Seis graus de liberdade.

A modelagem visual de objetos é um processo consumidor de tempo, exigindo, portanto, melhores ferramentas para suportá-la.

Uma vez que a produção de bons modelos exige considerável esforço, deve existir algum mecanismo para o compartilhamento dos modelos existentes. Existem dois aspectos para este compartilhamento:

✓ O primeiro é a determinação de um formato padrão para a codificação e a transmissão dos modelos (o formato deve ser capaz de codificar toda a informação de modelagem, incluindo estrutura hierárquica, propriedades físicas dos objetos, som e comportamento).

✓ O segundo aspecto: devem existir um ou mais lugares que armazenem, mantenham e distribuam esses modelos.

3.2.2. Tipos de Sistemas de Realidade Virtual

O que diferencia uma aplicação de Realidade Virtual é o tipo de interface entre o usuário e o mundo virtual. Sob este aspecto, os tipos mais comuns são:

- **Sistemas Não-Imersivos:** utilizam o monitor de computador para exibir o mundo;

- **Sistemas Imersivos:** são sistemas capazes de imergir completamente o usuário no mundo virtual. Normalmente, esses sistemas são equipados com capacetes (HMD), luvas especiais, cavernas ou outros dispositivos multi-sensoriais (KIRNER, 2005).

Além desses dois principais tipos, outras formas comuns também podem ser citadas, como a Teleprensença, Realidade Mista, Realidade Aumentada e Virtualidade Aumentada (KIRNER, 2005).

3.2.3. Aplicações de Realidade Virtual

Atualmente, existem diversas aplicações que são regularmente usadas para resolver problemas do mundo real. Porém, muitas daquelas que são factíveis, precisam ainda ser implementadas.

As seguintes áreas de aplicação têm muito a ganhar com o desenvolvimento da tecnologia de AV:

- **Visualização de Modelo ou Dados**: as técnicas de visualização científica são amplamente aceitas como um meio de extrair um entendimento de uma grande quantidade de dados.

- **Projeto e Planejamento**: qualquer método que ajude o projetista, durante o ciclo de análise/refinamento melhorará a atividade inteira.

- **Educação e Treinamento**: modelos computacionais podem ser apresentados por meio de um ambiente virtual, já que uma vez que o nível de fidelidade apropriado possa ser provido, o usuário poderá interagir com o sistema virtual como se ele fosse real.

- **Teleoperações**: o AV pode ser aplicado em situações, onde um ambiente é muito perigoso para um humano, mas um tele-sensor/operador pode atuar.

- **Testes Psicológicos**: os AVs podem produzir estímulos ambientais irrealizáveis, que são úteis para o teste de teorias de percepção e controle manual.

- **Aplicações Artísticas e de Treinamento**: dado o montante de público e mídia interessado na tecnologia de AV, existe um potencial econômico para entretenimento como o AV.

- **Aplicações Artísticas**: como a tecnologia de AV está se tornando mais acessível, é razoável esperar que os artistas comecem a explorar as possibilidades únicas para a expressão que os ambientes virtuais oferecem.

- **Colaboração e Comunicação**: os ambientes virtuais, acoplados com redes de alta velocidade e computação distribuída, podem fornecer um espaço comum, onde a comunicação, negociação e colaboração podem tomar lugar.

3.3. Ambientes Virtuais Distribuídos

3.3.1. Conceituação e Caracterização

Ambientes virtuais distribuídos figuram entre os sistemas de *software* mais complexos já construídos (STYTZ, 1996). Esses ambientes devem satisfazer uma variedade de características como:

(1) Resposta rápida a novos requisitos do sistema.

(2) Capacidade de manutenção.

(3) Suportar interação em tempo real.

(4) Fidelidade da inserção do usuário no mundo virtual em relação a uma referência.

(5) Alta taxa de quadros por segundo, reusabilidade e portabilidade.

(6) Ajustamento a novas interfaces e dispositivos de visualização.

(7) Requisitos para capacidades adicionais.

A elaboração de um sistema de realidade virtual distribuído inclui atividades envolvendo (STYTZ, 1996):

(1) Suporte de comunicação em rede.

(2) Criação de ambientes virtuais.

(3) Atuação no mundo real.

(4) Criação de atores gerados por computador.

(5) Inserção de fenômenos naturais.

(6) Uso de simulação tradicional.

O suporte de comunicação em rede fornece os meios para que as unidades computacionais heterogêneas separadas fisicamente sejam unificadas para implementar um único ambiente virtual.

Uma maneira de manter o ambiente de RV atualizado é usar comunicação intensiva entre as máquinas do sistema, toda vez que houver uma atualização de posição. Para diminuir o tráfego na rede, o sistema pode utilizar a técnica de *dead-reckoning* (STYTZ, 1996) para minimizar a troca de mensagens e suportar atrasos de comunicação. Essa abordagem trabalha com a previsão da posição de um elemento, levando em conta o seu trajeto, velocidade e posição anterior, decorrido certo tempo. Todas as máquinas fazem o mesmo cálculo de previsão e reposicionam o elemento. Aquele que estiver gerenciando o elemento conseguirá verificar a diferença da trajetória real com a trajetória calculada. Sempre que essa diferença atingir um valor máximo, o valor real da posição será então comunicado às outras máquinas para devida correção. Dessa forma, não haverá necessidade de informar continuamente a posição de um elemento para as outras máquinas, o que diminuirá bastante a comunicação pela rede.

Os sistemas de RV multi-usuários em ambiente distribuído vêm crescendo e apresentam elevado potencial de aplicações. Esse tipo de sistema permite que os usuários geograficamente dispersos atuem em mundos virtuais compartilhados, usando a rede para melhor o desempenho coletivo, por meio da troca de informações (ARAÚJO, 1996).

Um sistema de realidade virtual multi-usuário pode ser centralizado ou distribuído (STYTZ, 1996):

- ✓ No modelo centralizado, todos os usuários compartilham o mundo virtual, ao passo que, no modelo distribuído, o mundo virtual pode ser replicado (para mundos pequenos) ou particionado (para mundos virtuais de grande porte).

- ✓ Num sistema replicado com *n* usuários, quando um usuário fizer qualquer alteração no mundo virtual, isto deverá ser comunicado para todas as (*n*-1) versões do mundo virtual em que estão os outros usuários, constituindo a difusão (*broadcast*).

- ✓ Num sistema particionado com *n* usuários, a situação é mais complexa, uma vez que o mundo virtual é dividido em várias partes e cada máquina ficará encarregada de uma delas. Como o usuário pode navegar no mundo virtual, ele poderá penetrar em outras regiões, de forma que sua máquina ou servidor deverá receber uma réplica da região em que ele se encontra. Assim cada máquina estará cuidando de uma região fora da sua parcela. Se existirem vários usuários em uma mesma região do mundo virtual, esse grupo de usuários receberá uma cópia dessa região. Qualquer alteração no mundo virtual, feita por um membro do grupo, será retransmitida para o restante do grupo, constituindo a retransmissão por grupo (*multicast*).

Para reduzir o número de conexões e de mensagens na rede são utilizadas as técnicas de difusão, retransmissão por grupo e *dead-reckoning*.

Um AVD pode ser mais bem definido como um sistema, por meio do qual diversos usuários interagem entre si e em tempo real, sendo que estes podem estar situados em localidades diferentes. Tipicamente, cada usuário acessa seu próprio computador, usando-o para fornecer uma interface para o ambiente virtual. Esses ambientes geralmente oferecem aos usuários uma sensação de realismo por meio da incorporação de gráficos 3D e som estéreo, a fim de criar uma experiência imersiva (SINGHAL, 1998).

Um AVD pode ser evidenciado por cinco características comuns (SEMENTILLE, 1999):

1. **Sensação de Compartilhamento de Espaço**: todos os participantes têm a ilusão de estarem localizados no mesmo lugar, tais como na mesma sala, prédio ou região. Esse espaço compartilhado representa um local comum, no qual outras interações podem acontecer. Este local pode ser real ou fictício. O local compartilhado deve apresentar as mesmas características a todos os participantes.

2. **Sensação de Presença**: quando entra em um local compartilhado, cada participante torna-se uma "pessoa virtual", chamada avatar, o qual inclui uma representação gráfica, um modelo de estrutura corporal (por exemplo, a presença de braços, cabeça, juntas, etc.), um modelo físico (por exemplo, peso altura etc.) e outras características. No entanto, outras vezes a presença em ambiente pode ser apenas resultado do envolvimento causado pela aplicação. Uma vez dentro de um AVD, cada participante pode ver os outros avatares ou suas representações localizados no mesmo espaço. Similarmente, quando um usuário deixa o AVD, os outros participantes vêem seu avatar ou sua representação partir. Nem todos os participantes precisam ser controlados por seres humanos. Alguns participantes podem ser entidades sintéticas controladas por modelos de simulação dirigidos por eventos.

3. **Sensação de Tempo Compartilhado**: os participantes devem ser capazes de ver o comportamento uns dos outros em tempo real.

4. **Comunicação entre os Participantes**: embora a visualização seja a base para um AVD efetivo, muitos AVDs também se empenham em permitir que algum tipo de comunicação ocorra entre os participantes. Essa comunicação pode acontecer por meio de gestos, escrita ou voz, adicionando uma sensação maior de realismo a qualquer ambiente simulado e é fundamental para sistemas de engenharia e treinamento.

5. **Forma de Compartilhamento**: os elementos mencionados anteriormente efetivamente oferecem um sistema de videoconferência de alta qualidade. O poder real de um AVD deriva da habilidade dos usuários em interagir uns com os outros e com o próprio ambiente virtual. Em uma simulação de combate ou jogo, por exemplo, os usuários precisam atirar ou colidir uns nos outros em interações modeladas realisticamente. Os usuários podem ser capazes de escolher, mover e manipular objetos que existem no ambiente e até entregá-los a outros participantes.

A importância por trás dos AVDs reside na integração de gráficos 2D e 3D com a habilidade de possuir diversos usuários compartilhando informação e manipulando objetos no ambiente. A presença de diversos usuários independentes diferencia os AVDs dos sistemas de Realidade Virtual tradicionais. Os AVDs são mais apropriados para aplicações que demandam a criação de tele-presença[9], ou seja, a ilusão que outros usuários estão visíveis, mesmo que situados fisicamente em lugares remotos. Nessas aplicações, os usuários necessitam uma sensação de realismo próxima da alcançada em um contato face a face.

3.3.2. Componentes de um AVD

Um sistema de AVD consiste de quatro componentes básicos: 1) um display gráfico; 2) dispositivos de comunicação e controle; 3) um sistema de processamento; 4) e uma rede de comunicação. Estes componentes trabalham juntos para fornecer a sensação de imersão em diferentes localidades (SEMENTILLE, 1999).

Além desses componentes, aspectos quanto ao desenvolvimento de *software* para AVDs, também são muito importantes, e envolvem a comunicação em rede, visões do

[9] **Tele-presença** é um ambiente comum compartilhado por vários usuários em vários lugares diferentes se encontrando em um mesmo ambiente virtual.

ambiente, modelo de dados, gerenciamento da computação e comportamento dos objetos (MACEDONIA, 2005).

3.3.3. Comunicação em Rede

Diversas características da comunicação em rede devem ser consideradas e suas principais características são: a largura de banda, a latência, a confiabilidade e os esquemas de comunicação (VLOKA, 1995).

3.3.3.1. Largura de Banda

A largura de banda é a taxa na qual a rede pode entregar um dado a um computador (host) destinatário. É a quantidade de informação que pode ser transferida de um nó para outro em um determinado período. Algumas aplicações têm de transmitir dados, a certa velocidade para serem efetivas. Por exemplo, se uma aplicação de telefonia por Internet codifica a voz a 32 Kbps, então ela deve poder também enviar os dados para a rede e fazer com que sejam entregues na aplicação receptora a essa mesma taxa. Se essa largura de banda não estiver disponível, a aplicação precisará codificar a uma taxa diferente, ou então desistir, já que receber metade da largura de banda de que precisa de nada adianta para tal aplicação sensível à largura de banda (VLOKA, 1995; KUROSE, 2003).

3.3.3.2. Latência

A latência de uma rede é o tempo requerido pela rede para transferir um bit de dado de um ponto a outro. A latência de rede pode variar muito. Se o dado for enviado através de uma rede local *Ethernet*, a latência tipicamente gira em torno de poucos milissegundos (10 a 20 ms). Porém, se a comunicação for feita usando-se a Internet, através de um modem e do sistema telefônico, sua latência será de, no mínimo, 100 milissegundos (SINGHAL, 1998).

Do ponto de vista do ambiente virtual distribuído, é a latência que controla a natureza interativa e a dinâmica do sistema. Se o ambiente distribuído existe para emular o mundo real, deve operar em tempo real, em termos de percepção humana (MACEDONIA, 2005).

3.3.3.3. Confiabilidade da Rede

A confiabilidade da rede é uma medida que reflete a quantidade de dado que é perdido pela rede durante o trajeto do *host* fonte para o destino. Este dado perdido pode ser dividido em duas categorias: dado interrompido e dado corrompido. Muitos pesquisadores consideram a interrupção e a corrupção de dado essencialmente equivalente, isto porque, em ambos os casos, o dado não atinge efetivamente a aplicação destino (SINGHAL, 1998).

3.3.3.4. Esquemas de Comunicação

O modelo de comunicação tem muita influência com a escalabilidade de um sistema. A escalabilidade pode ser mensurada pela quantidade de entidades que podem, simultaneamente, participar do sistema (SEMENTILLE, 1999). Alguns esquemas escalam melhor do que outros. Entre os principais modelos de comunicação tem-se:

- *broadcast* (por difusão – ocorre quando o dado é enviado para todos os *hosts,* ou seja, uma mensagem enviada por um *host* é recebida por todos os outros, que podem até ignorá-las; este modelo possui alto custo e, é possível somente em redes locais);

- *unicast* (ponto a ponto - ocorre quando se estabelece uma comunicação entre dois *hosts*);

- *multicast* (ocorre quando um grupo de *hosts* se comunicam pela rede, via uma única transmissão) (KIRNER, 1988).

3.3.4. Visões do Usuário de um AVD

A visão (ou perspectiva) que um usuário ou processo tem do ambiente virtual é geralmente representada por meio de janelas. Existem dois paradigmas principais para visão em ambientes distribuídos: a visão síncrona e a visão assíncrona.

- Visão síncrona: nesse tipo de paradigma, têm-se várias máquinas controlando visões diferentes de um mesmo ambiente. As imagens exibidas por estas máquinas devem ser sincronizadas de forma a dar a ilusão de conjunto. Como exemplo, um simulador de vôo, em que máquinas diferentes controlam a visão da frente, da direita, da esquerda e da cabine. As imagens são coordenadas de maneira a parecer que todas fazem parte de uma única visão da cabine. Alguns AVDs que usam visão síncrona são: RAVEN (CATER, 1995) e o CAVE. Por exigir alta confiabilidade e menor latência, a visão síncrona é restrita a redes locais.

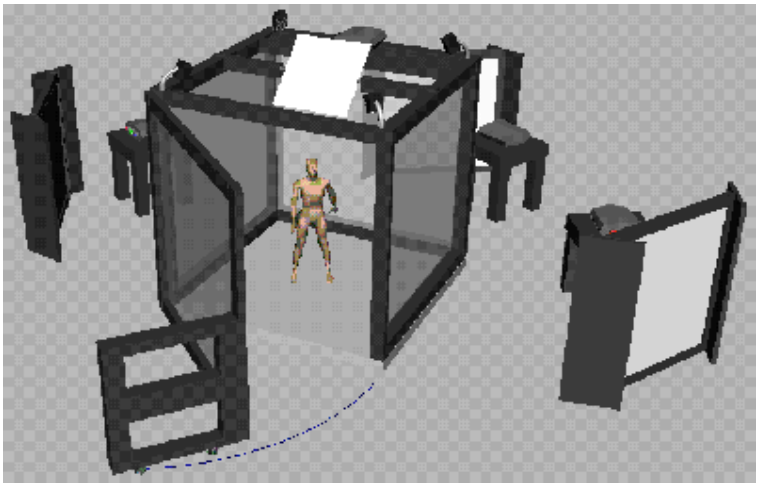


Figura 3.2. Imagem de uma CAVE

- Visão assíncrona: esse paradigma é mais geral e permite que múltiplos usuários tenham controle individual sobre "quando" e "o que" podem estar fisicamente separados por uma rede local ou uma rede a longa distância. Como exemplo de AVDs que utilizam a visão assíncrona, tem-se o NPSNET (NPSNET, 2004).

3.3.5. Modelos de Dados

A determinação do modelo de dados a ser utilizado em um AVD é, talvez, a decisão de construção mais difícil de ser tomada, pois afeta o escalonamento, os requisitos de comunicação e a confiabilidade do sistema.

Existem diversos conceitos para a distribuição persistente ou semi-persistente de dados. As consideradas mais relevantes são:

- **Modelo da Base de Dados Homogênea Replicada**: nesse modelo, tem-se uma base de dados homogênea que contém todas as informações sobre a geometria, texturas e comportamento de todos os objetos que integram o mundo virtual. Esta base de dados é, então, replicada em todos os *hosts* que participam do AVD. Uma vantagem desse modelo é que as mensagens são geralmente pequenas, uma vez que os participantes só são informados sobre as mudanças de estado dos objetos, como por exemplo, mudanças de posição e colisões entre objetos. As desvantagens são que a base de dados de usuário tende a crescer com o aumento do conteúdo do ambiente virtual.

- **Modelo da Base de Dados Centralizada e Compartilhada**: neste modelo, as máquinas dos participantes comunicam-se com um único servidor central, o qual mantém a base de dados com todas as informações e estado do mundo virtual. A principal vantagem desse modelo reside na facilidade de manutenção da consistência da base de dados. A principal desvantagem é que o uso de um servidor centralizado para mundos virtuais 3D limita o número de participantes devido à concentração de E/S e a manutenção de uma base de dados de objetos, em tempo real. O servidor central torna-se rapidamente um "gargalo" do sistema, já que deve retransmitir o estado de cada participante a todos os demais.

- **Modelo da Base de Dados Distribuída, Compartilhada, Ponto a Ponto**: a idéia por trás desse modelo é fazer com que o sistema distribuído simule as arquiteturas de

memória compartilhada. A base de dados é homogênea, distribuída e completamente replicada entre as máquinas dos participantes. Como exemplo tem-se o sistema DIVE (CARLSSON, 1993), no qual a base de dados inteira é dinâmica e utiliza protocolos *multicast* confiáveis para replicar novos objetos. A desvantagem é em relação aos custos de comunicação, associados com a manutenção da confiabilidade e da consistência dos dados.

- **Modelo da Base de Dados Distribuída, Compartilhada, Cliente/Servidor**: outro modelo que pode ser utilizado é uma variação do modelo cliente/servidor, no qual a base de dados é particionada entre os clientes e a comunicação é mediada por um servidor central. Nesse sistema, o servidor central possui o conhecimento de qual parte do mundo virtual reside em cada cliente. Toda vez que um objeto se move no ambiente virtual, a base de dados é atualizada por intermédio do servidor. Todavia, em um ambiente virtual dinâmico de grande escala, os servidores tendem a se tornar os "gargalos" de E/S do sistema, aumentando a latência inerente ao ambiente.

3.3.6. Gerenciamento da Computação

A computação, tanto quanto os dados, também pode ser distribuída. Como computação, no caso, entende-se qualquer tarefa, envolvendo um objeto especifico. Os modelos de gerenciamento mais importantes são:

Distribuição Completa (ou Total): nesse modelo, todas as operações sobre um objeto devem ser executadas no mesmo nó que mantém os dados sobre aquele objeto. Se um objeto deseja executar uma operação sobre outro, deve, então, enviar uma mensagem para o outro processo. A alocação de processos aos nós pode ser otimizada por meio da utilização de algoritmos para balanceamento de carga. Monitorando-se o consumo de recursos e padrões de comunicação, pode-se derivar uma alocação ótima. Isto permitirá que muitos objetos que se comunicam com freqüência entre si, possam ser colocados no mesmo nó – esta movimentação

de processos é comumente conhecida como migração. Essa abordagem funciona bem em sistemas fortemente acoplados.

Distribuição Parcial: este método é similar ao da distribuição completa, exceto que o estado dos objetos é normalmente adquirido, usando-se técnicas de replicação parcial de dados e as mudanças são feitas localmente. Porém, não existe duplicação de esforço computacional.

Replicação Parcial: para compensar ligações de comunicações mais lentas, por exemplo, como nos sistemas fracamente acoplados, é possível replicar o estado da computação em alguns ou todos os nós. Estes processos-réplica são geralmente usados para aproximar o comportamento do objeto, usando-se o método conhecido como *dead-reckoning*. O processo, que executa a completa simulação daquele objeto, também processa esse modelo em paralelo e, quando os dois diferentes de uma quantidade pré-definida, uma cópia das variáveis de estado reais daquele objeto é enviada para todos os processos-réplica. As subseqüentes aproximações são, então, baseadas na última atualização.

Replicação Total: a simulação de cada objeto em cada nó pode ser requerida, se a simulação está sendo executada numa rede de longa distância (francamente acoplada). Neste modelo, somente a informação, que muda o comportamento dos objetos espelhados, é enviada, fazendo com que todos os cálculos sejam feitos localmente. Por conseguinte, o comportamento parece correto em todo lugar (apesar de poder não ser exatamente o mesmo, no tempo), mas ao custo da duplicação de computação.

3.3.7. Comportamento dos Objetos

Atualmente, ainda é tema de discussão sobre o que consiste o "comportamento" do objeto e como este deve ser modelado. No estrito senso da orientação a objeto, os dados são atributos e os métodos manipulam os atributos de uma forma pré-definida, por exemplo,

mudando de posição a todo instante. Portanto, combinando-se dados e métodos, consegue–se a impressão de "comportamento". No entanto, a carga computacional requerida para suportar este comportamento do objeto pode ser muito alta.

É possível classificar o comportamento do objeto como determinístico ou não-determinístico. Em geral, objetos que não necessitam de entradas (via dispositivos de entrada) são determinísticos, ao passo que aqueles objetos que necessitam, inclusive aqueles sob o controle de humanos, são não–determinísticos. Por exemplo, as decisões tomadas por um carro-robô podem ser determinadas durante seu avanço, porém, em um carro virtual sendo dirigido por um humano, em um simulador de direção, isto não pode ser previsto (HAWKES, 1996). A capacidade de prever o comportamento significa que é possível melhorar a comunicação e a latência do sistema.

Segundo Roehl (2004), o comportamento determinístico pode ser subdivido em duas sub-categorias: estática e animada. Similarmente, o comportamento não-determinístico pode ser Newtoniano ou inteligente (ROEHL, 2004). O estado de um objeto estático é constante e, portanto, 100% previsível, a todo o momento; já um objeto animado muda seu estado a todo instante, mas ainda é previsível. Um objeto Newtoniano interage com seu ambiente, mas não de uma maneira direta. Portanto, um objeto inteligente pode ter um comportamento complexo e tão imprevisível quanto o de um ser humano.

3.4. Principais Ambientes Virtuais Distribuídos

O objetivo deste tópico é oferecer uma visão geral dos principais AVDs encontrados na literatura, expondo suas características mais importantes de acordo com uma metodologia. A preocupação na avaliação desses sistemas inicia em duas partes, modelagem e distribuição, porém em algumas vezes estas partes estão muito entrelaçadas com influência de uma em outra.

A seguir, consta a metodologia proposta para avaliação dos AVDs. Esta metodologia, relatada em sua totalidade em Snowdon (1994), Sementille (1999) e Brutzman (2004), foi utilizada parcialmente para caracterizar os AVDs avaliados neste trabalho.

- **Componentes dos Ambientes Virtuais:** nessa avaliação, a preocupação é relativa às características dos objetos;

- **Distribuição:** nesta avaliação, a preocupação é relativa à abordagem da distribuição no ambiente.

Seguindo essas duas premissas, a metodologia é apresentada a seguir em nove itens:

i. Múltiplos Mundos Virtuais
ii. Artefatos
iii. Geometria
iv. Modelo utilizado
v. Replicação
vi. Comunicação
vii. Computação Distribuída
viii. Ambiente de Rede
ix. Usuários

3.4.1. AVDs avaliados

i. **BrickNet *Toolkit (Institute of Systems Science – National University of Singapore)*** (MACEDONIA, 2005).

Múltiplos Mundos Virtuais: Permite. Os usuários possuem recursos para visões dinâmicas.

Artefatos: O sistema permite que os objetos sejam criados e modificados em tempo de execução.

Geometria: Permite edição de geometria.

Modelo utilizado: Base de Dados distribuída.

Comunicação: Modelo Cliente/Servidor - os dados são particionados entre os clientes por meio de um servidor único.

Computação Distribuída: Permite. Cada nó é responsável por uma tarefa menor colaborando numa tarefa maior.

Ambiente de Rede: Heterogêneo (rede de estações de trabalho Silicon *Graphics*).

Usuários: O usuário interage com o mundo virtual por de dispositivos e técnicas de interação.

ii. **PEPITO –** ***PEer-to-Peer: Implementation and TheOry (Swedish Institute of Computer Science)*** (EL-ANSARY, 2005)

Múltiplos Mundos Virtuais: Permite. Fornece "portais" para um usuário se mover de um mundo virtual para outro.

Artefatos: Não permite a edição de artefatos dinâmicos.

Geometria: Permite edição de geometria.

Modelo utilizado: Base de Dados compartilhada.

Replicação: A base de dados é completamente replicada.

Comunicação: Utiliza o modelo *peer-to-peer.*

Computação Distribuída: Distribuição parcial.

Ambiente de Rede: Heterogêneo.

Usuários: Existe uma separação explícita entre o usuário e a aplicação.

iii. **CurlSpace –** ***University of Bristo*** (KOTZIAMPASIS, 2003; KOTZIAMPASIS 2004)

Múltiplos Mundos Virtuais: Não permite.

Geometria: Não permite edição de geometria.

Modelo utilizado: Estrutura de dados compartilhada.

Replicação: Parcialmente replicada.

Comunicação: Modelo Cliente/Servidor.

Computação Distribuída: Permite.

Ambiente de Rede: Heterogêneo.

Usuários: Permite que vários usuários interajam com o ambiente, sendo que cada um representa um portal do sistema.

iv. DKS – *Distributed K-ary System (Swedish Institute of Computer Science - SICS)* (ALIMA, 2004)

Múltiplos Mundos Virtuais: Permite.

Artefatos: Pode criar um número arbitrário de instâncias em tempo de execução.

Modelo utilizado: Modelo orientado a objeto.

Replicação: Completa.

Comunicação: Utiliza o modelo ponto a ponto *multicast e broadcast.*

Computação Distribuída: Distribuição parcial.

Ambiente de Rede: Heterogêneo.

Usuários: Cada usuário no ambiente virtual representa um objeto.

v. NPSNET *(Naval Postgraduate School – USA)* (NPSNET, 2004)

Múltiplos Mundos Virtuais: Não permite.

Artefatos: Permite tipos e a criação dinâmica de artefatos.

Geometria: Permite edição de geometria.

Modelo utilizado: Modelo de simulação distribuída.

Replicação: Completa.

Comunicação: Utiliza o modelo ponto a ponto *multicast.*

Computação Distribuída: Permite.

Ambiente de Rede: Heterogêneo.

Usuários: O NPSNET é especificamente um sistema de simulação para treinamento militar.

vi. SIMNET - *Simulation Network (BBN Laboratories)* (KAMARA, 2005)

Múltiplos Mundos Virtuais: Não permite.

Geometria: Permite edição de geometria.

Modelo utilizado: Modelo de orientação a objetos e eventos, com base de dados replicada totalmente.

Replicação: Completa.

Comunicação: Modelo ponto a ponto.

Computação Distribuída: Permite.

Ambiente de Rede: Heterogêneo.

Usuários: Existe uma separação explícita entre o usuário e a aplicação. O DIVE trata usuários como processos.

vii. **DVE –** ***Distributed Virtual Environment (Chinese Univ. of Hong Kong, Shatin, China)*** (LUI, 2002)

Múltiplos Mundos Virtuais: Não permite.

Artefatos: Permite a edição de artefatos dinâmicos.

Modelo utilizado: Base de Dados distribuída.

Replicação: Parcial.

Comunicação: Ponto a ponto cliente/servidor.

Computação Distribuída: Permite.

Ambiente de Rede: Heterogêneo.

Usuários: Os usuários são representados como objetos os quais contém um estado encapsulado e um modelo comportamental explícito.

viii. **EVI3d framework** ***(University of Nottinghan – University of Manchester - UK)*** (TOURAINE, 2002)

Múltiplos Mundos Virtuais: Não permite.

Artefatos: Permite a criação dinâmica de artefatos.

Modelo utilizado: Base de Dados distribuída. O sistema pode ser representado como uma coleção de objetos autônomos comunicantes, executando concorrentemente.

Replicação: Parcial.

Comunicação: Ponto a ponto e cliente/servidor.

Computação Distribuída: Permite. Para distribuir a computação entre os processadores o EVI3d suporta uma classe de objetos conhecidos como "servidores de objetos".

Ambiente de Rede: Heterogêneo.

Usuários: O EVI3d modela um usuário como uma entidade separada do sistema RV, logicamente equivalente a uma aplicação.

ix. **Sistema MASSIVE –** ***Model, Architecture, and System for Spatial Interaction in Virtual Environment. (University of Nottinghan – UK, University Manchester – UK))*** (GREENHALGH, 2000)

Múltiplos Mundos Virtuais: Permite. O MASSIVE suporta múltiplos mundos virtuais conectados via "portais". Cada mundo virtual pode ser habitado por muitos usuários que interagem através de *interfaces* de gráficos, áudio e texto.

Modelo utilizado: Usa processos computacionais independentes que se comunicam através de conexões ponto a ponto.

Comunicação: Ponto a ponto.

Computação Distribuída: Permite.

Ambiente de Rede: Heterogêneo.

Usuários: Cada usuário pode incorporar suas próprias características gráficas através de um arquivo de configuração.

x. **Sistema TeamSpace** ***(Distributed Systems Group at Stanford University)*** (HEBERT, 2005)

Múltiplos Mundos Virtuais: Permite.

Artefatos: Permite a criação dinâmica de artefatos.

Modelo utilizado: Base de Dados compartilhada.

Replicação: Total.

Comunicação: Modelo *multicast*, cliente/servidor. Usa o IP *multicast*, associando um diferente endereço *multicast* a cada objeto ativo.

Computação Distribuída: Distribuição parcial.

Ambiente de Rede: Heterogêneo.

Usuários: O sistema TeamSpace endereça questões importantes da arquitetura do *software* e de rede, para a criação de ambientes com milhares de usuários.

xi. **Sistema SPLINE –** ***Scalable Platform for Large Interactive Networked Environments*** (GARCIA, 2002)

Múltiplos Mundos Virtuais: Não mencionado.

Artefatos: Permite a edição de artefatos dinâmicos.

Modelo utilizado: Base de Dados distribuída.

Replicação: Parcial. O SPLINE distribui o modelo do mundo, mantendo uma cópia parcial em cada processo.

Comunicação: Modelo *multicast.* A comunicação inter-processo é utilizada por meio do envio, e da recepção de mensagens de outros processos sobre as mudanças feitas remotamente.

Computação Distribuída: Permite. O SPLINE utiliza a idéia de que enquanto um mundo virtual pode ser muito grande, apenas uma pequena parte dele pode visualizada por um único usuário.

Ambiente de Rede: Heterogêneo.

Usuários: O SPLINE permite a participação de um grande número de usuários em uma mesma simulação. Os usuários podem se comunicar utilizando a linguagem natural.

xii. **Sistema VERN –** ***Virtual Environment RealTime Network (Institute for Simulation and Training – University of Central Florida)*** (YARDI, 2005)

Múltiplos Mundos Virtuais: Permite.

Artefatos: Permite a edição de artefatos dinâmicos.

Modelo utilizado: Base de Dados distribuída.

Replicação: Total.

Comunicação: Modelo *broadcast.*

Computação Distribuída: Permite. Cada objeto do mundo real participante da simulação é representado por um software chamado *Player.* A principal responsabilidade do *Player* é manter as informações de estado, ler e processar entradas, fornecer uma realimentação de gráficos em tempo real e informar a rede de qualquer mudança significativa que se desvie do modelo *dead-reckoning.*

Ambiente de Rede: Heterogêneo.

Usuários: O VERN permite que múltiplos participantes interajam em um ambiente, compartilhando idéias e resolvendo problemas, independente de sua localização física.

xiii. AVVIC *(Ambiente Virtual para Visualização Interativa Compartilhada)* (SEMENTILLE, 1999; KIRNER, 1999; KIRNER, 2004)

Múltiplos Mundos Virtuais: Permite.

Artefatos: O sistema permite que os objetos sejam criados e modificados em tempo de execução.

Geometria: Permite edição de geometria.

Modelo utilizado: Base de Dados distribuída.

Comunicação: Modelo Cliente/Servidor – cada nó do sistema distribuído receberá a descrição do espaço com seus objetos, bem como o processo de simulação a ser realizado.

Computação Distribuída: Permite. Cada nó é responsável por uma tarefa menor colaborando numa tarefa maior.

Ambiente de Rede: Heterogêneo (permite a aplicação em redes de longa distância).

Usuários: O usuário interage com o mundo virtual por meio de dispositivos e técnicas de interação.

As principais características observadas nos sistemas avaliados foram sintetizadas na Tabela 3.1. Esses sistemas apresentam alguns aspectos comuns a quase todos e merecem ressalva. A estrutura de comunicação Cliente/Servidor, o mecanismo de transporte Ponto a Ponto, o gerenciamento da computação com Distribuição Total, o ambiente de rede heterogêneo e o gerenciamento dos dados com Banco de Dados alternando entre compartilhada e distribuída caracterizaram a maioria dos sistemas avaliados e serão usados como referência na construção do modelo proposto neste trabalho. Em relação a multidisciplinaridade, nenhum dos AVDs avaliados foi construído com a preocupação de suportar ambientes virtuais distintos, como mostra a última coluna da Tabela 3.1. Esta característica encontrada nesses AVDs reforça a necessidade da construção de uma arquitetura de suporte à ambientes virtuais multidisciplinares.

Tabela 3.1. Principais características dos AVDs avaliados.

AVDs	CARACTERÍSTICAS					
	Mecanismo de Transporte	**Estrutura de Comunicação**	**Gerenciamento de Dados**	**Gerenciamento da Computação**	**Suporte a Múltiplos Mundos**	**Suporte a AVs Distintos**
BrickNet	Cliente/Servidor	Cliente/Servidor	BD Distribuída com replicação parcial	Distribuição Total	Sim	**Não**
PEPITO	Ponto a Ponto	Cliente/Servidor	BD Distribuída com replicação parcial	Distribuição Parcial	Sim	**Não**
CurlSpace	Cliente/Servidor	Cliente/Servidor	BD Distribuída com replicação parcial	Distribuição Parcial	Sim	**Não**
DKS	Ponto a Ponto, *Multicast, Broadcast*	Cliente/Servidor	BD Distribuída com replicação parcial	Distribuição Total	Sim	**Não**
NPSNET	Ponto a Ponto, *Multicast*	*Peer*	BD Distribuída com replicação total	Distribuição Parcial	Não	**Não**
SIMNET	Ponto a Ponto	*Peer*	BD Distribuída com replicação total	Distribuição Parcial	Não	**Não**
DVE	Ponto a Ponto, Cliente/Servidor	Cliente/Servidor	BD Centralizado	Replicação Total	Sim	**Não**
EVI3D	Ponto a Ponto, Cliente/Servidor	Cliente/Servidor	BD Compartilhado	Replicação Parcial	Sim	**Não**
MASSIVE	Ponto a Ponto	*Peer*	Não mencionada	Distribuição Total	Sim	**Não**
TeamSpace	*Multicast*, Cliente/Servidor	Cliente/Servidor	BD Distribuída com replicação total	Distribuição Total	Sim	**Não**
SPLINE	*Multicast*	*Peer*	BD Distribuída com replicação total	Distribuição Total	Não	**Não**
VERN	*Broadcast*	*Peer*	BD Distribuída com replicação total	Distribuição Total	Não	**Não**
AVVIC	*Multicast* e Ponto a ponto	Cliente/Servidor	BD Distribuída com replicação total	Distribuição Total	Sim	**Não**

3.5. Considerações Finais

Este capítulo apresentou os conceitos de Realidade Virtual e Ambiente Virtual. Mostrou que a junção de Sistemas Distribuídos e Realidade Virtual possibilita a existência dos AVDs (ambientes virtuais distribuídos), e relatou as principais características que um AVD necessita ter. E, por último, apresentou também conjunto de AVDs, avaliados com a finalidade de subsidiar a construção da arquitetura do sistema desenvolvido neste trabalho. Cabe ressaltar que o foco deste trabalho é a distribuição de AVs multidisciplinares, característica não contemplada nos AVDs avaliados.

No próximo capítulo, apresenta-se a arquitetura necessária que possibilita a construção de ambientes virtuais distribuídos multidisciplinares.

CAPÍTULO IV

4. ARQUITETURA DO SISTEMA

4.1. Introdução

A partir do estudo realizado nos Capítulos 2 e 3, foi possível conceber uma arquitetura para um ambiente virtual distribuído multidisciplinar que ofereça ao usuário condições de acompanhar e interagir com ambientes virtuais relacionados a diferentes áreas do conhecimento. Além da dinamicidade gerada por um conjunto de diversos ambientes (números de cópias indefinidas e não iguais de cada um dos mundos virtuais), esta arquitetura permite ao usuário entender melhor o processo disciplinar relativo a cada ambiente, contribuindo para o processo de educação.

Antes de mostrar a referida arquitetura do sistema em detalhes, será abordada a tecnologia de apoio necessária ao funcionamento do mesmo, bem como a importância de cada uma das tecnologias utilizadas.

4.2. Tecnologias de Apoio

Novas tecnologias têm sido criadas para dar suporte ao desenvolvimento de aplicações em Realidade Virtual (SEMENTILLE, 1999). Tradicionalmente, os displays gráficos somente eram disponíveis em estações de trabalho. Porém, atualmente os preços do *hardware* têm caído, a ponto de permitir que essas capacidades gráficas estejam presentes nos computadores pessoais (PCs).

4.2.1. OpenGL

OpenGL é definido como um programa de interface para *hardware* gráfico. Na verdade, OpenGL é uma biblioteca de rotinas gráficas e de modelagem, bi (2D) e tridimensional (3D), extremamente portável e rápida. Usando OpenGL, é possível criar gráficos 3D com uma alta qualidade visual. Entretanto, a maior vantagem na sua utilização é a rapidez, uma vez que usa algoritmos cuidadosamente desenvolvidos e otimizados pela *Silicon Graphics, Inc.*, líder mundial em Computação Gráfica e Animação (WOO, 1999).

OpenGL não é uma linguagem de programação, mas é uma poderosa e sofisticada API (*Application Programming Interface*) para criação de aplicações gráficas 2D e 3D. Normalmente, se diz que um programa é baseado em OpenGL ou é uma aplicação OpenGL, o que significa que ele é escrito em alguma linguagem de programação que faz chamadas a uma ou mais bibliotecas OpenGL (PINHO, 2004).

As aplicações OpenGL variam de ferramentas CAD a programas de modelagem usados para criar personagens para o cinema. Além do desenho de primitivas gráficas, tais como linhas e polígonos, OpenGL dá suporte a iluminação, colorização, mapeamento de textura, transparência, animação, entre muitos outros efeitos especiais. Atualmente, OpenGL é reconhecida e aceita como um padrão API para desenvolvimento de aplicações gráficas 3D em tempo real.

Em vez de descrever a cena e como ela deve parecer, quando se está utilizando OpenGL é preciso apenas determinar os passos necessários para alcançar a aparência ou efeito desejado. Estes passos envolvem chamadas a esta API portável, que inclui aproximadamente 250 comandos e funções (200 comandos do *core* OpenGL e 50 da GLU - *OpenGL Utility Library*). Por ser portável, OpenGL não possui funções para gerenciamento de janelas,

interação com o usuário ou arquivos de entrada/saída. Cada ambiente, como por exemplo o *Microsoft Windows*, possui suas próprias funções para esses propósitos. Não existe um formato de arquivo OpenGL para modelos ou ambientes virtuais. OpenGL fornece um pequeno conjunto de primitivas gráficas para construção de modelos, tais como pontos, linhas e polígonos. A biblioteca GLU (que faz parte da implementação OpenGL) é que fornece várias funções para modelagem, tais como superfícies quádricas, e curvas e superfícies NURBS (*Non Uniform Rational B-Splines*) (WOO, 1999; WRIGHT, 2000).

A palavra *pipeline* é usada para descrever um processo, que pode ter dois ou mais passos distintos. A Figura 4.1 mostra uma versão simplificada do *pipeline* OpenGL. Como uma aplicação faz chamadas às funções API OpenGL, os comandos são colocados em um *buffer* de comandos. Este *buffer* é preenchido com comandos, vértices, dados de textura etc. Quando este *buffer* é "esvaziado", os comandos e dados são passados para o próximo estágio.

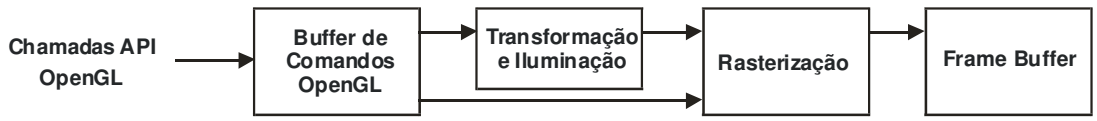


Figura 4.1. Versão simplificada do *pipeline* OpenGL (WOO, 1999).

Após a etapa de aplicação das transformações geométricas e da iluminação, é feita a rasterização, ou seja, é gerada a imagem a partir dos dados geométricos, de cor e textura. A imagem final, então, é colocada no *frame buffer*, que é a memória do dispositivo gráfico. Isto significa que a imagem é exibida no monitor (WRIGHT, 2000).

4.2.2. CORBA

Como mencionado no Capítulo 2, o padrão CORBA é um sistema que permite a aplicações distribuídas (local ou mesmo na Internet) realizar comunicação entre si e trocar

informações em uma rede. Esta tecnologia compõe a principal parte da arquitetura proposta para esse trabalho.

4.2.3. Visibroker

Existem muitas implementações CORBA disponíveis. Para implementação deste sistema, foi usada a implementação CORBA da *Borland* ^{TM}. O produto CORBA é chamado VisiBroker, que é um ORB (*Object Request Broker*) desenvolvido pela INPRISE (TEIXEIRA, 2002).

4.3. Arquitetura do Sistema

O propósito deste trabalho é a elaboração de uma arquitetura que permita a existência de *n* computadores capazes de hospedar *m* Ambientes Virtuais, propondo-se a seguinte arquitetura, conforme ilustra a Figura 4.2.

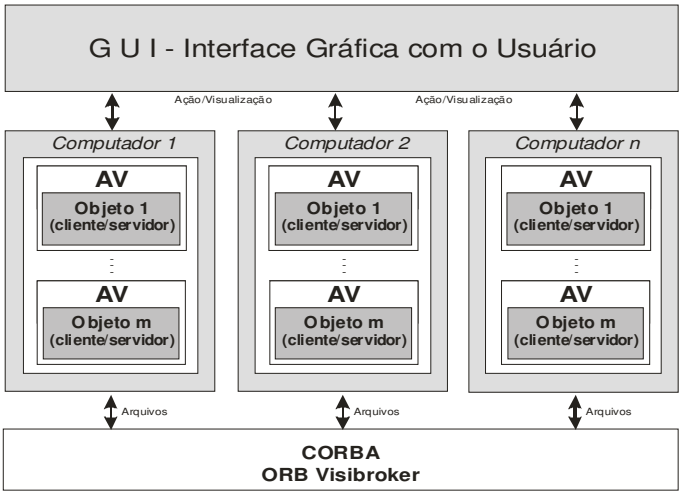


Figura 4.2. Arquitetura proposta para o sistema.

Resumidamente, verifica-se que:

- ✓ A Figura 4.2 ilustra a existência de *n* computadores, sendo que cada computador pode hospedar *m* ambientes virtuais e sempre haverá uma aplicação

servidora que proverá serviços para os clientes. Cada objeto pode simular um ambiente diferente (Biologia ou Química).

- ✓ Cada ambiente virtual pode ser executado separadamente e possui interação dentro do próprio ambiente. A interação é sempre disparada por um determinado evento. Se um par de ambientes estiver ativo em uma rede, além das respostas no próprio ambiente (animação), haverá também em outro ambiente o disparo de eventos.
- ✓ Os ambientes virtuais foram modelados com o uso da biblioteca gráfica OpenGl e parametrizados dentro das próprias funções de modelagem. Esses parâmetros são repassados de clientes para clientes, clientes para servidores e servidores para clientes.
- ✓ O usuário interage com um ambiente virtual, de tal forma que as informações são capturadas pelo objeto respectivo. O objeto-cliente localiza na rede um objeto-servidor com a ajuda VisiBroker (padrão CORBA), que responderá à requisição do objeto-cliente.
- ✓ O objeto-servidor repassa as informações ao ambiente virtual e apresenta a nova interface ao usuário.

Existem outros pontos importantes, não apresentados na figura que representa a arquitetura:

- ✓ Sempre existirá um servidor, único ou relacionado com cada ambiente, sendo que os outros ambientes se portarão apenas como clientes.
- ✓ Sempre que um servidor for desligado, outro automaticamente (por meio de algoritmo que permite ao sistema ser tolerante a falhas) assumirá sua posição até que não exista nenhum ambiente apto a se tornar servidor na rede.

- ✓ As informações de cada ambiente estarão armazenadas sempre em um servidor. Porém, cada cliente armazena informações sobre sua atual situação. Essa informação é repassada a um novo servidor, quando necessário.

Como descrito acima, o modelo proposto utiliza a arquitetura CORBA como camada intermediária (*middleware)* na comunicação de objetos. A Figura 4.3 representa também o modelo proposto, diferente da Figura 4.2, pois nessa o enfoque é sobre as camadas presentes na arquitetura.

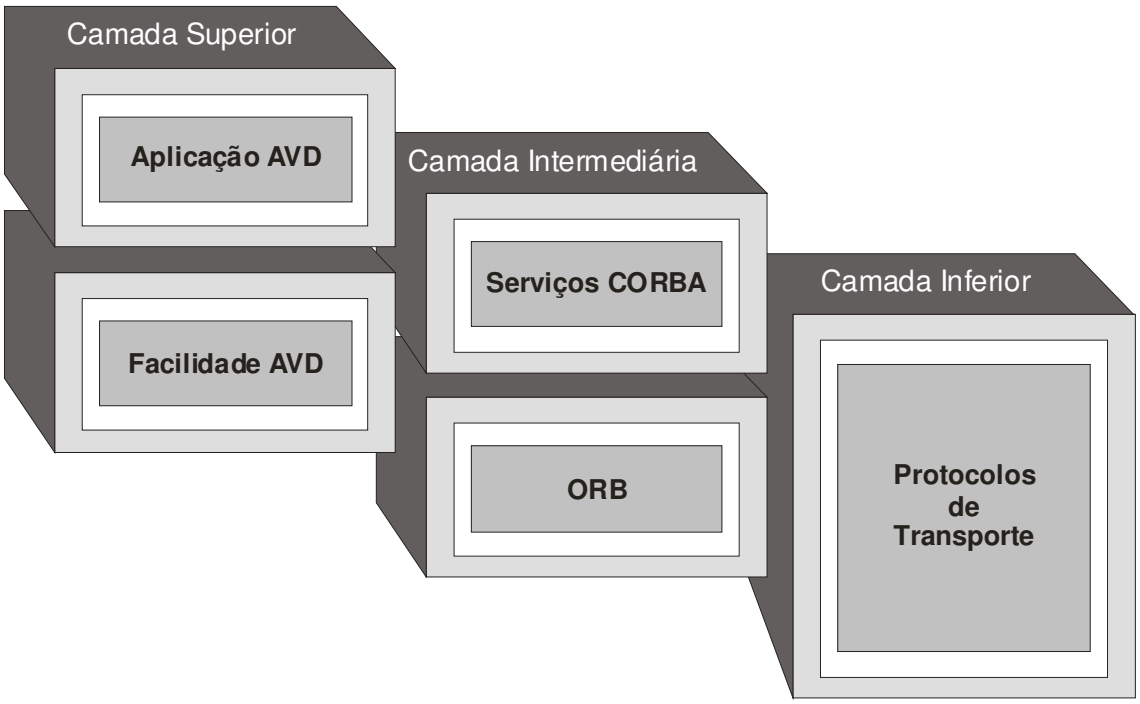


Figura 4.3. Modelo Proposto em Camadas.

Na camada inferior do modelo, considerou-se a existência de protocolos de transporte. Apesar de ser importante numa comunicação de dados, este trabalho não objetiva detalhar esta camada.

Na camada intermediária, encontram-se os serviços CORBA e operações básicas oferecidas pelo ORB. O serviço de nomeação e localização de objetos (*Naming Service*) e o

Canal de Eventos. Para atualização de todas as instâncias de um sistema distribuído, pode-se optar pela consulta que cada cliente pode fazer ao servidor, atualizando seus dados. No caso de AVDs, o sincronismo é fundamental e, portanto, qualquer atualização, que ocorra no sistema, deve ser informada imediatamente a todas as cópias. O serviço de localização de objetos pode permitir que o servidor informe aos clientes, atualizando todos os dados. Dependendo da estrutura definida, o Canal de Eventos pode ser usado para realizar comunicação cliente-cliente, sem a necessidade da interferência do servidor, propiciando um desempenho ainda melhor. Estas operações e serviços estão presentes nesta camada e são fundamentais para o funcionamento do AVD.

Na camada superior, encontram-se a Aplicação RV e uma facilidade criada especificamente para esse modelo, que possibilita sempre a existência de um servidor ativo na rede, possibilitando ao sistema tolerância à falhas. Esta facilidade foi denominada FAVD (Facilidade Ambiente Virtual Distribuído) e está presente em todos os protótipos desenvolvidos.

Para demonstrar a importância dos serviços usados e da facilidade criada, a Figura 4.4 ilustra a arquitetura CORBA original.

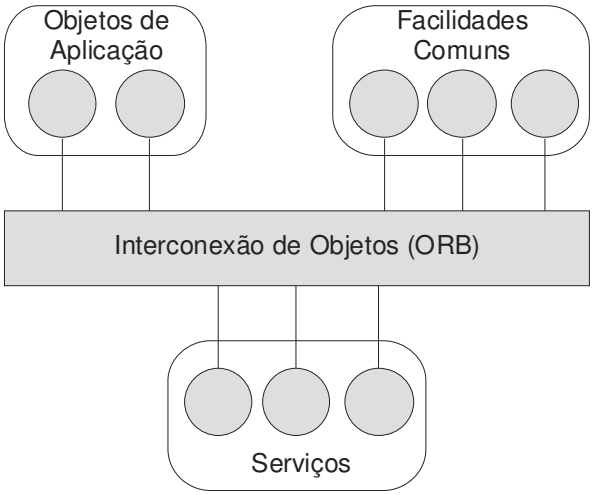


Figura 4.4. Arquitetura CORBA (OMG, 2004).

Na Figura 4.4, fica explícita a divisão que há entre Aplicação, Facilidades e Serviços, que podem também ser vistas através das camadas da Figura 4.3. A Figura 4.5 ilustra a Arquitetura CORBA específica para este modelo, com os objetos Biologia e Química, a FAVD e o Serviço de Replicação.

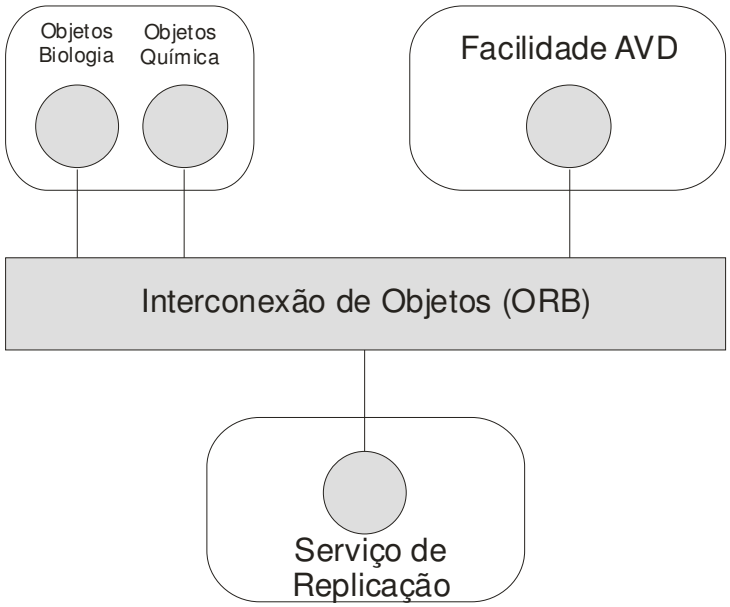


Figura 4.5. Arquitetura CORBA adaptada para este trabalho.

4.3.1. Interface Gráfica com o Usuário - GUI

Esse bloco permite ao usuário visualizar graficamente, de modo interativo e em tempo real, a entrada de dados e a saída de informações. A GUI exibe então o cenário, apresentando os ambientes virtuais modelados por meio da biblioteca gráfica OpenGL dispostos em janelas distintas.

O estudo de caso adotado apresenta o ambiente virtual de Biologia em uma janela e o ambiente virtual de Química em outra janela. O usuário interage no AV de Biologia, aumentando ou diminuindo a temperatura ambiente, e pode visualizar a resposta no AV de Biologia e também no AV de Química. No AV de Química, a interação é realizada pela inserção na cadeia da fotossíntese de uma molécula de Cianeto. Nesse caso, também há uma

resposta nos AVs de Química e de Biologia.

As Figuras 4.6 e 4.7 mostram os dois ambientes virtuais, que serão detalhados no capítulo seguinte.

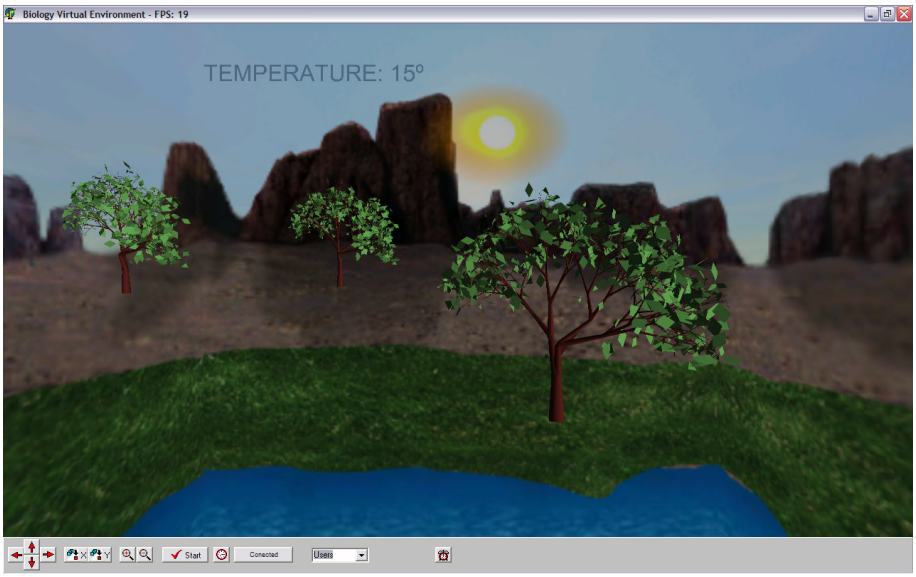


Figura 4.6. Ambiente Virtual de Biologia.

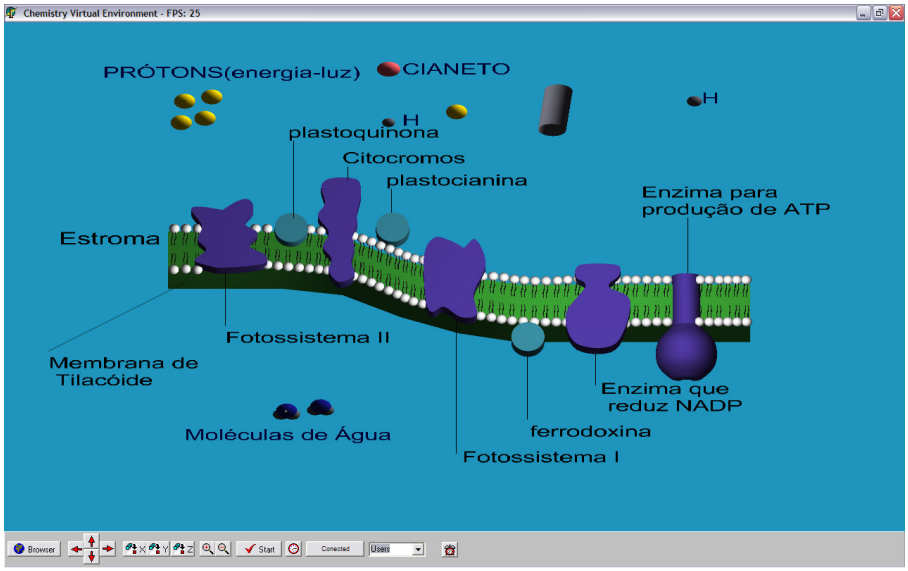


Figura 4.7. Ambiente Virtual de Química.

Para navegação nos ambientes virtuais, foi construída uma barra de opções que possibilita transformações geométricas nos objetos do ambiente, modo de visualização e a

mudança dos atributos de interação do ambiente (reiniciar o ambiente à configuração padrão), como mostra na Figura 4.8.

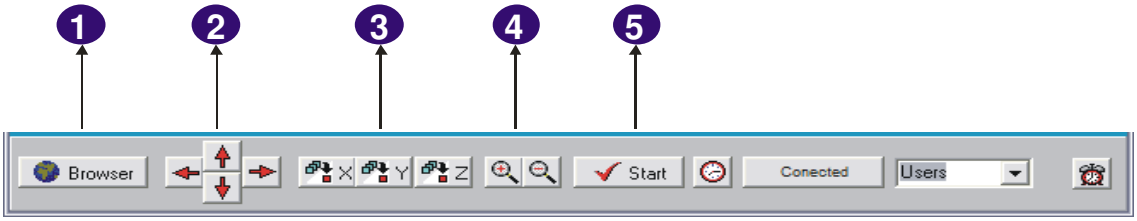


Figura 4.8. Barra de navegação.

De acordo com a Figura 4.6, a barra de navegação possui as seguintes opções:

1 – Botão de configuração para navegação usando o *mouse*. Existem três possibilidades: a) movimento do *mouse* para aproximar ou recuar; b) movimento do *mouse* para girar o ambiente em torno de algum eixo; c) movimento do *mouse* para translado do ambiente;

2 – Botões de navegação de translação nos eixos X e Y.

3 – Botões de navegação de rotação nos eixos X, Y e Z.

4 – Botões de navegação de translação no eixo Z.

5 – Botão para reiniciar o ambiente de acordo com configuração definida como inicial.

Outros botões que estão presentes na barra de navegação são responsáveis por informações e outras configurações pertinentes ao ambiente. O botão "*Browser*" permite configurar o modo de navegação via *mouse*. O botão "*start*" permite reiniciar o modo de visão. O modo de visão usado na aplicação é assíncrono, portanto permite que cada usuário visualize o ambiente de forma exclusiva. Este botão permite ao usuário reiniciar este modo de visão. O usuário também tem a informação, mostrando se a aplicação está conectada a um servidor pelo botão "*conected*", permitindo também fazer a conexão manualmente por meio deste botão. Cada usuário, conectado no sistema, possui um nome apresentado em um *list*

(*users).*

Cabe ressaltar que a tendência dentro da área de Realidade Virtual é de suprimir barras de navegação, propiciando ao usuário uma maior imersão e uma interação mais natural (PINHO, 2002). Porém, devido aos altos custos do uso de sistemas de RV imersivos, sistemas não-imersivos dependem ainda de mecanismos de interação tradicional, como é o caso desta barra de navegação.

4.4. Considerações Finais

A arquitetura, que proporciona um ambiente virtual multidisciplinar aqui proposta, foi construída com base nas características AVDs apresentadas no Capítulo 3 deste trabalho.

A natureza multidisciplinar do sistema foi constituída a partir da separação dos ambientes virtuais de cada disciplina. Além da separação local, a comunicação e interação destes mundos, em computadores separados com localização indefinida, possibilitam uma série de resultados imprevisíveis, enriquecendo ainda mais o resultado produzido pela relação entre os conteúdos.

No próximo capítulo, serão mostrados os detalhes da implementação da arquitetura do sistema proposto.

CAPÍTULO V

5. IMPLEMENTAÇÃO DO SISTEMA

5.1. Introdução

Este capítulo apresenta as implementações de protótipos baseados na metodologia apresentada nos capítulos anteriores. Além de comprovar que a arquitetura CORBA é válida para fornecer suporte intermediário (*middleware)* de comunicação e sincronização aos AVDs, esta tese propõe a escolha da melhor abordagem na utilização da arquitetura.

Essa implementação está dividida em três etapas: 1) o desenvolvimento do ambiente virtual 1; 2) o desenvolvimento do ambiente virtual 2; 3) e a comunicação entre os ambientes, utilizando um modelo de distribuição diferente para cada protótipo.

Também neste capítulo, são apresentadas considerações finais sobre a implementação desse sistema multidisciplinar.

5.2. Desenvolvimento do Ambiente Virtual da Biologia

Esse ambiente virtual, como já foi dito anteriormente, foi modelado com uso da biblioteca gráfica OpenGL, sendo necessário o uso da linguagem de programação Delphi TM 6.0 para compilação do código e implementação da interação do usuário com o mundo e das animações necessárias.

No ambiente virtual de Biologia, encontram-se os objetos Árvore, modelados em por

meio de uma matriz de índices, vértices e normais gerados por um *loader*[10]. A modelagem inicial das árvores foi realizada por meio de um software 3D (3D Studio Max). Além destes objetos, o objeto Sol, foi também inserido por meio de textura, mapeado para simular sua seleção, visando o aumento da temperatura, fator de estímulo, no fenômeno da fotossíntese. A temperatura do mundo de Biologia assume valores que variam de 15ºC a 35ºC, cuja alteração pelo usuário, inicia uma animação nas folhas das árvores, aumentando ou diminuindo o tamanho das mesmas, de acordo com o aumento ou decréscimo da temperatura respectivamente.

A seguir (Figura 5.1), apresenta-se um trecho do código OpenGL, utilizado no desenvolvimento do objeto Árvore. A Figura 5.2 ilustra o ambiente modelado.

```
    // construção do objeto
objects2 : TObject_def2 = (
     (obj_name : 'tree010'; face_indicies : @face_indicies2_0; face_number : MAX_FACE_INDICES2_0;
vertices : @verticies2_0; vertex_number : MAX_VERTICES2_0; normals : @normals2_0; normal_number :
MAX_NORMAL2_0; textures : @textures2_0; texture_number : MAX_TEXTURE2_0; mat_ref :
DEF_MAT_REF2_0)
…

 glNewList(ARVORE1, GL_COMPILE); // início de lista de comandos denominada ARVORE1
 glBegin(GL_TRIANGLES); // modela o objeto com base em triângulos
  for i:= 1 to MAX_OBJECTS2 do  // quantidade de objetos
  begin
    qtdvert := objects2(i).face_number - 1;
    for j:=0 to qtdvert do
    begin
      glNormal3f(normaux^(faciaux^(j,3),0), normaux^(faciaux^(j,3),1), normaux^(faciaux^(j,3),2));
      glVertex3f(vertaux^(faciaux^(j,0),0), vertaux^(faciaux^(j,0),1), vertaux^(faciaux^(j,0),2));
      glNormal3f(normaux^(faciaux^(j,4),0), normaux^(faciaux^(j,4),1), normaux^(faciaux^(j,4),2));
      glVertex3f(vertaux^(faciaux^(j,1),0), vertaux^(faciaux^(j,1),1), vertaux^(faciaux^(j,1),2));
      glNormal3f(normaux^(faciaux^(j,5),0), normaux^(faciaux^(j,5),1), normaux^(faciaux^(j,5),2));
      glVertex3f(vertaux^(faciaux^(j,2),0), vertaux^(faciaux^(j,2),1), vertaux^(faciaux^(j,2),2));
    end;
  end;
 glEnd(); // fim da modelagem
 glEndList(); // fim da lista
```

Figura 5.1. Visão parcial do arquivo da modelagem do ambiente de Biologia.

[10] *Loader* – software que reconhece formatos de extensão de arquivos de modelagem 2D e 3D e os converte para outros formatos de texto. Este *loader* converte arquivos com extensão 3DS para formato texto compatíveis com o código OpenGL. Apesar de existir outros similares, este software foi desenvolvido especificamente para este trabalho. Este *loader* foi adaptado pelo autor do trabalho para produzir formato OpenGL preparados para a linguagem *Object Pascal*.

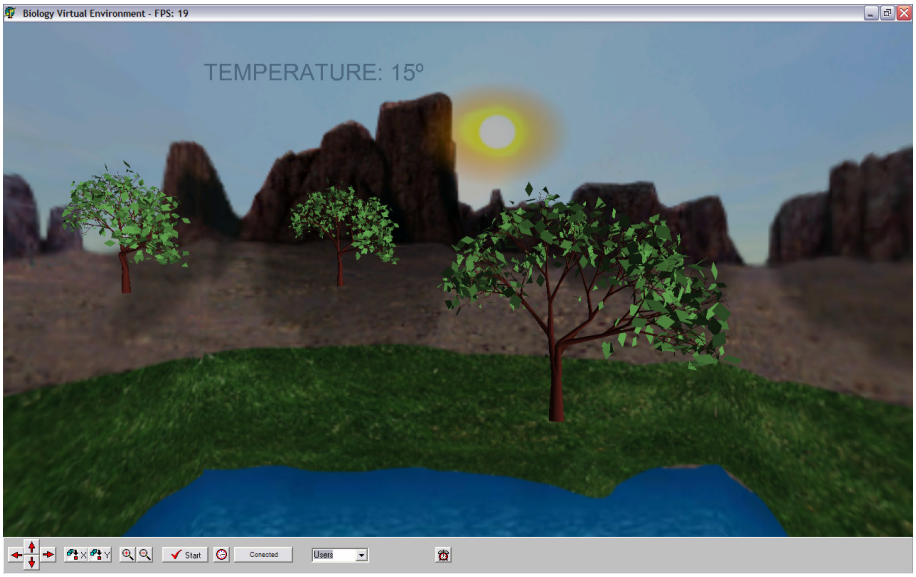


Figura 5.2. Ambiente de Biologia.

Para controlar a incidência de luz solar no ambiente, por meio da mudança de temperatura, foi necessário mudar posição da luz pelo comando *glLightfv* com a diretiva *GL_POSITION,* sendo que cada nível de luminosidade foi definido pela proximidade da luz

Para criar uma sensação de presença dentro do ambiente, foi criado um cubo com seis texturas diferentes (topo: imagem do céu; base: imagem da terra; lados: uma paisagem composta de imagem de terra, montanhas e sol, seccionada em quatro partes), sendo uma para cada lado do cubo. Ao iniciar a aplicação, o usuário é inserido dentro do mesmo e algumas transformações geométricas são tratadas de formas diferentes, impossibilitando a saída do usuário do ambiente. O usuário pode se aproximar das plantas, porém não o faz em relação ao sol e ao horizonte.

5.3. Desenvolvimento do Ambiente Virtual da Química

Para desenvolver a animação da Fotofosforilação[11] Acíclica, foi criada, primeiramente, a estrutura vegetal especializada no processo da fotossíntese, conhecida por

[11] Refere-se ao processo de formação do ATP (trifosfato de adenosina) durante a fotossíntese.

cloroplasto. Esta estrutura foi representada por estroma, membrana interna e externa (LINHARES, 2000), que foi modelada por um software de modelagem 3D (3D Studio Max) e convertida por meio de um *loader* criando um arquivo texto com informações geométricas (índice dos triângulos, coordenadas x, y e z para os vértices, normais e texturas, informações sobre materiais, luzes e câmeras) do objeto para serem usadas com a primitiva *GL_TRIANGLES* da OpenGL. A Figura 5.3 apresenta o código OpenGL utilizado na modelagem do estroma.

```
        // objetos do objeto principal "ESTROMA"
              (obj_name : 'Box010'; face_indicies : @face_indicies_0; face_number :
MAX_FACE_INDICES_0; vertices : @verticies_0; vertex_number : MAX_VERTICES_0; normals :
@normals_0; normal_number : MAX_NORMAL_0; textures : textures_0; texture_number :
MAX_TEXTURE_0; mat_ref : DEF_MAT_REF_0),

        . . . ao todo são 242 objetos

        // função que cria o objeto
     glNewList(Estroma, GL_COMPILE); // início de lista de comandos denominada Estroma
     glBegin(GL_TRIANGLES); // modela o objeto com base em triângulos
   …
        for j:=0 to qtdvert do
        begin
          glNormal3f(normaux^(faciaux^(j,3),0), normaux^(faciaux^(j,3),1), normaux^(faciaux^(j,3),2));
          glVertex3f(vertaux^(faciaux^(j,0),0), vertaux^(faciaux^(j,0),1), vertaux^(faciaux^(j,0),2));

          glNormal3f(normaux^(faciaux^(j,4),0), normaux^(faciaux^(j,4),1), normaux^(faciaux^(j,4),2));
          glVertex3f(vertaux^(faciaux^(j,1),0), vertaux^(faciaux^(j,1),1), vertaux^(faciaux^(j,1),2));

          glNormal3f(normaux^(faciaux^(j,5),0), normaux^(faciaux^(j,5),1), normaux^(faciaux^(j,5),2));
          glVertex3f(vertaux^(faciaux^(j,2),0), vertaux^(faciaux^(j,2),1), vertaux^(faciaux^(j,2),2));
        end;
      end;
     glEnd(); // fim da modelagem
     glEndList(); //  fim da lista

        // função que chama o objeto
          glPushMatrix();
            glDisable(GL_CULL_FACE);
            glRotate(90,1,0,0);
            glRotate(200,1,0,0);
            glScale(0.27,0.2,0.4);
            glTranslate(-35,260,-50);
            glCallList(ESTROMA);
          glPopMatrix();
```

Figura 5.3. Visão parcial do arquivo de modelagem do ambiente de Química.

Os outros objetos que fazem parte do cenário do ambiente virtual de Química foram modelados, assim como também o SOL, no ambiente de Biologia pela biblioteca auxiliar da OpenGL, a GLU (*OpenGL Library Utility*). Estes objetos são moléculas de hidrogênio, cianeto, prótons, elétrons, etc. A Figura 5.4 ilustra o ambiente modelado.

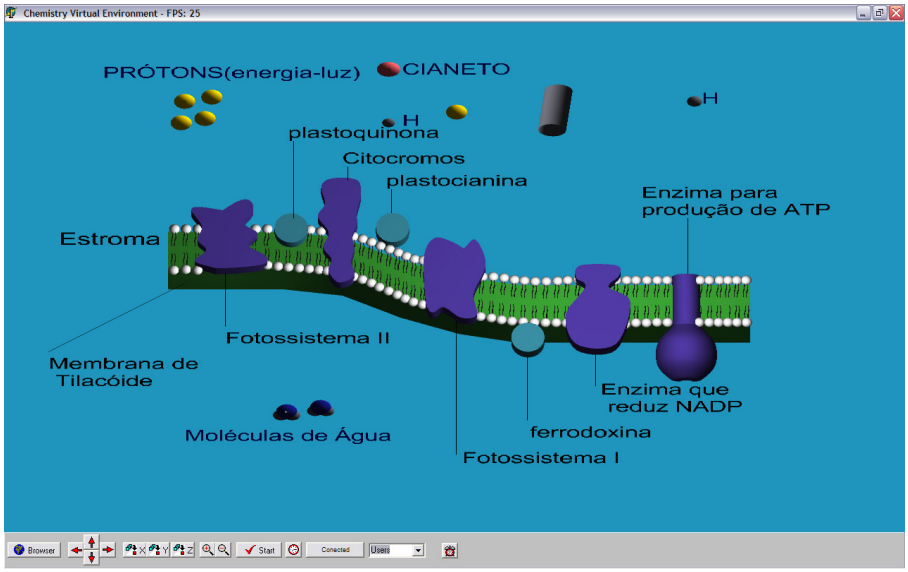


Figura 5.4. Ambiente de Química.

5.4. Implementação dos Protótipos

Os AVDs vêm sendo objeto de estudo por pesquisadores e estudiosos, desde o surgimento da Realidade Virtual. Existem várias arquiteturas que possibilitam ou oferecem suporte à distribuição, como foi mencionado no Capítulo 2. Geralmente, estas arquiteturas são usadas para distribuição de ambientes únicos, mesmo em grande escala, como é o caso de cidades virtuais. Porém, uma mesma arquitetura pode ser implementada, usando abordagens diferentes (Capítulo 3 – itens 3.3.4 a 3.3.7), e, partir destas abordagens, pode-se escolher a melhor baseada em testes e análises. Para possibilitar a definição de uma melhor abordagem, alguns protótipos foram implementados, conforme é mostrado a seguir:

5.4.1. Protótipo 1 (Espelho)

Neste protótipo, foi construída uma aplicação com o Modelo de Dados Distribuído e Cliente/Servidor com a existência de um servidor para cada tipo de ambiente, porém em ordem inversa, servidor Química para *clientes* Biologia e servidor Biologia para *clientes* Química. A plataforma CORBA foi utilizada com seu Serviço de Eventos. A Figura 5.5 ilustra a estrutura do protótipo.

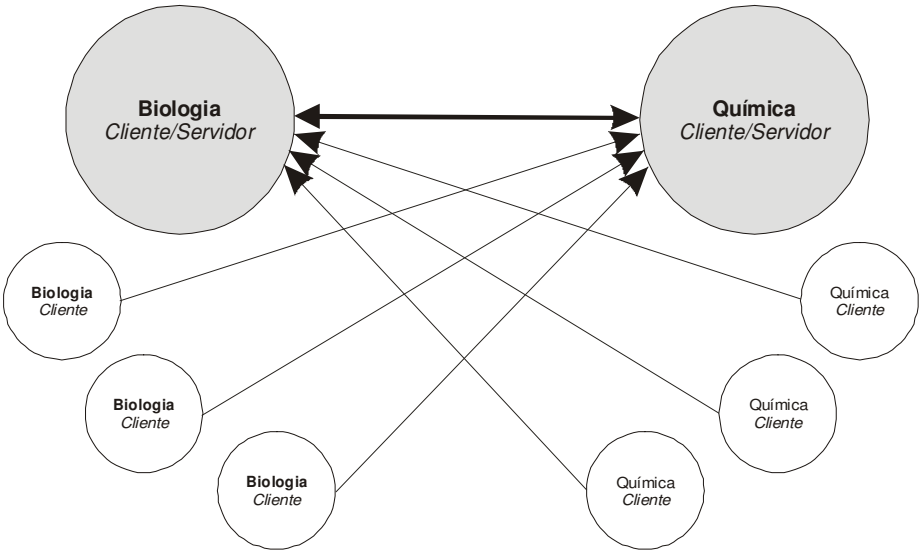


Figura 5.5. Esquema de distribuição dos ambientes virtuais (protótipo 1).

Toda interação realizada pelo usuário, excluindo o modelo de visão, é transformada em valor (estado) e é enviada ao servidor. No protótipo 1, a responsabilidade de armazenar os dados de um determinado ambiente é de uma outra aplicação relacionada a um outro ambiente (cliente Biologia requisita e envia dados para o servidor Química). A distribuição entre os ambientes é iniciada, a partir de uma interação ocasionada pelo usuário. Esta interação é transformada em uma requisição ao servidor. Quando um servidor recebe a requisição, a informação é armazenada e um retorno ao cliente é disparado, ao mesmo tempo outros clientes de ambiente igual ao ambiente do servidor são informados da mudança, e a alteração

é replicada. Quando o cliente, que faz a requisição e recebe o retorno, ambos distribui a informação para os outros clientes de ambientes similares. A requisição sempre é realizada por meio de um método, definido na criação do projeto de interface, presente nos arquivos de implementação. A comunicação cliente-cliente é possibilitada por meio do Serviço de Eventos (Canal de Eventos), tornando as ações de um cliente replicáveis a outros. Nesse exemplo, o ambiente virtual de Química (servidor) avisará todos os seus clientes (Biologia) da nova posição que o ambiente deve estar (aumento das folhas e da luminosidade). Por outro lado, o servidor de Biologia avisará todos os clientes Química que o processo da Fotofosforilação Acíclica deve ser disparado.

Independente da quantidade de cópias dos ambientes haverá sempre um cliente/servidor de cada ambiente. Todos os outros serão apenas clientes. Biologia será cliente de Química e vice-versa. Para que isto aconteça, a primeira aplicação iniciada em uma rede de computadores, sempre procura um servidor contrário (Biologia procura Química). Caso o encontre, se conecta, ou caso contrário, transforma-se em módulo isolado. Depois, tenta localizar outro servidor do mesmo ambiente (Biologia procura Biologia). Quando não encontra, se transforma em servidor. Uma próxima aplicação a ser iniciada fará o mesmo processo. Abaixo, está o código, que possibilita esta ação descrita acima. A Figura 5.6 mostra o algoritmo da ativação dos servidores e da conexão dos clientes do protótipo 1.

```
// procedimento onde a aplicação verifica se existe um servidor para o seu modelo de ambiente
// neste caso a aplicação é um AV de Química
Procurar servidor de ambiente virtual de Química
Se não encontrar
    Iniciar aplicação como servidor do ambiente

// procedimento para localizar o servidor contrário
Verificar se existe algum servidor Biologia
Se não existir
   iniciar a aplicação isoladamente
Senão
   Conectar ao servidor de Biologia como cliente
```

Figura 5.6. Algoritmo de ativação de servidores e conexão de clientes.

Quando um dos servidores é desligado, o servidor contrário recebe esta informação e avisa a um de seus clientes para que assuma o posto de servidor, informando-lhe a configuração de seus clientes que fica armazenada no servidor.

Vantagens:

1 – Flexibilidade: qualquer aplicação pode ser cliente e/ou servidora.

2 – A tarefa de servir é distribuída, a responsabilidade em armazenar os dados dos clientes é dividida em dois servidores.

3 – Facilidade na comunicação cliente-cliente, possibilitando replicação com mais rapidez.

Desvantagens:

1 – Complexidade na criação do algoritmo de comunicação entre clientes e servidores. Devido a sua característica morfológica, cada aplicação pode em um determinado momento ter a responsabilidade de: a) servir; b) servir e solicitar; c) solicitar. Desta forma, o algoritmo de comunicação torna-se mais extenso e complicado.

2 – Necessidade de um algoritmo de tolerância à falhas (FAVD) mais robusto. Por não existir um servidor centralizado e dedicado, quando ocorre alguma falha, é necessário eleger uma aplicação qualquer para que se transforme no servidor e avise aos clientes da mudança.

3 – A aplicação servidora perde no quesito desempenho, por estar responsável pela renderização dos objetos e também servindo a outros clientes.

4 – Permite apenas a comunicação em pares, dificultando a extensibilidade. Para inserir um novo ambiente virtual, por exemplo Física, seria necessária alteração na abordagem adotada no protótipo.

5.4.2. Protótipo 2 (Dedicado)

Nesse protótipo, foi construída uma aplicação com o Modelo de Dados Centralizada e Compartilhada com a existência de um servidor único para qualquer ambiente participante da aplicação. A plataforma CORBA foi usada sem seu Serviço de Eventos. Utilizou-se, aqui, apenas o modelo cliente/servidor com o serviço de localização de objetos. A Figura 5.7 ilustra a estrutura de funcionamento do protótipo.

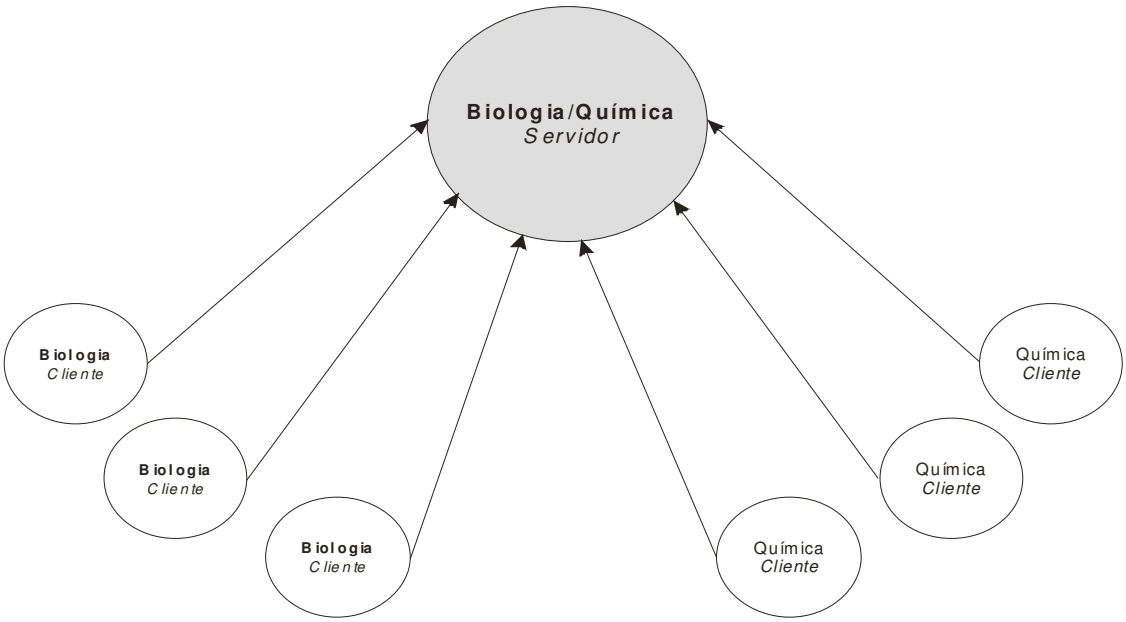


Figura 5.7. Esquema de distribuição dos ambientes virtuais (protótipo 2).

Da mesma forma que no protótipo 1, toda interação realizada pelo usuário, excluindo o modelo de visão, é transformada em valor (estado) e enviada ao servidor. No protótipo 2, a responsabilidade de armazenar os dados de qualquer ambiente existente na rede é de uma aplicação servidora, única e exclusiva a esta tarefa. A comunicação é iniciada por meio de uma interação transformada em uma requisição ao servidor. Quando o servidor recebe a requisição, a informação recebida é armazenada e um retorno ao cliente é disparado e, ao mesmo tempo, todos os outros clientes, independente do ambiente, são atualizados com a

replicação da alteração.

Independente da quantidade de cópias dos ambientes, haverá sempre um servidor responsável por todos os ambientes (Biologia e Química). Todas as outras aplicações serão apenas clientes. Para que isto aconteça, a primeira aplicação iniciada em uma rede de computadores sempre procura um servidor. Caso encontre, se conecta, caso contrário, a própria aplicação cliente ativa o servidor. Quando não encontra, se transforma em servidor. Uma próxima aplicação a ser iniciada fará o mesmo processo. Abaixo, está o código que possibilita esta ação descrita acima. A Figura 5.8 mostra o algoritmo da ativação do servidor e da conexão dos clientes do protótipo 2.

```
// procedimento onde a aplicação verifica se existe um servidor
// neste caso a aplicação pode ser um AV de Química ou Biologia
Procurar servidor de ambientes
Se não encontrar
        Iniciar a aplicação servidora
Senão
        Conectar-se ao servidor
```

Figura 5.8. Algoritmo de ativação de servidores e conexão de clientes.

Quando o servidor é desligado, qualquer outra aplicação poderá ativar um outro servidor.

Vantagens:

1 – Facilidade na criação do algoritmo de comunicação e no algoritmo de tolerância à falhas (FAVD). Diferente do protótipo 1, a aplicação cliente do protótipo 2 apenas solicita, e a aplicação servidora somente serve. Se algum cliente falha, não afeta o servidor, e, quando o servidor falha, qualquer cliente pode solicitar a ativação de um outro servidor atualizando-o.

2 – Ideal para pequenos mundos com pouca troca de mensagens. O servidor é dedicado à função de servir. Para ambientes com poucos clientes, onde não há sobrecarga de tarefas, o servidor se porta muito bem.

3 – Permite facilmente a inserção de ambientes relacionados a outras áreas, por exemplo, o conteúdo de Física.

Desvantagens:

1 – Grande fluxo de E/S centralizada em um único servidor, causando queda no desempenho.

2 – O servidor torna-se rapidamente um "gargalo", quando há um aumento de participantes (escalabilidade).

3 – Não recomendada para ambientes de grande escala (onde muitas informações são armazenadas) e ambientes com grande número de participantes.

5.4.3. Protótipo 3 (Encadeado)

Nesse protótipo, foi construída uma aplicação com o Modelo de Dados Distribuída e Cliente/Servidor com a existência de um servidor para cada tipo de ambiente, em ordem direta, servidor Química para *clientes* Química e vice-versa, e com comunicação entre servidores. A plataforma CORBA foi utilizada com seu Serviço de Eventos. A Figura 5.9 ilustra a estrutura de funcionamento do protótipo 3.

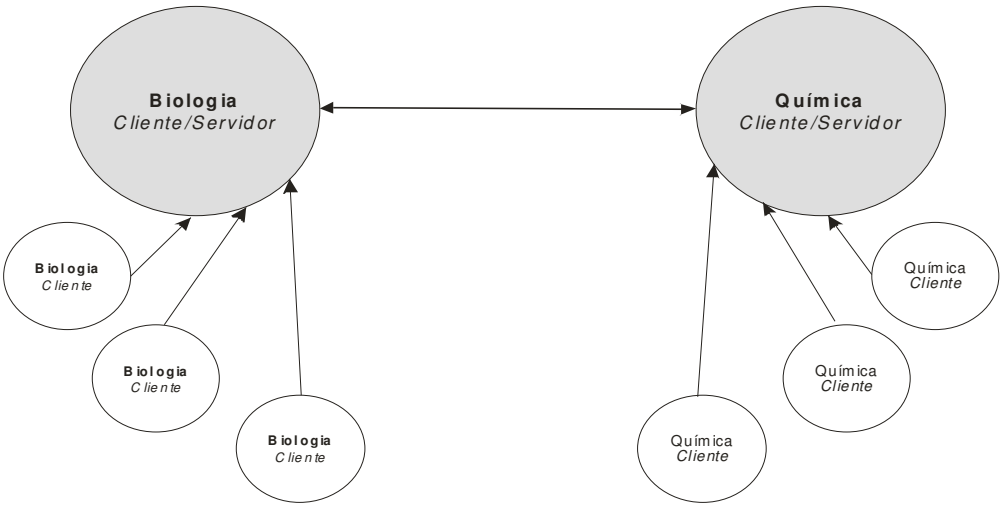


Figura 5.9. Esquema de distribuição dos ambientes virtuais (protótipo 3).

No protótipo 3, a responsabilidade de armazenar os dados de um determinado tipo de ambiente é de um servidor, portanto, se existir dois tipos de ambientes (Biologia e Química), existirão dois servidores. O protótipo 3 é similar ao protótipo 2, porém, cada conjunto de ambientes possui um servidor, e estes servidores se comunicam. Quando um servidor recebe uma requisição, a informação recebida é armazenada, um retorno ao cliente é disparado e uma outra requisição é feita a outro servidor. Desta forma, todos os clientes são atualizados. A comunicação cliente-cliente é possibilitada por meio do Serviço de Eventos (canal de eventos), tornando as ações de um cliente replicáveis a outros.

Independente da quantidade de cópias dos ambientes haverá sempre um servidor de cada ambiente. Todos os outros serão apenas clientes. Para que isto aconteça, a primeira aplicação iniciada em uma rede de computadores sempre procura por um servidor. Caso encontre, se conecta, caso contrário provoca a ativação de um servidor. O servidor tenta localizar um outro, quando alguma requisição é efetuada, caso não consiga o conjunto de ambientes, terá uma comunicação apenas entre os mesmos. Abaixo, está o código que possibilita esta ação descrita acima. A Figura 5.10 mostra o algoritmo da ativação dos servidores e da conexão dos clientes do protótipo 3.

```
// procedimento onde a aplicação verifica se existe um servidor para o seu modelo de ambiente
// neste caso a aplicação é um AV de Química
Procurar servidor de ambiente virtual de Química
Se não encontrar
        Iniciar a aplicação servidora


// procedimento para localizar o servidor contrário
Verificar se existe algum servidor Biologia
Se não existir
   iniciar a aplicação isoladamente
Senão
   Conectar ao servidor de Biologia como cliente
```

Figura 5.10. Algoritmo de ativação de servidores e conexão de clientes.

Quando um dos servidores é desligado, se houver algum cliente, este pode ativar novamente um outro servidor e atualizá-lo.

Vantagens:

1 – Possibilita o funcionamento da aplicação com apenas um tipo de ambiente. Por exemplo, caso seja necessário o funcionamento apenas do ambiente de Biologia, este protótipo possui estrutura mais propícia, pois funciona de forma encadeada.

2 – Distribuição na tarefa de servir, a responsabilidade em armazenar dados é dividida em dois servidores.

Desvantagens:

1 – Permite apenas a comunicação em pares, dificultando a extensibilidade. Para inserir um novo ambiente virtual, por exemplo a área de Física, seria necessária uma alteração na abordagem adotada no protótipo.

5.4.4. Protótipo 4 (Dividido)

Nesse protótipo, foi construída uma aplicação com o Modelo de Dados Distribuída, Compartilhada e Cliente/Servidor, com a existência de um servidor que atende a qualquer tipo de cliente. A plataforma CORBA foi usada sem seu Serviço de Eventos. Utilizou-se, aqui, apenas o modelo cliente/servidor com o serviço de localização de objetos. A Figura 5.11 ilustra a estrutura de funcionamento do protótipo 4.

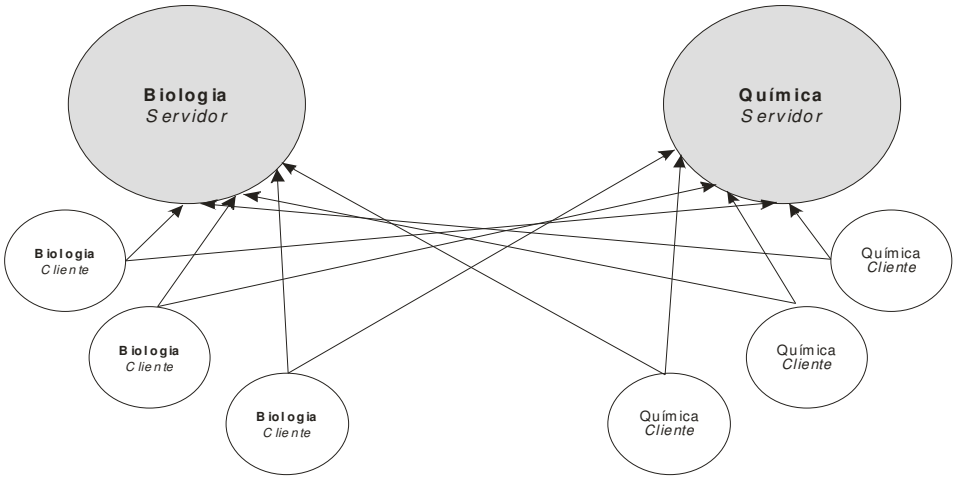


Figura 5.11. Esquema de distribuição dos ambientes virtuais (protótipo 4).

O protótipo 4 é similar ao protótipo 3; a diferença é em relação ao servidor, que serve tanto a clientes Biologia quanto cliente Química e conseqüentemente não há ligação entre os servidores. A distribuição entre os ambientes é iniciada, a partir de uma interação ocasionada por um usuário, sendo que esta interação é transformada em uma requisição ao servidor, e neste caso a requisição é enviada a dois servidores ao mesmo tempo, atualizando os mesmos e provocando a replicação para os demais clientes.

Independente da quantidade de cópias dos ambientes haverá sempre um servidor de cada ambiente. Todos os outros serão apenas clientes. Para que isto aconteça, a primeira aplicação iniciada em uma rede de computadores sempre ativa um servidor e não necessariamente haverá dois servidores, o que ocorrerá somente se existirem dois tipos de ambientes. Abaixo, está o código que possibilita esta ação descrita acima. A Figura 5.12 mostra o algoritmo da ativação dos servidores e da conexão dos clientes do protótipo 4.

```
// procedimento onde a aplicação verifica se existe um servidor
// neste caso a aplicação é um AV de Química
Procurar servidor de ambiente virtual de Química
Se não encontrar
        Iniciar uma aplicação como servidora do ambiente
Procurar servidor de ambiente virtual de Biologia
Se encontrar
        Conectar-se ao servidor
```

Figura 5.12. Algoritmo de ativação de servidores e conexão de clientes.

Quando um dos servidores é desligado, uma aplicação cliente pode ativar novamente um outro servidor, atualizando-o.

Vantagens:

1 – Possibilita o funcionamento da aplicação com apenas um tipo de ambiente.

2 – Distribuição na tarefa de servir, a responsabilidade em armazenar dados é dividida em dois servidores.

3 – Facilidade na comunicação cliente-cliente, possibilitando uma replicação com mais rapidez.

4 – Facilidade na extensibilidade, desde que o projeto faça previsão para a existência de outros ambientes.

Desvantagens:

1 – Apesar de facilitar a extensibilidade, a inserção de novos ambientes provoca "gargalos" na comunicação, devido à necessidade do cliente informar a todos os servidores.

2 – Não recomendada para ambientes de grande escala (nos quais muitas informações são armazenadas) e ambientes com grande número de participantes.

5.4.5. Pré-requisitos

Para desenvolvimento dos ambientes e a posterior distribuição dos mesmos, foi seguida uma metodologia de desenvolvimento que consiste em:

a) - **Verificação das necessidades de cada ambiente**: esta é uma fase importante no desenvolvimento de um AVD, pois o projeto de interface (IDL – *Interface Definition Language*) depende diretamente desta verificação. Um projeto de interfaces conciso evita a recompilação e a reescrita de código, e um ciclo de vida longínquo. Esta análise pode ser feita usando também mecanismos da Qualidade de Software (ISO, 1991), que se preocupa com o sucesso do projeto e implantação de softwares. Para possibilitar uma extensibilidade no modelo, as interfaces são orientadas a objeto, permitindo herança. A IDL é independente de implementação; é uma linguagem declarativa e não uma linguagem de programação. A IDL tem a aparência da linguagem ANSI C++.

b) - **Verificação das especificidades das operações dos clientes e suas conseqüências na aplicação servidora:** foram construídos quatro modelos de relação cliente/servidor. Para cada modelo, o servidor se comporta de

forma específica. Para tornar cada modelo tolerante a falhas[12], foi construído um algoritmo de gerenciamento de servidores com adoção de sistemas de redundâncias.

Para caracterizar o objeto de estudo deste trabalho, foi necessário classificá-lo de acordo com os itens 3.3.4 (Modelo de Visão), 3.3.5 (Modelo de Dados), 3.3.6 (Gerenciamento da Computação) e 3.3.7 (Comportamento dos Objetos), enquadrando-o na conceituação dos mesmos.

PROTÓTIPO 1 (Espelho):

Em relação ao Modelo de Visão: **Assíncrona.**

Em relação ao Modelo de Dados: **Distribuída e Cliente/Servidor**.

Em relação ao Gerenciamento da Computação: **Distribuição Parcial**.

Em relação ao Comportamento dos Objetos: **Estático**.

PROTÓTIPO 2 (Dedicado):

Em relação ao Modelo de Visão: **Assíncrona.**

Em relação ao Modelo de Dados: **Centralizada e Compartilhada**.

Em relação ao Gerenciamento da Computação: **Distribuição Parcial**.

Em relação ao Comportamento dos Objetos: **Estático**.

PROTÓTIPO 3 (Encadeado):

Em relação ao Modelo de Visão: **Assíncrona.**

Em relação ao Modelo de Dados: **Distribuída e Cliente/Servidor.**

Em relação ao Gerenciamento da Computação: **Distribuição Parcial**.

Em relação ao Comportamento dos Objetos: **Estático**.

PROTÓTIPO 4 (Dividido):

[12] Quando um servidor for desligado, para não haver perdas e o acionamento de outro servidor ser imediato; permitir por meio de redundâncias a proteção dos dados; possibilitar a replicação em todos os ambientes apenas quando não houver falhas na comunicação.

Em relação ao Modelo de Visão: **Assíncrona.**

Em relação ao Modelo de Dados: **Distribuída, Compartilhada e Cliente/Servidor.**

Em relação ao Gerenciamento da Computação: **Distribuição Parcial**.

Em relação ao Comportamento dos Objetos: **Estático**.

A maior diferença entre os quatro protótipos é em relação ao Modelo de Dados. Para possibilitar estas relações cliente/servidor deste projeto, alguns aspectos do CORBA foram importantes: as Facilidades CORBA e os Serviços CORBA. O FAVD, já descrito anteriormente, é um algoritmo criado para este projeto (todos os protótipos) que possibilita a aplicação desta metodologia de AVD para mundos multidisciplinares. Para cada um dos protótipos foi necessária implementação de recursos e serviços diferentes para o FAVD. Os Serviços CORBA, especificamente Serviço de Nomeação de Objetos (***Naming Service)*** e Serviço de Eventos (Canal de Eventos), foram fundamentais na implementação dos protótipos. A Tabela 5.1 mostra as principais características dos modelos propostos, que podem ser comparadas com os sistemas avaliados no Capítulo 3 (Tabela 3.1).

Tabela 5.1. Principais características dos protótipos propostos.

CARACTERÍSTICAS						
Protótipos	Mecanismo de Transporte	Gerenciamento de Dados	Gerenciamento da Computação	Múltiplos Mundos Virtuais	Suporte ao Usuário	Ambiente de Rede
1 – ESPELHO	Multicast	Distribuída	Distribuição Parcial	Suporta	Múltiplos	Heterogêneo
2 – DEDICADO	Ponto a ponto	Centralizada	Distribuição Parcial	Suporta	Múltiplos	Heterogêneo
3 – ENCADEADO	Multicast	Distribuída	Distribuição Parcial	Suporta	Múltiplos	Heterogêneo
4 - DIVIDIDO	Ponto a ponto	Distribuída/ Compartilhada	Distribuição Parcial	Suporta	Múltiplos	Heterogêneo

5.5. Comunicação e Distribuição dos Ambientes Virtuais

Na implementação da distribuição e comunicação entre os AVs foi usado o ambiente

de programação Borland Delphi TM 6.0 Enterprise com suporte ao Inprise Visibroker 4.0 (TEIXEIRA, 2002). A Figura 5.13 mostra um diagrama de blocos da arquitetura CORBA. A parte comum para cliente e servidor é o ORB. O ORB trata de toda comunicação entre os objetos. Além de cuidar de todo o tráfego de mensagens, o ORB também corrige variações de plataforma.

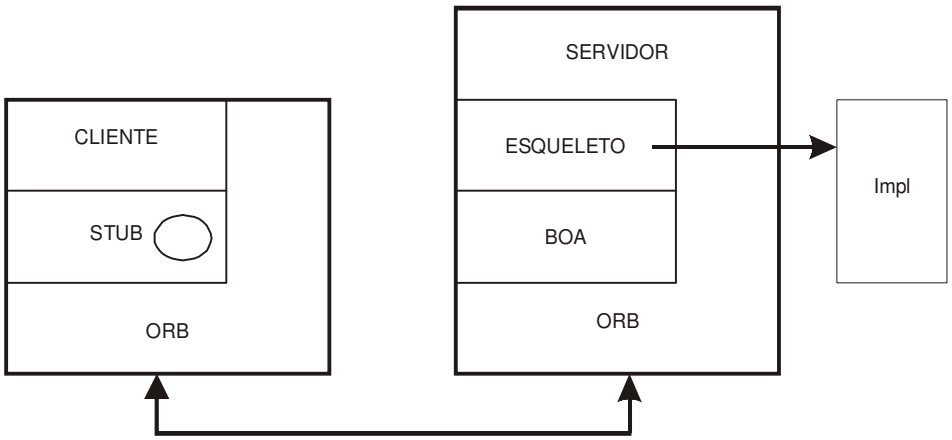


Figura 5.13. Arquitetura CORBA (TEIXEIRA, 2002).

De acordo com a Figura 5.13, o lado do cliente consiste em duas camadas. O bloco cliente é a aplicação que foi escrita pelo desenvolvedor. A parte mais interessante é o *stub*. O *stub* é um arquivo gerado automaticamente pelo compilador IDL2Pas incluído no Delphi Enterprise. Sua finalidade é apanhar arquivos que descrevem as interfaces do servidor e gerar código em *Object Pascal,* que possa interagir com o ORB da arquitetura CORBA. O arquivo de *stub* contém uma ou mais classes que "espelham" o servidor CORBA. As classes contêm as mesmas interfaces publicadas e tipos de dados que são expostos pelo servidor. O cliente usa classes *stub* a fim de se comunicar com o servidor. As classes *stub* atuam como um *proxy* para os objetos servidores. O símbolo arredondado no bloco de *stub* representa uma conexão com o servidor. A conexão é estabelecida por meio de uma chamada de ligação emitida pelo cliente. O *stub* invoca a chamada, por meio de sua referência ao objeto servidor, usando o ORB. Quando o servidor responde, a classe *stub* recebe a mensagem do ORB e entrega a

resposta de volta ao cliente. O cliente também pode chamar algumas funções diretamente no ORB, como mostra o diagrama de blocos.

O lado servidor contém uma interface ORB chamada de adaptador de objeto básico (*Basic Object Adaptor* – BOA). O BOA é responsável por rotear as mensagens do ORB para a interface de esqueleto. O esqueleto é uma classe gerada pelo compilador IDL2Pas, assim como o *stub*. O esqueleto contém uma ou mais classes que publicam as interfaces CORBA no lado do servidor. Existe outro arquivo, que contém as classes que representam os detalhes funcionais do servidor. Este é conhecido como arquivo IMPL (implementação).

Quando uma mensagem chega ao lado do servidor, o ORB passa um *buffer* de mensagem ao BOA, que, por sua vez, passa o *buffer* à classe do esqueleto. O esqueleto desembrulha os dados do *buffer* e determina qual método deve ser chamado no arquivo IMPL. O esqueleto apanha os resultados de retorno e os conduz de volta ao cliente. (TEIXEIRA, 2002).

Caso existisse apenas um tipo de ambiente virtual (unidisciplinar) e fosse necessário distribuí-lo por meio de replicação ou particionamento, uma metodologia possível seria implementar as duas partes (cliente e servidor), sendo que o cliente seria a aplicação do ambiente virtual, e o servidor ficaria responsável por controlar os clientes e as informações dos mesmos. A primeira aplicação iniciada na rede requisitaria um servidor. Outras aplicações iniciadas usariam o mesmo servidor.

Os protótipos 2 e 4 seguem essa arquitetura, em que o cliente é aplicação RV e o servidor é responsável pela distribuição e conexão dos clientes.

Os protótipos 1 e 3 usam uma arquitetura um pouco diferente da convencional, em que a aplicação tem poderes para ser cliente e/ou servidor, ao mesmo tempo. No caso do protótipo 3, apenas o servidor tem esse poder. O que enriquece a arquitetura é o fato de se

tratar de dois ambientes distintos (Biologia e Química). Além da interferência que cada ambiente sofre do outro, é necessário replicar para cada instância do ambiente toda alteração relevante. Para possibilitar a implementação deste modelo, uma nova estrutura foi construída em que um objeto (ambiente virtual) passa se comportar como cliente e servidor, de acordo com a necessidade do mesmo.

A Figura 5.14 ilustra uma adaptação do diagrama de blocos, apresentando objetos que podem ser ao mesmo tempo cliente e servidor, usado nos protótipos 1 e 3.

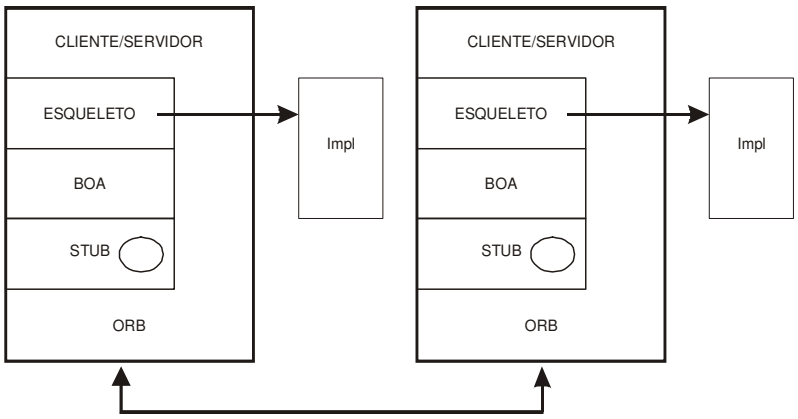


Figura 5.14. Adaptação da arquitetura CORBA cliente/servidor para os protótipos 1 e 3.

Para todos os protótipos, a implementação está dividida em duas fases: a) criação de uma interface para o objeto, pois todo objeto CORBA é descrito por sua interface; b) segundo, escrever o código respectivo a cada método declarado na interface. A aplicação servidora necessita publicar declarações de tipo, interfaces e métodos específicos num projeto puramente orientado a objeto. Esta descrição de interface foi escrita na linguagem de Definição de Interface (IDL – *Interface Definition Language*) e traduzida para a linguagem específica (no caso deste estudo, o *Object Pascal),* por meio de um compilador (termo mais apropriado seria gerador de código), o IDL2Pas. A seguir (Figura 5.15), segue o código das interfaces dos ambientes de Biologia e Química, de cada protótipo, respectivamente.

```
//// PROTÓTIPO 1 -  Interface do ambiente virtual de Biologia
module Biologia {
```

```
        interface AVDB {  // interface é uma definição de classe que contém apenas métodos
          long interfere(in long posicao);
          long situacao();
          void usuario(in string nome);
          string listusers();
          string latencia(in string pacote);
        };
};
// Interface do ambiente virtual de Química
module Quimica {
        interface AVDQ { // interface é uma definição de classe que contém apenas métodos
            long interfere(in long posicao);
            long situacao();
            void usuario(in string nome);
            string listusers();
            string latencia(in string pacote);
        };
};
//// PROTÓTIPO 2 -  Interface do ambiente virtual de Biologia e Química
module BioQui {
        interface AVD { // interface é uma definição de classe que contém apenas métodos
          long interfereb(in long posicao);
          long interfereq(in long posicao);
          long situacaob();
          long situacaoq();
          void usuario(in string nome);
          string listusers();
          string latencia(in string pacote);
        };
};
//// PROTÓTIPO 3 -  Interface do ambiente virtual de Biologia
module Biologia {
        interface AVB { // interface é uma definição de classe que contém apenas métodos
          long interfereb(in long posicao);
          long situacaob();
          void usuario(in string nome);
          string listusers();
          string latencia(in string pacote);
        };
};
// Interface do ambiente virtual de Química
module Quimica {
        interface AVQ {
            long interfereq(in long posicao);
            long situacaoq();
            void usuario(in string nome);
            string listusers();
            string latencia(in string pacote);
        };
};
 //// PROTÓTIPO 4 -  Interface do ambiente virtual de Biologia
module Biologia {
        interface AVB {
          long interfereb(in long posicao);
          long situacaob();
          void usuario(in string nome);
          string listusers();
          string latencia(in string pacote);
```

```
        };
    };
    // Interface do ambiente virtual de Química
    module Quimica {
        interface AVQ {
            long interfereq(in long posicao);
            long situacaoq();
            void usuario(in string nome);
            string listusers();
            string latencia(in string pacote);
        };
    };
```

Figura 5.15. Visão dos arquivos Biologia.idl e Química.idl.

As interfaces necessárias ao projeto foram processadas gerando os seguintes arquivos: Biologia_I.pas, Química_I.pas e BioQui_I.pas – Arquivos que contêm todas as interfaces e definições de tipo (Figura 5.16).

```
// arquivo Biologia
    ...
    type
      AVDB = interface;
      AVDB = interface
        ('{7987A03C-E9E5-4E25-A357-D1C3491A9792}')
        function  interfere (const posicao : Integer): Integer;
        function  situacao : Integer;
        procedure usuario (const nome : AnsiString);
      end;

// arquivo Quimica
    …
    type
      AVDQ = interface;
      AVDQ = interface
        ('{EBBFEB05-CDCE-C124-93FC-156257D8A0B6}')
        function  interfere (const posicao : Integer): Integer;
        function  situacao : Integer;
        procedure usuario (const nome : AnsiString);
      end;
```

Figura 5.16. Visão parcial dos arquivos de interface.

Biologia_C.pas, Quimica_C.pas e BioQui_C.pas – arquivos que contêm quaisquer tipos definidos pelo usuário, exceções e classes *stub* do cliente (Figura 5.17).

```
// arquivo Biologia
    …
    type
      EFolhasError = class;
      TAVDBHelper = class;
      TAVDBStub = class;
```

```
            EFolhasError = class(UserException)
            private
              Fsit_atual : Integer;
            protected
              function  _get_sit_atual : Integer; virtual;
            public
              property  sit_atual : Integer read _get_sit_atual;
              constructor Create; overload;
              constructor Create(const sit_atual : Integer); overload;
              procedure Copy(const _Input : InputStream); override;
              procedure WriteExceptionInfo(var _Output : OutputStream); override;
            end;
          …

// arquivo Quimica
          ...
          type
            ECianetoError = class;
            TAVDQHelper = class;
            TAVDQStub = class;
            ECianetoError = class(UserException)
            private
              Fsit_atual : Integer;
            protected
              function  _get_sit_atual : Integer; virtual;
            public
              property  sit_atual : Integer read _get_sit_atual;
              constructor Create; overload;
              constructor Create(const sit_atual : Integer); overload;
              procedure Copy(const _Input : InputStream); override;
              procedure WriteExceptionInfo(var _Output : OutputStream); override;
            end;
          …
```

Figura 5.17. Visão parcial dos arquivos *stub.*

Biologia_S.pas, Quimica_S.pas e BioQui_S.pas – Esses arquivos possuem o esqueleto das definições de classe no servidor (Figura 5.18).

```
// arquivo Biologia
          ...
          type
            TAVDBSkeleton = class;

            TAVDBSkeleton = class(CORBA.TCorbaObject, Biologia_i.AVDB)
            private
              FImplementation : AVDB;
            public
              constructor Create(const InstanceName: string; const Impl: AVDB);
              destructor Destroy; override;
              function GetImplementation : AVDB;
              function  interfere ( const posicao : Integer): Integer;
              function  situacao : Integer;
              procedure usuario ( const nome : AnsiString);
            published
              procedure _interfere(const _Input: CORBA.InputStream; _Cookie: Pointer);
```

```
            procedure _situacao(const _Input: CORBA.InputStream; _Cookie: Pointer);
            procedure _usuario(const _Input: CORBA.InputStream; _Cookie: Pointer);
          end;
        …

// arquivo Quimica
        …
        type
          TAVDQSkeleton = class;

          TAVDQSkeleton = class(CORBA.TCorbaObject, Quimica_i.AVDQ)
          private
            FImplementation : AVDQ;
          public
            constructor Create(const InstanceName: string; const Impl: AVDQ);
            destructor Destroy; override;
            function GetImplementation : AVDQ;
            function  interfere ( const posicao : Integer): Integer;
            function  situacao : Integer;
            procedure usuario ( const nome : AnsiString);
          published
            procedure _interfere(const _Input: CORBA.InputStream; _Cookie: Pointer);
            procedure _situacao(const _Input: CORBA.InputStream; _Cookie: Pointer);
            procedure _usuario(const _Input: CORBA.InputStream; _Cookie: Pointer);
          end;
```

Figura 5.18. Visão parcial dos arquivos *skeleton.*

Biologia_Impl.pas, Quimica_Impl.pas e BioQui_Impl.pas – Esses arquivos possuem uma definição de classe geral para uma aplicação no servidor. Nestes arquivos foram inseridos códigos para realizar as ações necessárias aos clientes e que eram responsabilidade do servidor (Figura 5.19).

```
// arquivo Biologia_pas
        ...
        type
          tusuarios = record
            iduser : integer;
            nomeuser : string;
            local : string;
            conect : boolean;
          end;
        type
          TAVDB = class;

          TAVDB = class(TInterfacedObject, Biologia_i.AVDB)
          protected
            function posicao(): integer;
          public
            _usuarios : array(1..50) of tusuarios;
            constructor Create;
```

```
            function  interfere (const   posicao : Integer): Integer;
            function  situacao : Integer;
            procedure usuario ( const nome : String);
          end;
        …
// arquivo Quimica
        ...
        type
          tusuarios = record
            iduser : integer;
            nomeuser : string;
            local : string;
            conect : boolean;
          end;
        type
          TAVDQ = class;

          TAVDQ = class(TInterfacedObject, Quimica_i.AVDQ)
          protected
            _posicao : Integer;
            _usuarios : array(1..50) of tusuarios;
            function posicao(): integer;
          public
            constructor Create;
            function  interfere (const   posicao : Integer): Integer;
            function  situacao : Integer;
            procedure usuario ( const nome : String);
          end;
        …
```

Figura 5.19. Visão parcial dos arquivos de implementação.

A segunda fase da implementação foi escrever os códigos de pedido dos clientes e as respectivas respostas dos servidores. Nos servidores, o código recebe os pedidos e os armazena, verificando se os mesmos não causavam nenhuma exceção. A implementação nas aplicações cliente resumem-se em interceptar a interação do usuário no ambiente virtual e transformá-la em valor (estado), enviando ao servidor.

5.5.1. Características Gerais

A idéia principal do projeto de distribuição deste trabalho é usar os servidores para armazenar as informações de seus clientes. Porém, cada abordagem (protótipo) desenvolvida possui características específicas. No **protótipo 1,** o servidor tem um papel invertido, em que o servidor da aplicação virtual de Biologia armazena todas as informações da aplicação virtual

de Química e vice-versa. Além disso, a própria aplicação tem propriedades de se tornar servidora, sempre que for necessário. O **protótipo 2** possui um servidor único responsável pelas informações de todos os clientes. Este servidor funciona independente das aplicações RV (ambientes virtuais). O **protótipo 3** possui um servidor para cada tipo de ambiente virtual. A principal diferença entre este protótipo e o primeiro está no fato dos servidores serem independentes das aplicações e não terem papéis invertidos (servidor Química atende a *clientes* Química e vice-versa). Uma outra característica deste protótipo é que o servidor se comporta como cliente de outro servidor, propiciando a comunicação entre os mesmos. O **protótipo 4** é similar ao protótipo 3, porém não há conexão entre os servidores e os clientes se comunicam com todos os servidores.

No código anterior (arquivo de implementação), onde estão descrições das interfaces, existem três principais métodos. O **método *interfere*** é usado pelo cliente para informar ao servidor que houve alguma interação do usuário. Esta interação é classificada de acordo com a posição do ambiente virtual (zero a três no ambiente de Biologia, de acordo com a temperatura ambiente; e, zero a dois no ambiente de Química, de acordo com o momento em que o processo da fotossíntese se encontra). Exemplificando, quando um usuário aumentar a temperatura no ambiente virtual de Biologia, o método *interfere* será invocado, provocando alteração nas propriedades ou estados de todos os ambientes virtuais de Química. O **método *situacao*** armazena e retorna o atual estado de cada ambiente virtual; portanto possibilita a um cliente, que iniciar na rede, atualizar seu estado, de acordo com a informação dos outros ambientes, armazenada no servidor. O **método *usuario*** permite a inserção, remoção e consulta dos usuários que estão ativos na rede. Todos estes métodos estão presentes nos quatro protótipos e funcionam similarmente.

5.6. Considerações Finais

Este capítulo apresentou a implementação de quatro protótipos com base na arquitetura proposta nos Capítulos 2 e 3 e mostrou as principais técnicas utilizadas na implementação do estudo de caso, que simula o Processo da Fotossíntese.

O estudo de caso apresentou dois ambientes virtuais, que se comunicam devido à aplicação do padrão para objetos distribuídos CORBA por meio da implementação Visibroker, que acompanha a versão 6.0 do Borland Delphi TM 6.0.

Os sistemas protótipos implementados mostraram diferentes abordagens de comunicação, nos quais, ambientes virtuais de Biologia se comunicam com ambientes virtuais de Química, sendo que a localização do computador, onde está um servidor ou cliente, é transparente. A comunicação se mostrou eficaz em todos os modelos, possibilitando uma comunicação bidirecional entre os ambientes com replicação para as cópias distribuídas em uma rede de computadores.

No próximo capítulo, apresenta-se o funcionamento do sistema.

CAPÍTULO VI

6. FUNCIONAMENTO DO SISTEMA

6.1. Introdução

Este capítulo apresenta o funcionamento do sistema protótipo que simula o Processo da Fotossíntese, considerando aspectos da Biologia e da Química.

Para isso, serão apresentados alguns conceitos sobre o Processo da Fotossíntese e, em seguida, o funcionamento do sistema protótipo.

6.2. Estudo de Caso - Processo da Fotossíntese

Quando os sacerdotes das religiões primitivas prestavam culto ao Sol, estavam certamente executando algo mais que um mero ato simbólico. De uma forma ou de outra, reconheciam um fato, mais tarde confirmado pela ciência moderna: toda a vida terrestre depende em última análise das radiações solares (PAULINO, 2001).

A Fotossíntese é o processo, pelo qual plantas verdes e alguns outros organismos transformam energia luminosa em energia química. Nas plantas verdes, a fotossíntese aproveita a energia da luz solar para converter dióxido de carbono, água e minerais em compostos orgânicos e oxigênio gasoso (LINHARES, 2000).

A fotossíntese só ocorre na natureza em presença de luz solar, por meio de células clorofiladas de uma folha. Na prática, é como se a planta estivesse liberando gás oxigênio e daí a sua ação ser “purificadora” do ar atmosférico.

A fotossíntese se realiza em duas etapas: a fase luminosa e a fase escura. Na fase luminosa ocorre a absorção da luz e a transformação da energia luminosa em energia de ATP (molécula de adenosina trifosfato). As moléculas de ATP ficam dissolvidas na célula e, gradativamente, vão sendo usadas no metabolismo espalhando-se pelas várias partes da célula (TREICHEL, 1998). Durante essa etapa, ocorre a quebra das moléculas de água em hidrogênio e oxigênio, os átomos de hidrogênio são capturados pelo NADP (nicotinamida adenina dinucleotídeo fosfato) e libera o oxigênio pela planta.

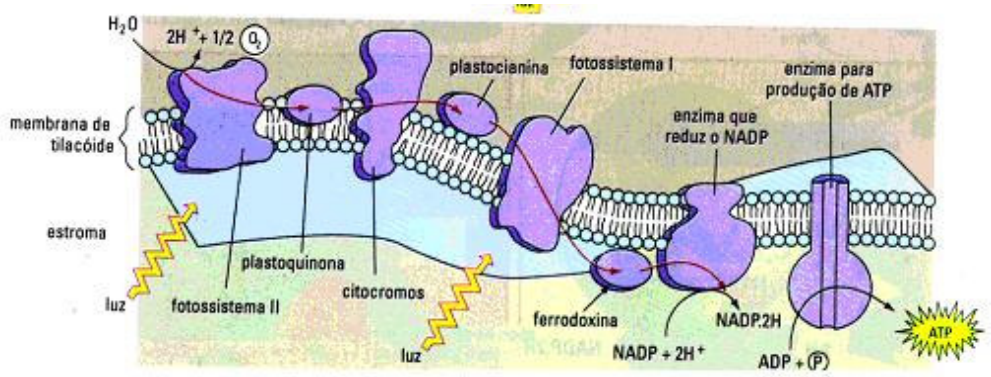


Figura 6.1. Esquema geral da fotofosforilação acíclica (LINHARES, 2000).

Nessa fase, também chamada de fotoquímica, a molécula de clorofila absorve energia luminosa. Uma vez absorvida, a energia é acumulada em elétrons que, por isso, escapam da molécula, sendo recolhidos por outras substâncias. Dependendo das substâncias transportadoras, os elétrons podem realizar dois tipos de trajeto: a fotofosforilação cíclica e a fotofosforilação acíclica. Em ambos os trajetos, os elétrons cedem energia que é utilizada para a síntese de ATP pela fosforilação. A fosforilação é um processo que adiciona ao ADP (adenina dinucleotídeo fosfato) um fosfato rico em energia. Como a energia do elétron vem originalmente da luz, dizemos que essas moléculas de ATP são produzidas por fotofosforilação, como mostra a Figura 6.1. A fase escura, também chamada de bioquímica, em que ATP e NADPH produzidos na fase clara são utilizados para a fixação de CO2 e ocorre

no estroma do cloroplasto. Esta denominação para as etapas da fotossíntese pode ser inadequada, porque mecanismos reguladores determinam que a chamada "fase escura" também seja dependente de luz.

6.3. Funcionamento do Sistema

O sistema apresenta dois modelos de ambientes virtuais (podem existir *n* cópias de cada um) com características e comportamentos independentes. Num modelo, tem-se o ambiente virtual da Biologia e num outro modelo o ambiente virtual da Química, como se pode observar na Figura 6.2. Um ambiente (Biologia) demonstra uma situação macro de um ambiente real. O outro ambiente (Química) demonstra uma situação micro de uma parte do outro ambiente. O que torna relevante esta comunicação é o fato de que cada ambiente está ligado a uma área distinta do conhecimento (Biologia e Química), permitindo a visualização real entre as mesmas (multidisciplinaridade). O ambiente virtual da Biologia apresenta alguns objetos, como o sol e as árvores. No ambiente virtual da Química, ocorre o processo da fotofosforilação acíclica, disparado durante o processo da fotossíntese, que simula o momento em que os elétrons são transportados, cedendo energia para a composição de ATP e produção de NADP.2H. Isso ocorre por meio de uma visão aumentada *n* vezes da membrana de uma das folhas das árvores.

No ambiente virtual da Biologia, de acordo com a luminosidade emitida pelos raios solares (fator importante no processo da fotossíntese), as folhas das árvores podem aumentar ou diminuir em tamanho. Esta luminosidade é controlada pela interação do usuário com um clique no sol. A Figura 6.3, mais adiante, ilustra a cena após o usuário ter interagido com o ambiente e alterado a temperatura, influenciando a quantidade de folhas nas árvores.

Ao mesmo tempo, um evento é enviado ao ambiente virtual da Química e os elétrons são liberados, passando por uma cadeia transportadora e liberando energia utilizada na

produção de ATP e NADP.2H.

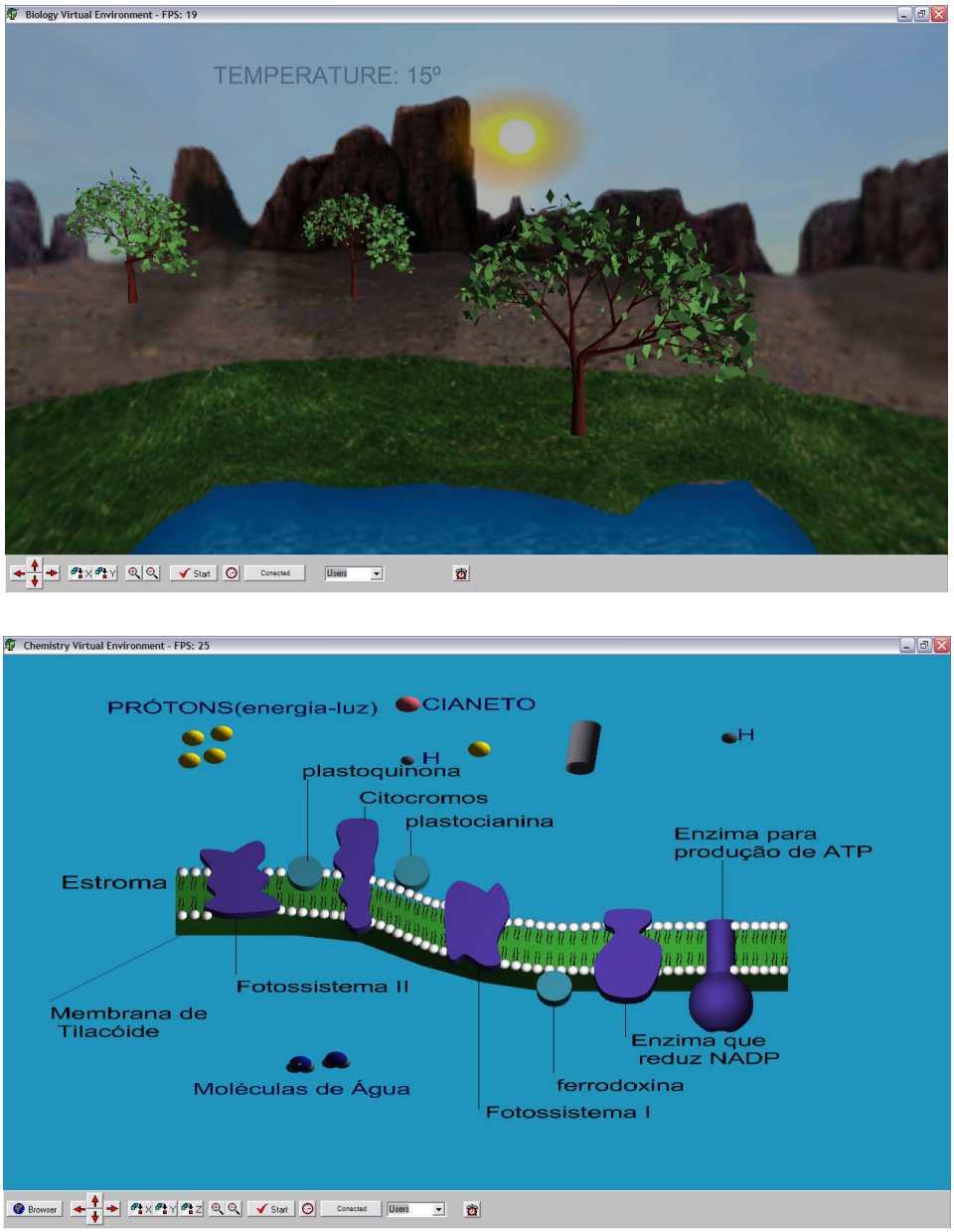
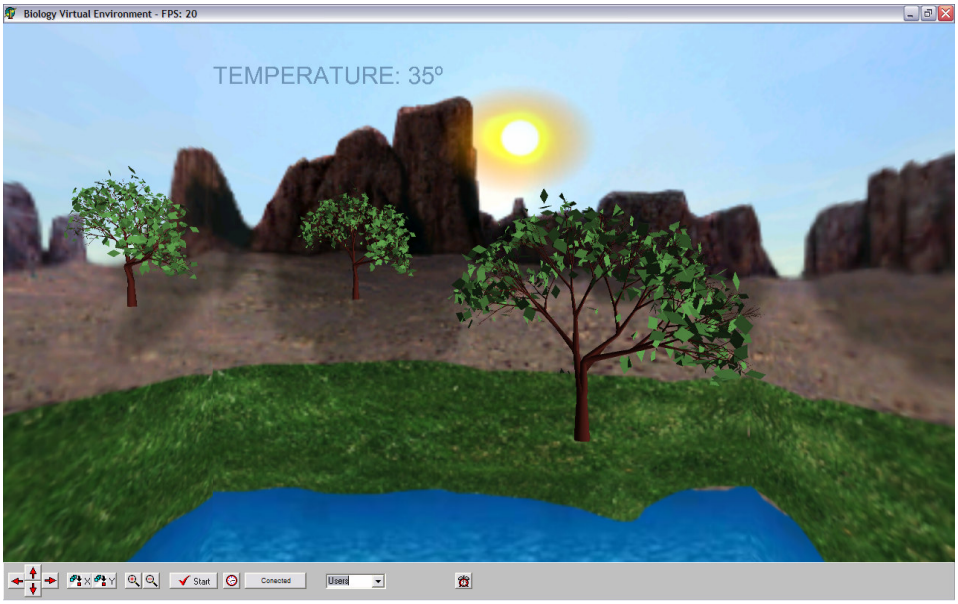


Figura 6.2. Estágio inicial dos ambientes virtuais.



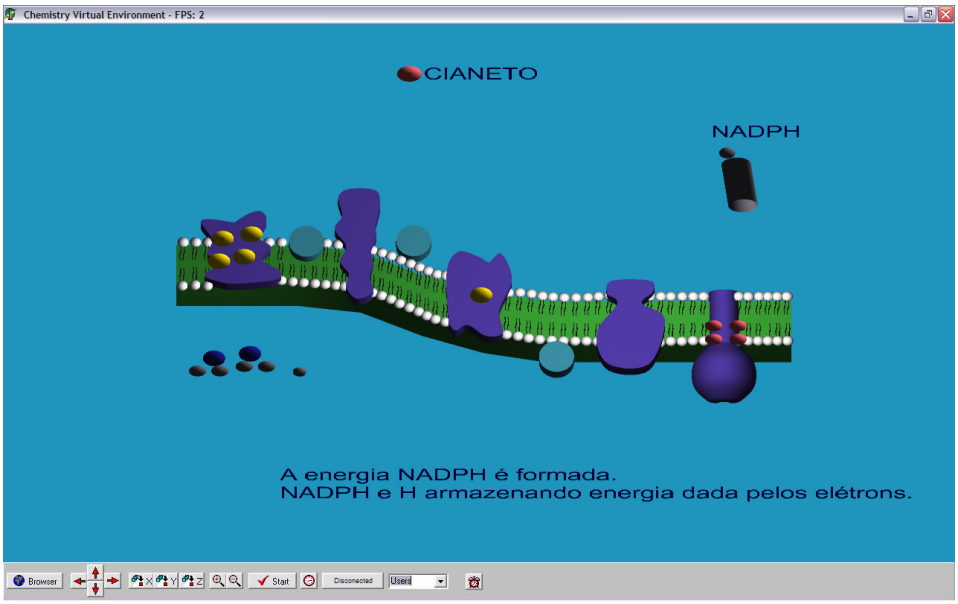


Figura 6.3. Ambientes virtuais após a interação do usuário no mundo virtual da Biologia.

O ambiente virtual da Química também interfere no ambiente virtual da Biologia. Para isto, basta clicar no objeto que representa uma molécula de Cianeto. O Cianeto no Estroma representa a morte das plantas ou simplesmente a perda das folhas. A Figura 6.4 ilustra a cena após o usuário ter interagido com o ambiente e inserido a molécula de Cianeto na cadeia do processo de fotofosforilação acíclica. Como conseqüência desta interação, no

ambiente virtual da Biologia as árvores perdem as folhas e no ambiente virtual da Química o Cianeto presente no Estroma.

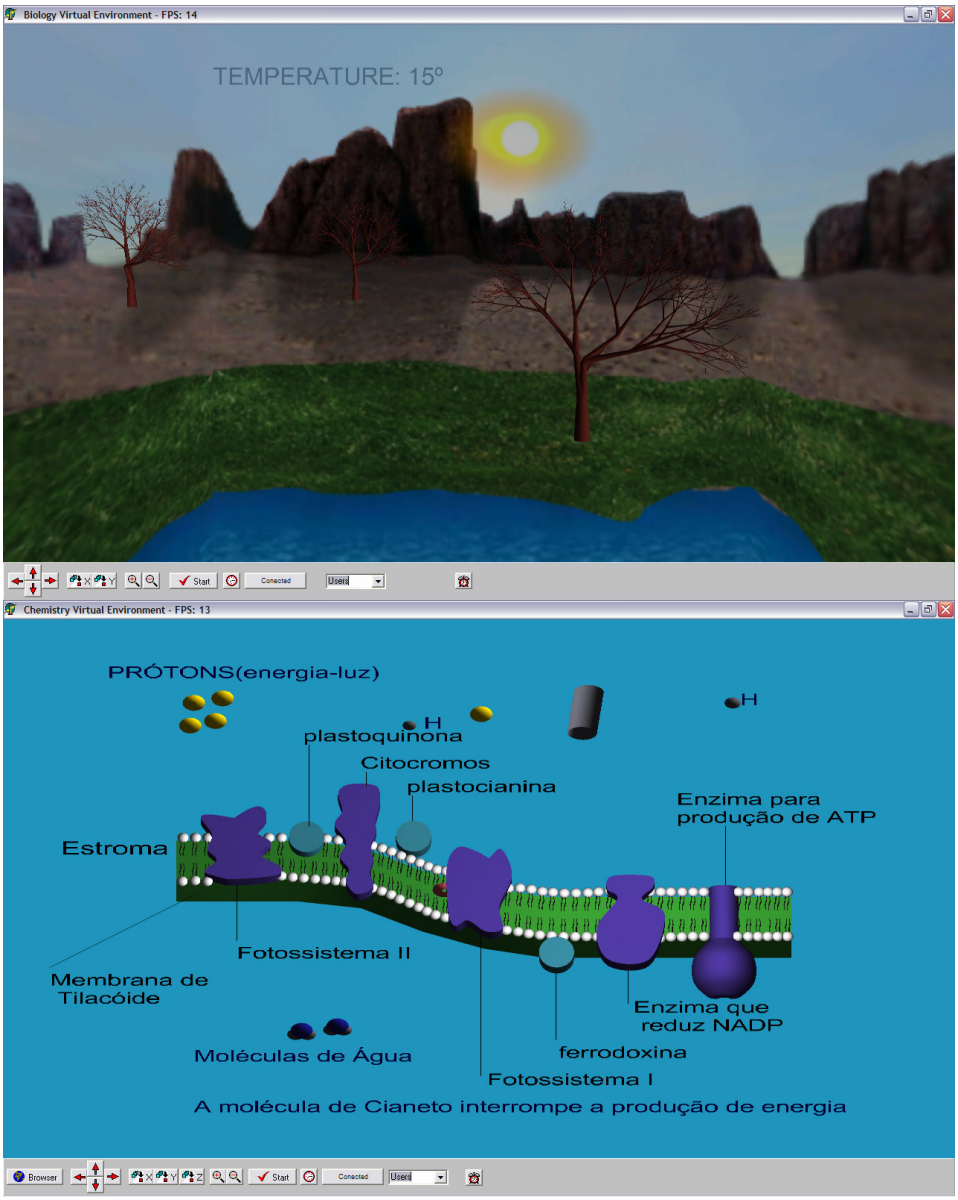


Figura 6.4. Ambiente Virtual de Biologia e Ambiente Virtual de Química no estágio final.

6.4. Distribuição da informação

Qualquer que seja a atividade ou fenômeno simulado nos ambientes virtuais, os mesmos acontecem apenas seqüencialmente. Ou seja, não é possível, por exemplo, iniciar o

processo da fotofosforilação acíclica e tentar colocar a molécula de cianeto na membrana. Portanto, para cada tipo de interação em cada ambiente é associado um estado que possui valor que varia de zero a três de acordo com cada ambiente. Sempre que um ambiente é iniciado na rede, o mesmo procura por um servidor. Caso não o encontre, o ambiente começa a interagir com o usuário a partir do estado zero (Biologia: temperatura 15 graus, árvores com folhas pequenas e luminosidade baixa; Química: cianeto fora da cadeia, processo da fotofosforilação acíclica paralisado). A responsabilidade do armazenamento destes estados é do primeiro par de ambientes virtuais que for iniciado na rede.

A arquitetura proposta para este trabalho suporta a comunicação de *n* ambientes que se comunicam ou interagem com *m* ambientes diferentes. Diante disso, após a ativação do primeiro par de ambientes sempre que houver outra ativação – não necessariamente em pares – a aplicação irá iniciar com estado de visualização de acordo com o valor armazenado no servidor. Sempre que houver alguma interação, o valor do estado do ambiente será modificado e enviado ao servidor, que replica esta informação para todas as cópias.

Além do estado inicial, valor zero, os ambientes possuem as possíveis configurações:

Biologia:

Estado 1 - temperatura 25 graus, árvores com folhas médias e luminosidade média.

Estado 2 – temperatura 35 graus, árvores com folhas grandes e luminosidade alta.

Estado 3 – árvores sem folhas.

Química:

Estado 1 – processo da fotofosforilação acíclica em andamento.

Estado 2 - cianeto dentro da cadeia.

6.5. Considerações Finais

Este capítulo apresentou o sistema, detalhando o funcionamento dos ambientes

virtuais no contexto do estudo de caso do Processo da Fotossíntese, e explicitou como e qual informação é distribuída.

Cabe ressaltar que os detalhes apresentados neste capítulo foram expostos por meio de figuras, que mostram apenas uma instância de cada ambiente, porém a arquitetura suporta vários ambientes de cada área (Biologia e Química).

No próximo capítulo, apresenta-se a análise dos resultados obtidos.

CAPÍTULO VII

7. COMPARAÇÃO DOS PROTÓTIPOS E ANÁLISE DOS RESULTADOS

7.1 Introdução

Neste capítulo, é apresentada uma análise dos protótipos descritos no Capítulo 5. Esta análise tem como objetivo demonstrar a viabilidade das abordagens propostas nesta tese e dar uma idéia do desempenho do protótipo.

A análise realizada abrange os seguintes aspectos:

a) A obtenção da latência de comunicação exibida pelo protótipo, por meio da mensuração dos tempos de comunicação dos sistemas.

b) A identificação das tendências, em relação à escalabilidade (aumento do número de clientes) dos protótipos.

c) Uma estimativa do desempenho exibido pelas aplicações, em relação à extensibilidade (aumento do número de modelos de ambientes, Biologia, Física, Química, etc.).

Mesmo tendo três aspectos comparativos, a escalabilidade é a principal preocupação quando se desenvolve um AVD, pois, a capacidade do sistema de aceitar novos clientes sem perder desempenho ou afetar sua funcionalidade é requisito fundamental.

7.2 Ambiente Experimental

Como ambiente para os experimentos, foi usada uma rede local Ethernet dedicada

operando comutada com velocidade de 10 Mbits, composta de estações implementadas com microcomputadores compatíveis com IBM PC (*Pentium IV 3.2 MHz; memória RAM 512 Mbytes e Disco Rígido de 40 Gbytes).*

De forma a reduzir o número de variáveis independentes potencialmente não gerenciáveis, seguiu-se o seguinte procedimento para os experimentos:

a) Todos os computadores utilizados possuíam idêntica configuração de *hardware* e *software* (sistema operacional).
b) Todas as aplicações clientes possuíam cópias idênticas do ambiente virtual.

7.3. Análise dos Resultados Obtidos

No capítulo anterior, foi descrito que toda interação acontece no cliente (Biologia ou Química) e que o resultado desta interação somente é apresentado, quando o cliente faz uma requisição ao servidor e recebe de volta uma aprovação do pedido. Portanto, neste sistema (em todos os protótipos), a obtenção da latência é iniciada pelo envio, a partir de uma aplicação cliente, de uma mensagem com tamanho conhecido (quatro bytes – em nível de aplicação). O maior tempo gasto para que esta mensagem chegue até o último cliente foi considerada. O tempo foi medido em milissegundos. De acordo com Wang (1999) existem vários fatores que influenciam ou alteram resultados de avaliações em redes *Ethernet,* dependendo do momento e das configurações. Para tanto, esse procedimento foi repetido dez vezes para cada número de clientes (nós) na rede, e o resultado final foi baseado no maior valor, considerando que quando houve uma grande diferença entre o menor valor e o maior valor, os teste foram refeitos.

7.3.1. Latência da Comunicação

Do ponto de vista do ambiente virtual distribuído, é a latência que controla a natureza

interativa e dinâmica do sistema. Se o ambiente distribuído existe para emular o mundo real, deve operar em termos da percepção humana (MACEDONIA, 2005). Segundo Wloka (1995), sistemas que envolvem operadores humanos devem entregar pacotes com a mínima latência e gerar gráficos texturizados 3D a uma freqüência de 30 a 60 Hz, para garantir a ilusão de realidade. Este desafio torna-se maior em sistemas que utilizam redes a longa distância, conforme mencionado anteriormente. A latência da rede pode ser reduzida, até certo ponto, por meio da utilização de enlaces (*links*) dedicados. Todavia, uma maior largura de banda não é necessariamente a melhor solução.

A latência na comunicação não será o parâmetro principal de análise de desempenho do modelo proposto neste estudo. A latência será o suporte para analisar a escalabilidade (capacidade de aumento na quantidade de usuários) e a extensibilidade (capacidade de inserção de outros ambientes virtuais). Para a realização desta análise, as mensagens de solicitação de tempo foram enviadas para os nós (computadores) que possuem pelo menos um par de clientes (Biologia e Química). O pacote (quatro bytes) foi enviado inicialmente para um total de cinco computadores cada um, com dois clientes (Biologia e Química). O experimento foi repetido, seguidas vezes, em cada um dos nós. O valor final foi calculado pela média para cada pedido de envio. A Figura 7.1 ilustra o resultado obtido.

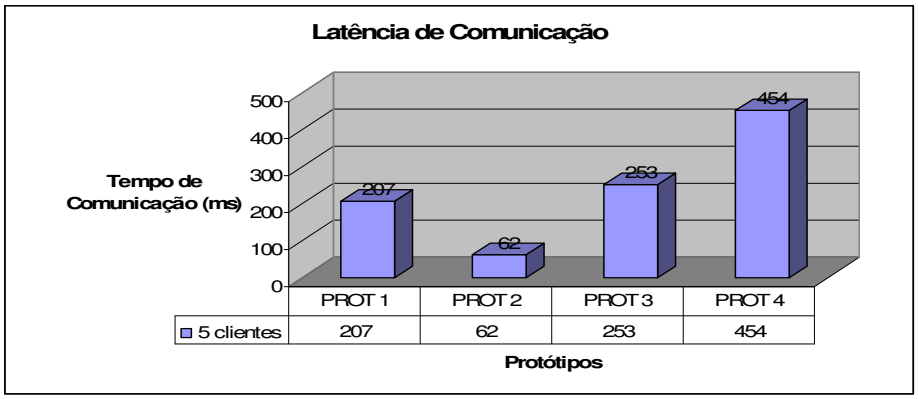


Figura 7.1. Latência para 100 mensagens.

Como pode ser observado, a menor latência foi obtida pelo protótipo 2, em segundo

lugar o protótipo 1, em terceiro lugar o protótipo 3 e por último o protótipo 4. Estes resultados já eram esperados, uma vez que:

- O Protótipo 1, por utilizar uma comunicação direta entre clientes, apesar de sofrer atrasos causado pelo uso de servidores teve o segundo melhor desempenho.

- O Protótipo 2, apesar da centralização causada pelo servidor, portou-se melhor com essa quantidade de clientes. Isto se deve ao fato da troca de mensagens ser pequena e a quantidade de requisições não sobrecarregar o servidor.

- O Protótipo 3 apresentou o terceiro melhor desempenho. Este protótipo possui características bem próximas do protótipo 1. Porém, o fato de existir um servidor dedicado relacionado a cada ambiente e cada servidor se conectar ao outro para propiciar a comunicação entre os ambientes, causou uma latência de comunicação maior que o primeiro protótipo.

O Protótipo 4 apresentou, como era esperado, a pior performance. Isto se deve ao fato de existir dois servidores dedicados; a comunicação ocorre mediante o uso de servidores, e o cliente necessita estar conectado a cada servidor existente (um servidor para cada tipo de ambientes).

7.3.2. Escalabilidade

A escalabilidade foi interpretada como sendo a capacidade do sistema em aceitar novos clientes, numa mesma máquina ou em máquinas diferentes. A escalabilidade foi medida em função do resultado da análise anterior, levando-se em conta a degradação do tempo de comunicação com o aumento do número de clientes. Os testes iniciais foram feitos com base em cinco clientes. A escalabilidade foi testada, aumentando esta quantidade para 10, 15 e 20 clientes, respectivamente. Os resultados obtidos estão nas Figuras 7.2, 7.3, 7.4 e 7.5.

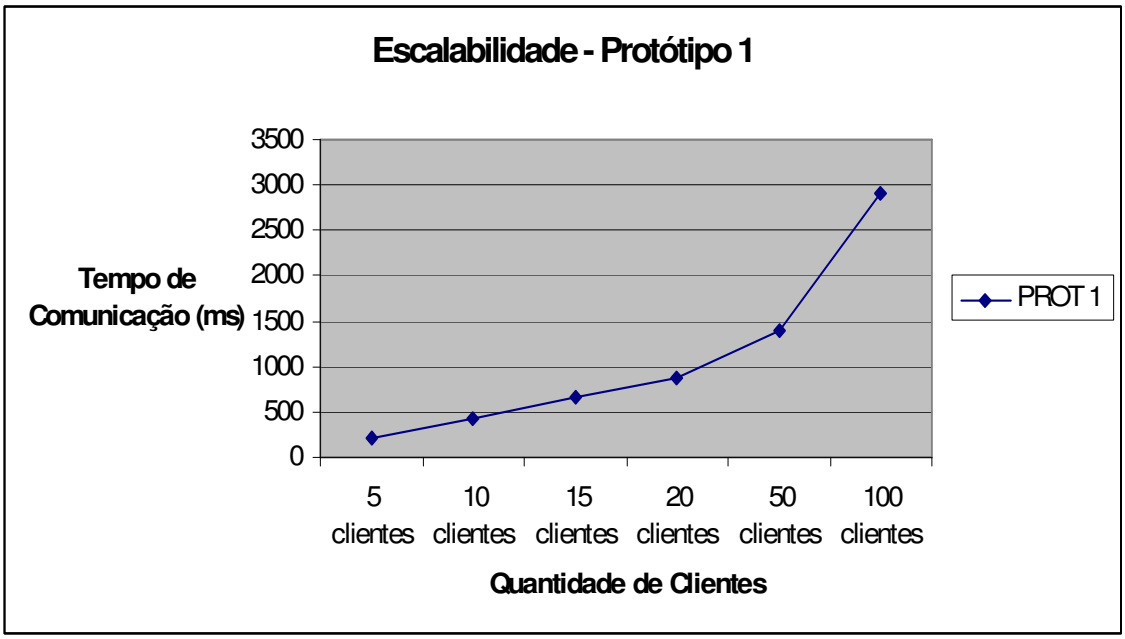


Figura 7.2. Estimativa da Escalabilidade do Protótipo 1.

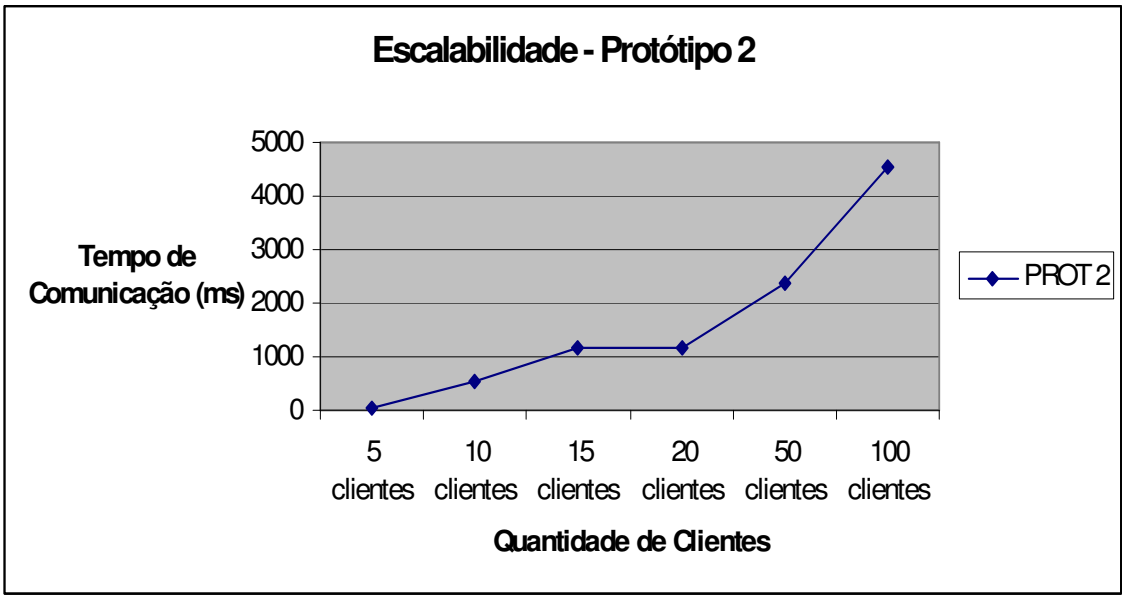


Figura 7.3. Estimativa da Escalabilidade do Protótipo 2.

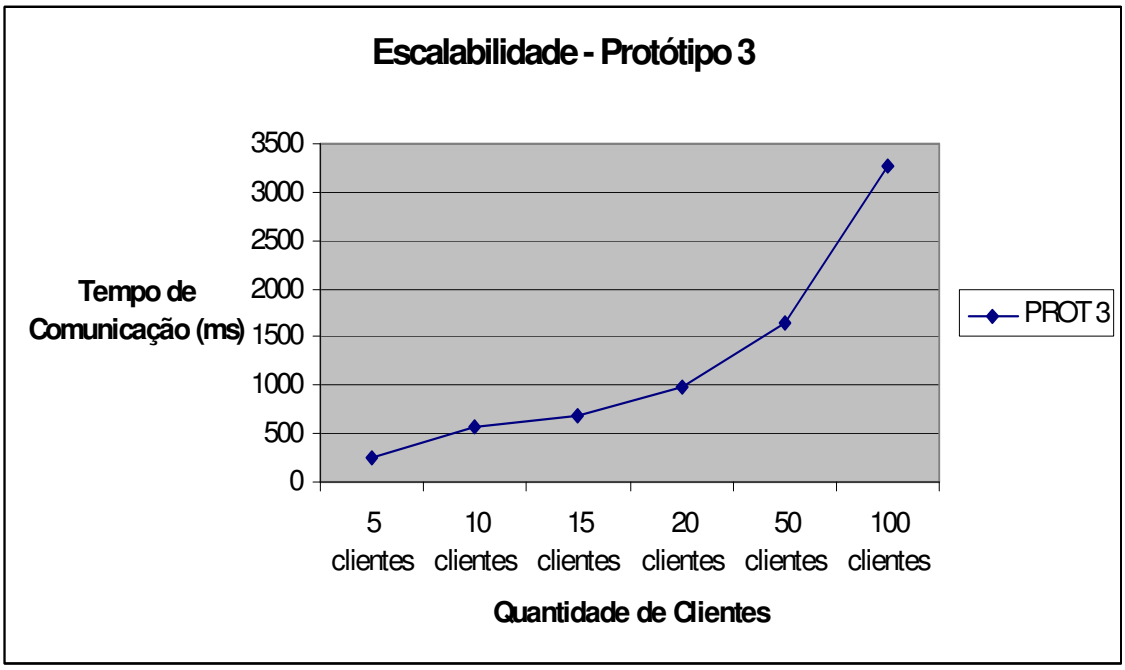


Figura 7.4. Estimativa da Escalabilidade do Protótipo 3.

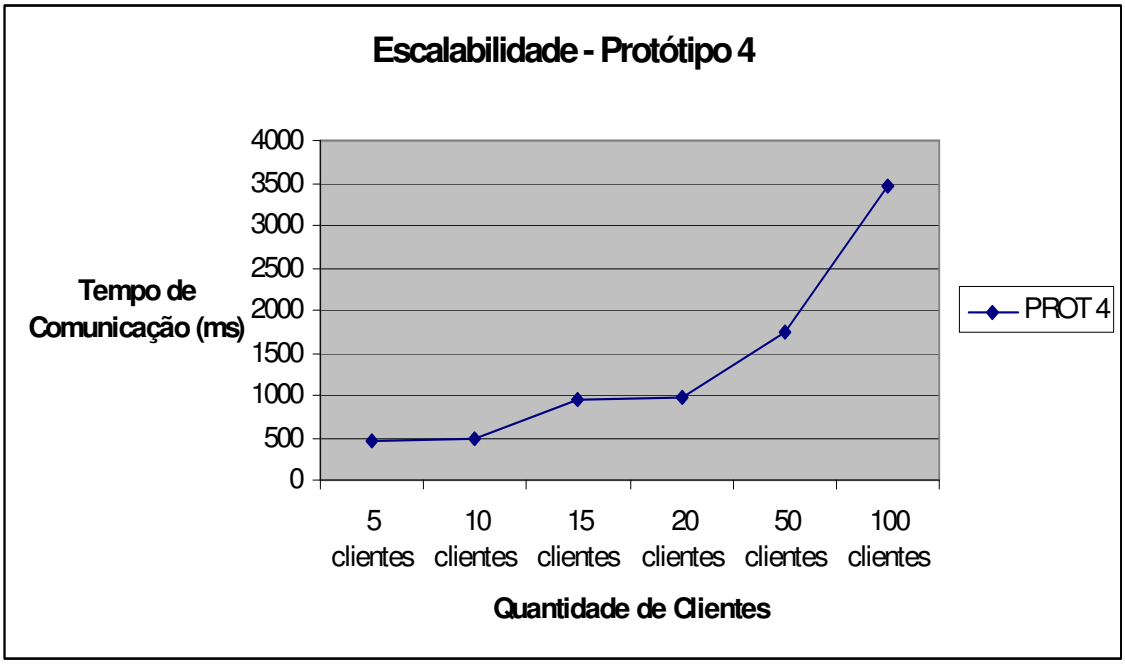


Figura 7.5. Estimativa da Escalabilidade do Protótipo 4.

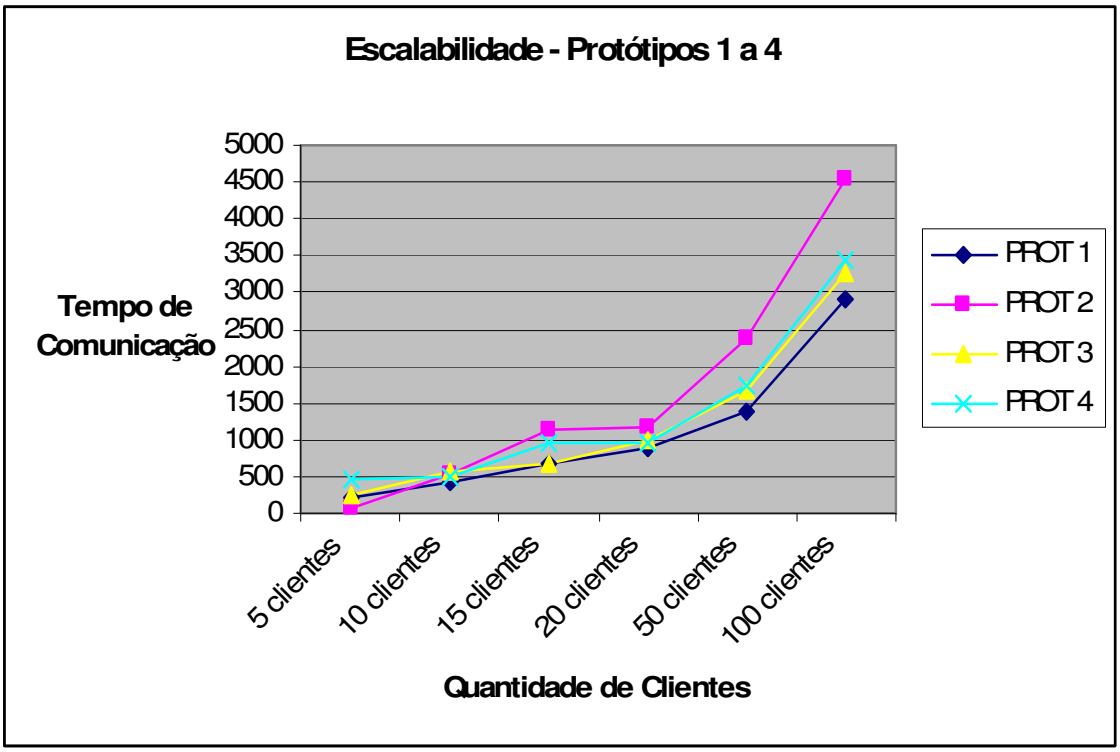


Figura 7.6. Comparação da Escalabilidade entre os Protótipos.

Nota-se, pelas Figuras 7.2, 7.3, 7.4, 7.5 e 7.6, que o protótipo 1 apresentou as menores degradações do tempo de comunicação. O protótipo 2, que apresentou os melhores resultados com apenas cinco clientes, teve um aumento considerável na latência de comunicação causada pelo "gargalo" que a centralização no servidor causa na rede. Os protótipos 3 e 4 tiveram os piores resultados. O fato de trabalhar com servidores dedicados, facilita a construção da aplicação, porém causa um desempenho desfavorável, quando comparado a um modelo, onde o papel de servidor é exercido por uma aplicação qualquer e este papel pode ser alterado, durante o processo de comunicação, sem falhas ou transtornos. Fica claro também na análise que a partir de um aumento considerável de clientes (20 ou mais) que a curva de tempo de comunicação se acentua, demonstrando um mesmo padrão para todos os protótipos. Para possibilitar uma dinamicidade, o algoritmo de tolerância a falhas (FAVD), criado para estes protótipos, sugerindo o protótipo 1 como a melhor opção ou

a melhor abordagem para AVDs.

7.3.3. Extensibilidade

A extensibilidade foi interpretada como sendo a capacidade de inserir novos modelos de ambientes virtuais (áreas diferentes) numa mesma máquina ou em máquinas diferentes. A extensibilidade foi medida, comparando o tempo total gasto no envio de mensagens com apenas um modelo de ambiente virtual (Biologia) e suas réplicas, dois (Biologia e Química) e três (Biologia, Química e outro) ambientes virtuais, mantendo a mesma abordagem cada protótipo. Para o terceiro e o quarto ambientes virtuais, foram criados modelos híbridos que possuem a mesma estrutura de funcionamento que os demais, mesclando algumas das funcionalidades de cada ambiente. O teste foi realizado, tendo sempre a mesma quantidade de clientes (cinco) e com a mesma quantidade de testes (10). As Figuras 7.7 e 7.8 ilustram os resultados obtidos.

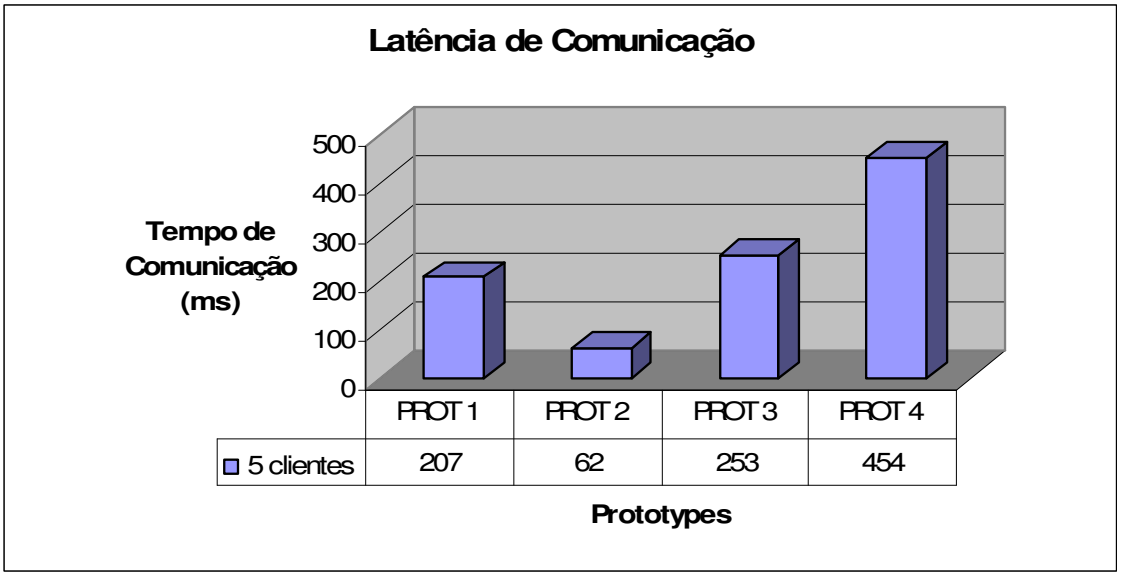
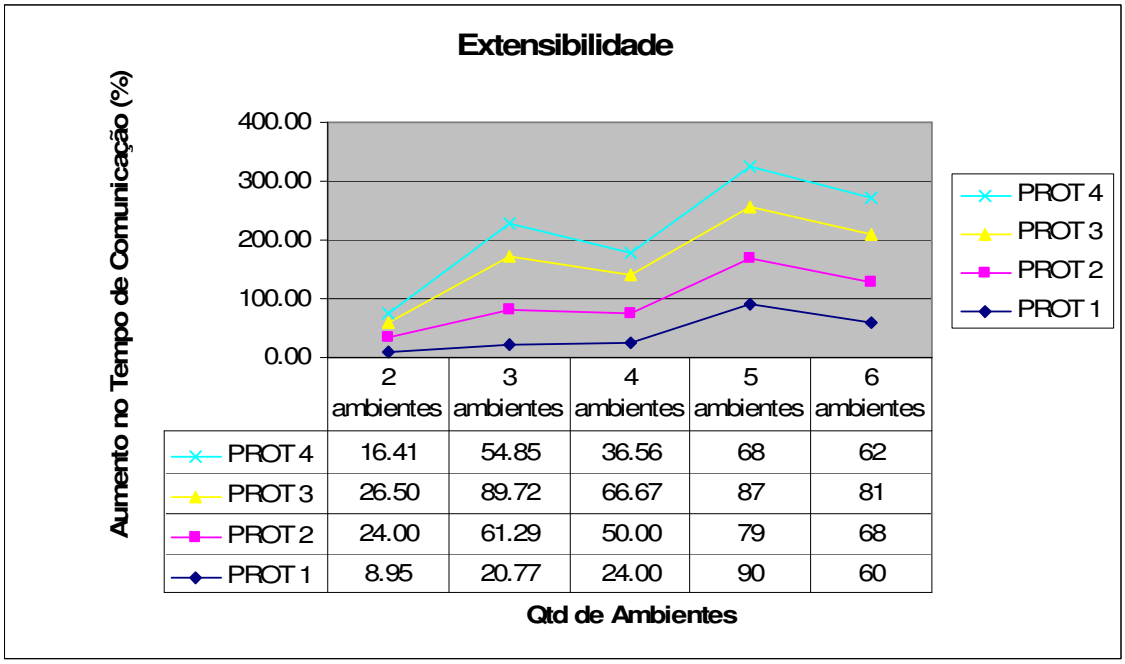


Figura 7.7. Latência de Comunicação com cinco clientes e apenas um ambiente.



	2 ambientes	3 ambientes	4 ambientes	5 ambientes	6 ambientes
PROT 4	16.41	54.85	36.56	68	62
PROT 3	26.50	89.72	66.67	87	81
PROT 2	24.00	61.29	50.00	79	68
PROT 1	8.95	20.77	24.00	90	60

Figura 7.8. Estimativa da Extensibilidade dos Protótipos com cinco clientes.

Assim, os testes demonstram que a inserção de outros ambientes não afetam de forma prejudicial o desempenho do sistema. Os resultados foram obtidos, verificando o tempo de comunicação (ms) resultante do aumento da quantidade de clientes. O protótipo 1 continua tendo os melhores resultados, juntamente com o protótipo 4. Cabe ressaltar que, em termos de implementação, o protótipo 1 causaria grandes dificuldades, pois é necessário mudar vários aspectos de sua estrutura para permitir uma comunicação diferente da proposta com um par de ambientes (Biologia e Química). Também é possível observar na Figura 7.8 que existe uma tendência de estabilização no desempenho no sistema quando o número de ambientes é par. Isso se deve ao fato da arquitetura dos protótipos ser construída para comunicação de ambientes em pares.

7.3.4. Análise da Geração de Imagens

Segundo Kirner (2005), grande parte da ênfase do projeto de sistema de realidade

virtual tem sido estabelecida pelas restrições de geração da cena visual.

Na geração da cena visual, o sistema requer taxas altas de quadros (*frames*) por segundo. O conceito de quadros é proveniente da animação baseada em uma sucessão rápida de uma seqüência de fotografias. Para manter a ilusão de movimento, a taxa ideal da troca de fotos é 20 quadros por segundos. Do ponto de vista gráfico, que está ligado à sensação de presença ou imersão, o mínimo aceitável está na ordem de oito a dez quadros por segundo.

No modelo proposto, a taxa de quadros foi testada no mesmo conjunto de equipamentos em que foram submetidos os testes de escalabilidade e extensibilidade. A menor taxa levantada nos teste foi de 22 quadros por segundo e a maior foi de 66 quadros por segundo (KIRNER, 2005).

7.3.5. Comparação com outros AVDs

De acordo com Tabela 7.1, a principal diferença entre os protótipos desenvolvidos neste trabalho com os AVDs avaliados refere-se ao fato da arquitetura dos protótipos possibilitar o funcionamento de sistemas multidisciplinares.

Tabela 7.1. Comparação com outros AVDs

AVDs	CARACTERÍSTICAS						
	Mecanismo de Transporte	**Estrutura de Comunicação**	**Gerenciamento de Dados**	**Gerenciamento da Computação**	**Suporte a Múltiplos Mundos**	**Suporte a Múltiplos Usuários**	**Suporte a AVs Distintos**
BrickNet	Cliente/Servidor	Cliente/Servidor	BD Distribuída com replicação parcial	Distribuição Total	Sim	Sim	Não
PEPITO	Ponto a Ponto	Cliente/Servidor	BD Distribuída com replicação parcial	Distribuição Parcial	Sim	Sim	Não
CurlSpace	Cliente/Servidor	Cliente/Servidor	BD Distribuída com replicação parcial	Distribuição Parcial	Sim	Sim	Não
DKS	Ponto a Ponto, *Multicast, Broadcast*	Cliente/Servidor	BD Distribuída com replicação parcial	Distribuição Total	Sim	Sim	Não
NPSNET	Ponto a Ponto, *Multicast*	*Peer*	BD Distribuída com replicação total	Distribuição Parcial	Não	Sim	Não
SIMNET	Ponto a Ponto	*Peer*	BD Distribuída com replicação total	Distribuição Parcial	Não	Sim	Não
DVE	Ponto a Ponto, Cliente/Servidor	Cliente/Servidor	BD Centralizado	Replicação Total	Sim	Sim	Não
EVI3D	Ponto a Ponto, Cliente/Servidor	Cliente/Servidor	BD Compartilhado	Replicação Parcial	Sim	Sim	Não
MASSIVE	Ponto a Ponto	*Peer*	Não mencionada	Distribuição Total	Sim	Sim	Não
TeamSpace	*Multicast*, Cliente/Servidor	Cliente/Servidor	BD Distribuída com replicação total	Distribuição Total	Sim	Sim	Não
SPLINE	*Multicast*	*Peer*	BD Distribuída com replicação total	Distribuição Total	Não	Sim	Não
VERN	*Broadcast*	*Peer*	BD Distribuída com replicação total	Distribuição Total	Não	Sim	Não
AVVIC	*Multicast* e Ponto a ponto	Cliente/Servidor	BD Distribuída com replicação total	Distribuição Total	Sim	Sim	Não
ESPELHO	*Multicast*	Cliente/Servidor	BD Distribuída	Distribuição Parcial	Sim	Sim	Sim
DEDICADO	Ponto a ponto	Cliente/Servidor	BD Centralizada	Distribuição Parcial	Sim	Sim	Sim
ENCADEADO	*Multicast*	Cliente/Servidor	BD Distribuída	Distribuição Parcial	Sim	Sim	Sim
DIVIDIDO	Ponto a ponto	Cliente/Servidor	BD Distribuída/Compartilhada	Distribuição Parcial	Sim	Sim	Sim

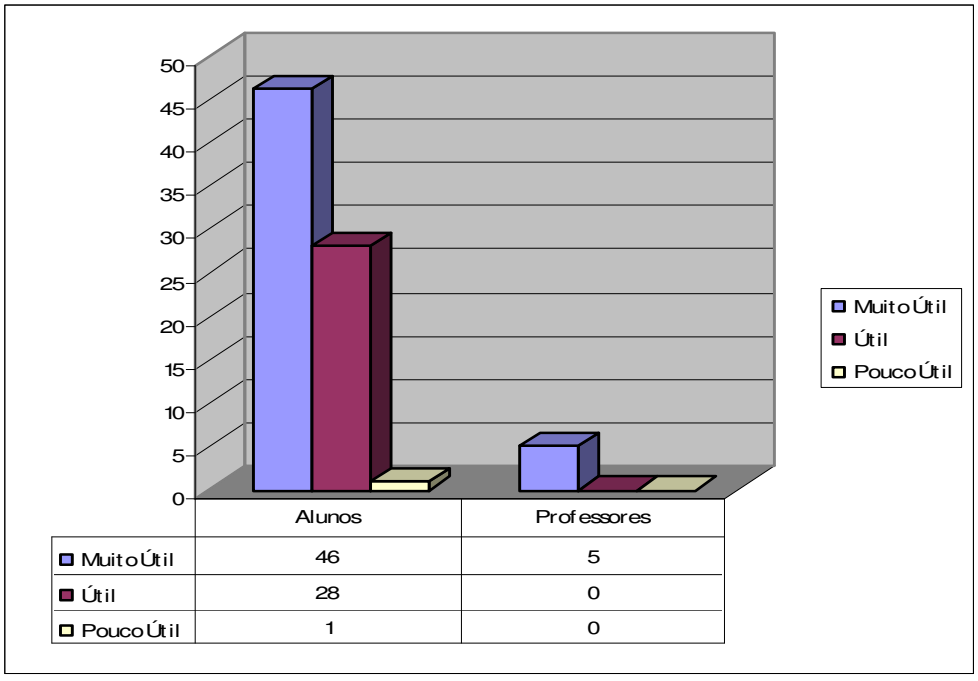
7.4. Avaliação do Sistema

O sistema foi apresentado a 80 (oitenta) usuários, sendo cinco professores (um da área de tecnologia, um da área de Pedagogia e três da área de Biologia), 35 alunos do ensino médio e 40 alunos do ensino superior.

Primeiramente, foi explicado a esses usuários o objetivo do sistema e, em seguida, grupos de 15 alunos testaram coletivamente o sistema. Após a execução do sistema, os usuários responderam a um questionário.

Analisando as respostas nos questionários, foi possível avaliar os itens que seguem abaixo e para cada item foi gerado um gráfico comparativo.

- ✓ Quanto à finalidade do uso da ferramenta:



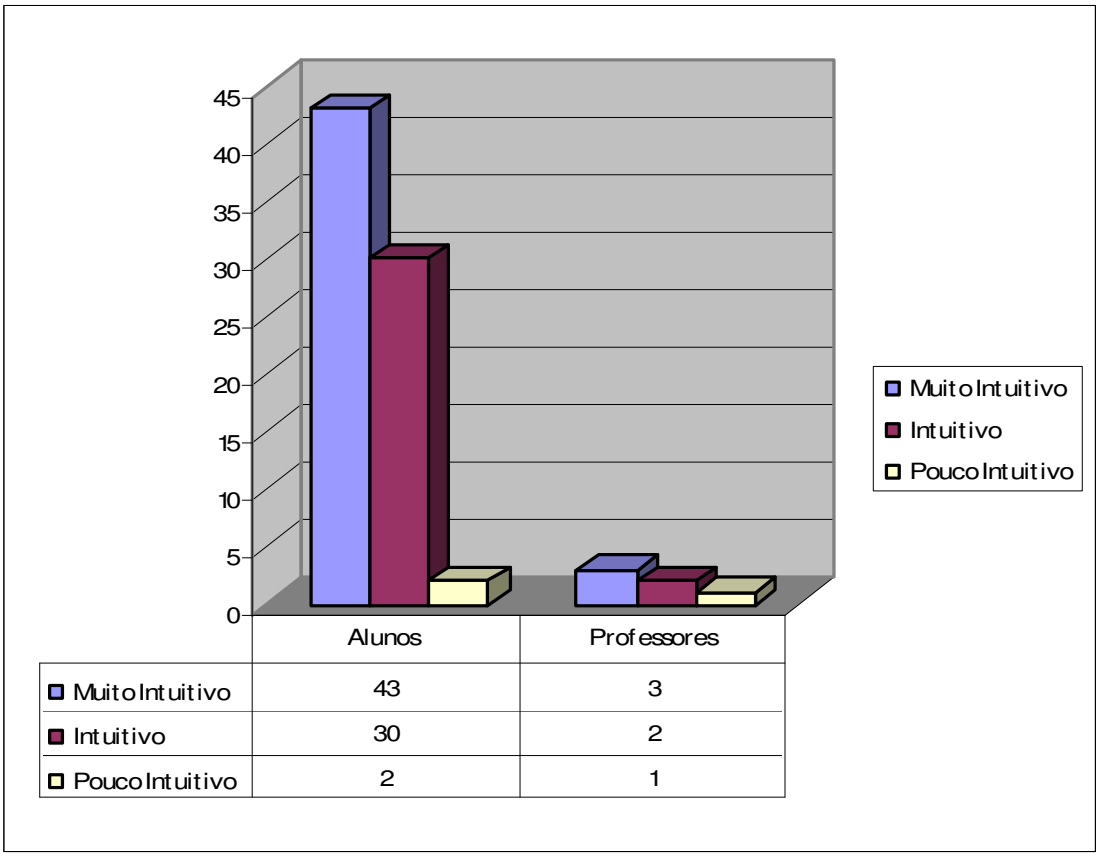
	Alunos	Professores
Muito Útil	46	5
Útil	28	0
Pouco Útil	1	0

Figura 7.9. Análise quanto a finalidade.

No gráfico, observa-se que a grande maioria dos usuários respondeu que o sistema é **muito útil**. Algumas pessoas que responderam que o sistema é **útil** justificaram as suas

respostas, considerando que o sistema é adequado apenas como complemento do conteúdo abordado em sala de aula, pois necessita de um prévio conhecimento sobre o assunto abordado.

- ✓ Quanto à interface com o usuário:



	Alunos	Professores
Muito Intuitivo	43	3
Intuitivo	30	2
Pouco Intuitivo	2	1

Figura 7.10. Análise quanto a Interface.

A maior parte das pessoas que avaliou o sistema como **muito intuitivo e intuitivo** e apenas três usuários o consideraram **pouco intuitivo** e não justificaram as suas respostas, a maioria das pessoas elogiou sistema mesmo o considerando **intuitivo**, e outras citaram a necessidade de um prévio conhecimento de Informática para a execução do mesmo.

✓ Quanto à facilidade de uso:

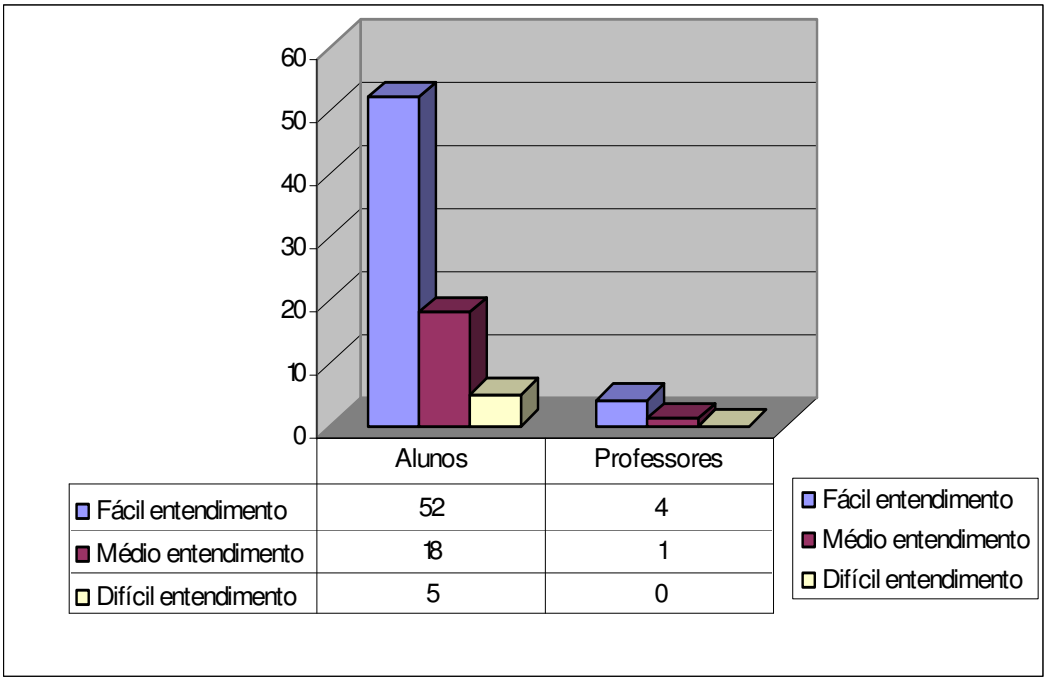
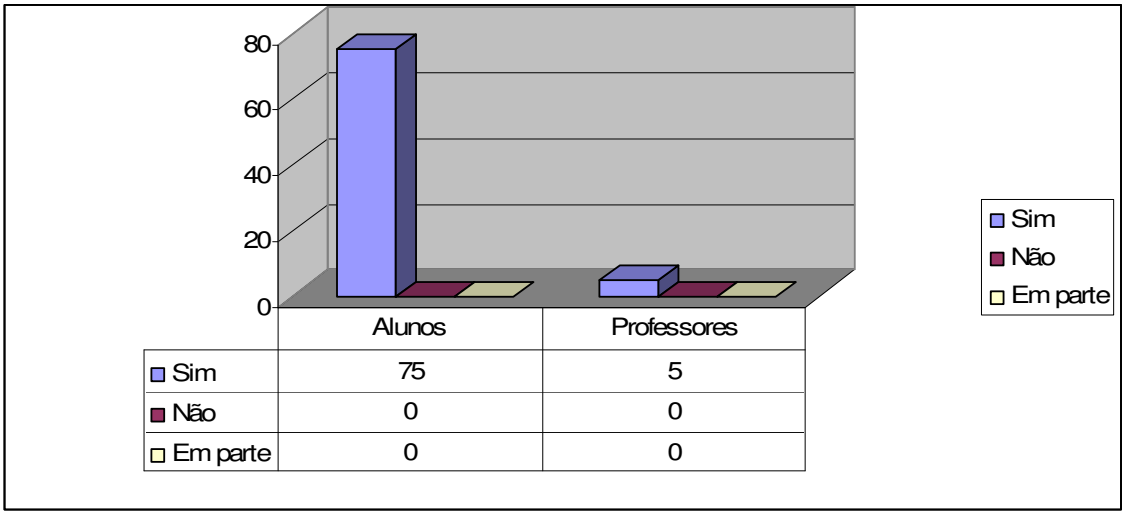


Figura 7.11. Análise quanto a facilidade de uso.

Observando o gráfico, a maioria dos usuários considerou os comandos apresentados de fácil entendimento, os demais justificaram a necessidade de um prévio conhecimento de Informática, considerando que muitos usuários poderão ter dificuldades em executar os comandos, exigindo antes da execução a apresentação de algumas informações adicionais para operacionalização do sistema.

✓ Conseguiu visualizar a relação da disciplina de Biologia com a disciplina de Química, por meio do estudo de caso apresentado?

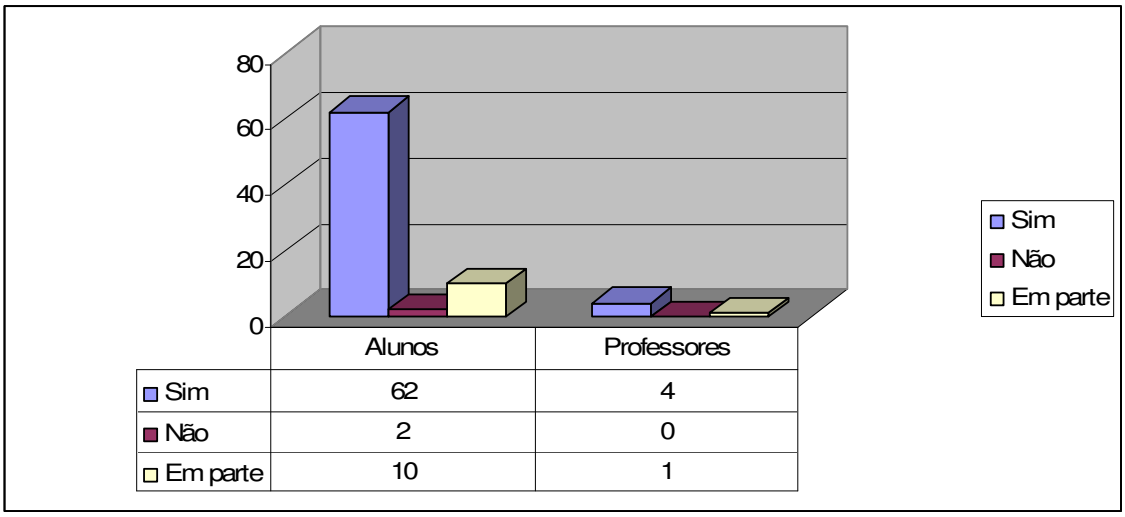


	Alunos	Professores
Sim	75	5
Não	0	0
Em parte	0	0

Figura 7.12. Análise quanto a multidisciplinaridade.

De acordo com o gráfico acima, fica claro que todos os usuários conseguiram visualizar a relação da disciplina de Biologia com a disciplina de Química.

✓ O programa permitiu a aquisição de informações úteis a respeito de como funciona o Processo da Fotossíntese?



	Alunos	Professores
Sim	62	4
Não	2	0
Em parte	10	1

Figura 7.13. Análise quanto aos recursos do programa.

Pelo gráfico, pode-se observar que a maioria dos usuários, que respondeu ao questionário, considerou que o programa permitiu a aquisição de informações úteis a respeito

de como funciona o Processo da Fotossíntese. Entre várias justificativas, a existência de usuários que consideraram o ambiente da Química uma demonstração prática do que ocorre no processo da fotossíntese na formação de energia.

Em uma outra avaliação do sistema, de acordo com as normas ISO/IEC 9126 (ISO, 1991), adaptado para avaliar *software* educacional, foram obtidos os seguintes resultados como mostra a Figura 7.14.

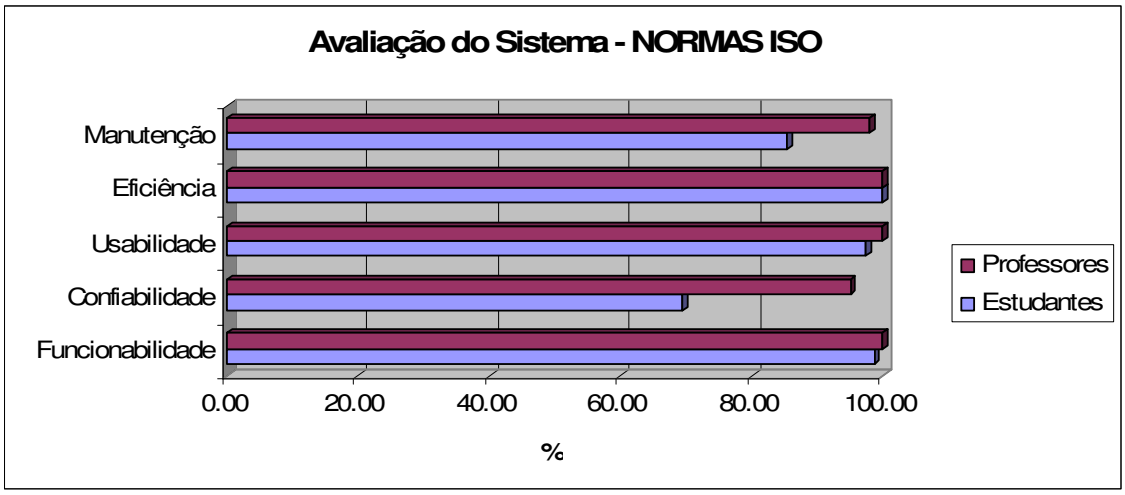


Figura 7.14. Análise dos aspectos de avaliação para *softwares* educacionais

Analisando todos os itens avaliados nos questionários, conclui-se que o sistema protótipo desenvolvido foi bem aceito pelos usuários entrevistados. Estes contribuíram com algumas sugestões, descritas a seguir:

- ✓ Inserir no mundo virtual da Química, códigos ou legendas explicativas sobre a composição química.
- ✓ Inserir outros objetos que fazem parte do meio ambiente.
- ✓ Mostrar o processo da fotossíntese à noite (fase escura).
- ✓ Inserir novas relações entre os objetos, como o caso da poluição no processo da fotossíntese.
- ✓ Inserir recursos sonoros.

- ✓ Mostrar a relação das ações dos raios UVA e UVB.
- ✓ Especificar o tipo do clima (outono, inverno, primavera e verão).
- ✓ Todos os professores e alguns alunos sugeriram a criação de um mecanismo de controle, administrado por um professor, para controlar as requisições entre os clientes.
- ✓ Criar um meio de comunicação textual entre os participantes para que o aluno possa enviar ao grupo mensagens.

Analisando a avaliação feita pelos usuários e as sugestões propostas pelos mesmos, constatou-se que houve motivação por parte deles na utilização do sistema, comprovando que a introdução do computador no processo multidisciplinar de educação tende a impulsionar novos paradigmas no processo de ensino/aprendizagem.

Entretanto, observou-se que o sistema foi muito bem avaliado pelos professores. Já com relação aos alunos, o sistema não agradou a todos. Isso se deve ao fato dos alunos terem uma dificuldade maior de abstração e os professores apresentarem uma melhor visão da aplicabilidade da ferramenta.

7.5. Considerações Finais

Este capítulo apresentou a avaliação do sistema desenvolvido para o estudo de caso proposto. A preocupação inicial quanto ao sistema desenvolvido foi a simples eficácia do modelo, ou seja, verificar se a tecnologia escolhida propiciava a distribuição dos mundos. Historicamente, esta verificação se deu em quatro etapas descritas a seguir:

1 – Distribuição de dois ambientes distintos, permitindo uma comunicação unidirecional (o ambiente virtual de Biologia localizado em um computador X age sobre o ambiente virtual de Química em um computador Y).

2 – A evolução da fase 1, mudando de comunicação unidirecional para comunicação

bidirecional (o ambiente virtual de Biologia localizado em um computador X alterando e sofrendo alterações do ambiente virtual de Química em um computador Y).

3 – A inserção de outras cópias de um dos ambientes sujeitos às interações de outros (um ambiente virtual de Biologia localizado em um computador X, interagindo com vários ambientes virtuais de Química localizados em computadores indefinidos).

4 – E, por último, a comunicação entre múltiplos ambientes virtuais de cada área, com a replicação de todas as alterações em todas as cópias.

Após a verificação desta eficácia, foi necessário analisar o sistema desenvolvido, de acordo com as características de comunicação em rede, modelo de visão, modelo de dados, gerenciamento da computação e comportamento dos objetos. Cada um dos protótipos foi construído com variações no modelo de dados, para propiciar uma comparação da melhor abordagem de distribuição dos ambientes. Estes protótipos foram testados e avaliados de acordo com o principal mecanismo de análise de AVDs, que é a escalabilidade, e também a extensibilidade, pela peculiaridade do sistema.

Quanto aos resultados destes dois mecanismos, foi possível provar a aplicabilidade desta metodologia e definir a melhor abordagem para sistemas virtuais distribuídos com as mesmas características do modelo proposto.

Um dos protótipos foi usado para avaliação com professores, pesquisadores e alunos de ensino médio e superior, causando principalmente nos professores e pesquisadores uma motivação quanto ao uso do sistema, sugerindo a aplicação imediata do mesmo nas escolas que usaram a aplicação.

CAPÍTULO VIII

8. CONCLUSÕES E TRABALHOS FUTUROS

8.1 Introdução

Este capítulo tem como objetivo apresentar os principais pontos estudados nesta tese, relacionar os possíveis trabalhos futuros advindos desta pesquisa e avaliar a principal contribuição deste trabalho para a área científica.

8.2 Conclusões

Durante a pesquisa, constatou-se que existem diversos ambientes virtuais distribuídos. Porém, a maioria desses ambientes tem uma preocupação quanto à distribuição e comunicação entre réplicas.

Dessa forma, este trabalho apresentou uma aplicação das técnicas de Realidade Virtual não-imersiva de forma distribuída, explorando a multidisciplinaridade por meio da distribuição de AVs distintos e co-relacionados. Sendo assim, foi apresentada uma arquitetura para o sistema proposto e implementado um estudo de caso que apresenta o Processo da Fotossíntese. Para tal, foram utilizadas as linguagens Borland Delphi 6.0 Enterprise, uma implementação CORBA, o Visibroker que acompanha o Delphi Enterprise e a biblioteca gráfica OpenGL, utilizada na modelagem dos ambientes virtuais. Dentro da metodologia proposta para caracterizar e avaliar AVDs, existem várias possibilidades e, para este modelo de aplicação multidisciplinar, o Modelo de Dados é uma abordagem que pode variar muito,

causando resultados diferentes.

Como conclusões da implementação da arquitetura, por meio do estudo de caso apresentado, pode-se citar que:

- ✓ A arquitetura CORBA se mostrou eficaz e capaz de atender requisitos ainda mais complexos na construção de AVDs distintos.
- ✓ AVDs de áreas totalmente distintas (multidisciplinares), porém com interdependência direta ou indireta, podem necessitar da concepção de uma metodologia diferente e específica, não se adequando aos modelos utilizados apenas para distribuir cópias de ambientes virtuais.
- ✓ É necessário elaborar um projeto que envolva um AVD por completo, lembrando que os componentes (objetos) do ambiente possuem relação direta com a distribuição.
- ✓ Finalmente, a implementação do modelo com características diferentes possibilitou a comparação, a análise e a posterior recomendação da melhor abordagem para distribuição de AVs. Apesar do protótipo 1 não apresentar estrutura convencional (espelho), se mostrou como a melhor escolha para distribuir dois modelos de ambientes virtuais numa relação multidisciplinar.

8.2.1. Contribuições do Trabalho

A principal contribuição deste estudo refere-se à definição da melhor arquitetura para desenvolvimento de um Ambiente Virtual Distribuído Multidisciplinar. Esta contribuição pode ser melhor detalhada pelos tópicos que estão ordenados de acordo com a evolução do estudo:

1 – No levantamento efetuado no decorrer deste trabalho, verificou-se que, até o momento da elaboração desta tese, nenhum AVD era propriamente dito uma distribuição de ambientes virtuais de áreas distintas (multidisciplinaridade).

2 – Este trabalho veio ressaltar a eficácia da arquitetura CORBA como camada *middleware* de comunicação, comparando-a com outras existentes, relatando o que resultou em sua escolha.

3 – Devido à diversidade de características e objetivos dos AVDs em uso, este estudo ratifica a metodologia definida por Snowdon (1994) como meio para caracterizar e avaliar AVDs.

4 – Os protótipos construídos também representam uma contribuição, podendo ser utilizados principalmente como ferramenta de ensino.

5 – Outra contribuição, considerada a mais importante deste trabalho, resulta da comparação dos protótipos implementados, definindo a melhor abordagem, quanto ao modelo de dados, para criação de AVDs Multidisciplinares. O protótipo mais eficiente necessitou de um algoritmo auxiliar para que o sistema tolerasse falhas e propiciasse a comunicação eficiente entre ambientes virtuais multidisciplinares. Este algoritmo foi denominado de FAVD. A Figura 8.1 ilustra a evolução desta pesquisa em etapas.

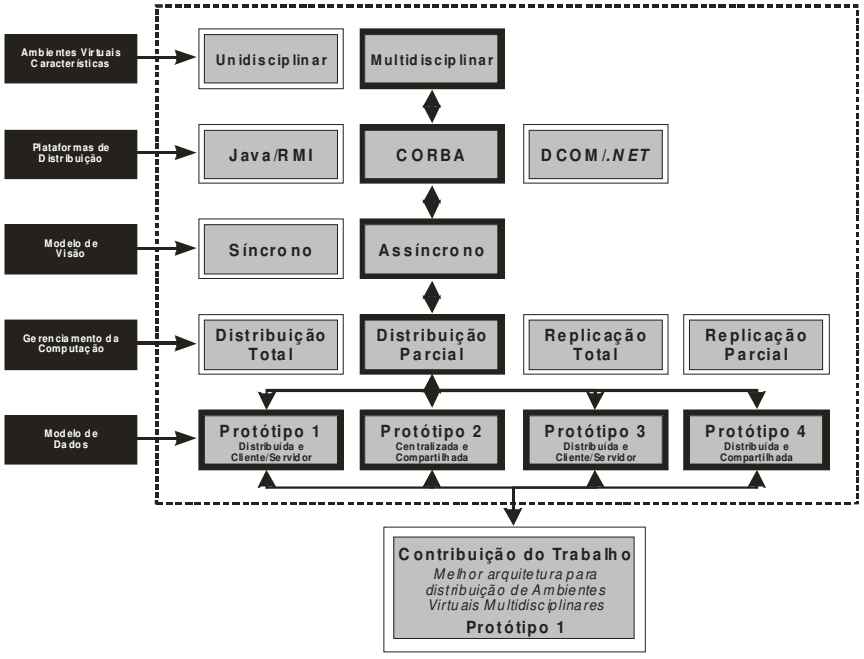


Figura 8.1. Evolução do trabalho.

De acordo com a Figura 8.1, é possível verificar a evolução ocorrida para que fosse possível definir a melhor abordagem para distribuição de ambientes virtuais. Após a adoção de uma metodologia na construção dos protótipos, fica explícito que o Protótipo 1 apresentou os melhores resultados, podendo ser usado como referência na construção de outros ambientes virtuais multidisciplinares distribuídos.

8.3 Trabalhos Futuros

De acordo com a evolução deste trabalho, alguns aspectos foram considerados importantes na consolidação de um produto final com todos os requisitos necessários para utilização do mesmo, tanto no meio acadêmico/científico quanto comercial. O objetivo deste trabalho é validar um aspecto específico, portanto não tem pretensão de resolver todos os problemas ou necessidades advindas desta pesquisa, possibilitando a este estudo ser referência para outros trabalhos futuros. As Tabelas 8.1 e 8.2 mostram uma relação de questões levantadas por este trabalho, algumas solucionadas, e outras sugeridas para trabalhos futuros.

Tabela 8.1. Questões levantadas no decorrer do trabalho.

Aspectos	Requisito/ Conseqüência	Abordado	Trabalhos Futuros	Relação
Inserir ou remover folhas das árvores	Mudança no processo da fotossíntese	Não	Recomendado	Estudo de Caso
Inserir raízes e frutos nas árvores	Mudança no processo da fotossíntese	Não	Recomendado	Estudo de Caso
Simular chuva, mostrando a retirada de nutrientes do solo.	Mudança no processo da fotossíntese	Não	Recomendado	Estudo de Caso
Mostrar relação das ações dos raios UVA e UVB	Mudança no processo da fotossíntese	Não	Recomendado	Estudo de Caso
Fotossíntese na fase escura (bioquímica)	Alteração das características do ambiente de Biologia	Não	Recomendado	Estudo de Caso
Especificar o tipo de clima (verão, outono, inverno e primavera)	Alteração das características do ambiente de Biologia	Não	Recomendado	Estudo de Caso
Transformar as aplicações em jogos educacionais mantendo o mesmo estudo de caso		Não	Recomendado	Estudo de Caso

Tabela 8.2. Questões levantadas no decorrer do trabalho.

Aspectos	Requisito/ Conseqüência	Abordado	Trabalhos Futuros	Relação
Criar comunicação por meio de mensagens automáticas e digitadas entre os participantes		Não	Recomendado	Aplicação
Criar um administrador do sistema para coordenar as interações entre os participantes		Não	Recomendado	Aplicação
Adaptar o sistema para o funcionamento em situações de ensino a distância (EAD)		Não	Recomendado	Aplicação
Adaptar o sistema para avaliação de conteúdo	Criar banco de dados para armazenar as experiências no sistema	Não	Recomendado	Aplicação
Utilizar invocação dinâmica da arquitetura de distribuição		Não	Recomendado	Arquitetura CORBA
Utilizar serviços tais como o Serviço de Nomeação e de Tempo para melhorar o desempenho do AVD		Não	Recomendado	Arquitetura CORBA
Incorporar o mecanismo *dead-reckoning* para minimizar latência da comunicação	Incorporar comportamento inteligente dos objetos	Não	Recomendado	Realidade Virtual Distribuída
Incorporar detecção de colisões entre os objetos	Incorporar comportamento inteligente dos objetos	Não	Recomendado	Realidade Virtual Distribuída
Transformar a aplicação de não-imersiva para imersiva incorporando dispositivos multi-sensoriais (HMD e *data gloves*)	-	Não	Recomendado	Realidade Virtual Distribuída
Aumentar a quantidade de tipos ambientes virtuais	-	Parcialmente	Recomendado	Realidade Virtual Distribuída
Transformar a aplicação de não-imersiva para Realidade Aumentada		Não	Recomendado	Realidade Virtual Distribuída
Comparar diversos modelos de Gerenciamento de Computação	-	Não	Recomendado	Realidade Virtual Distribuída
Comparar modelos de comportamentos de objetos	Mudança no processo da fotossíntese	Não	Recomendado	Realidade Virtual Distribuída

As Tabelas 8.1 e 8.2 demonstram a existência de várias ramificações e que seria quase impossível para um único trabalho abranger todos os aspectos envolvidos, encontrar todas as soluções e propor melhorias. Partindo do princípio que a pesquisa é dinâmica e que não existe nenhum produto totalmente acabado ou pronto, ou que não necessite de alguma

evolução ou melhoria, este trabalho demonstra os diversos caminhos que podem ser objetos de trabalhos futuros, ratificando o potencial da área de pesquisa em questão.

REFERÊNCIAS BIBLIOGRÁFICAS

ALIMA, L.; GHODSI, A.; BRAND, P.; HARIDI, SEIF. Multicast in DKS(N, k, f) Overlay Networks. In: Principles of Distributed Systems (OPODIS'2003), 7, 2003. *Proceedings*… Berlin: Springer-Verlag, 2004, p37-48.

ANTUNES, Claudia et al. Realidade Virtual em Subdivisão. In: 9º ENCONTRO PORTUGUÊS DE COMPUTAÇÃO GRÁFICA, 2000, Marinha Grande. Disponível em: <http://virtual.inesc.pt/9epcg/actas/resumos.html#a5>. Acesso em: 20 jan. 2005.

ARAÚJO, R. B.; KIRNER, C. Especificação e Análise de um Sistema Distribuído de Realidade Virtual. XIV SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES, Fortaleza, 1996. *Anais...* Fortaleza: SBC, 1996, p.24-35.

AUKSTAKALNIS, S; BLATNER, D. *Silicon Mirage:* The Art and Science of Virtual Reality. Berkeley: Peatchpit Press, 1992. 235p.

BETZ, M. Interoperable Object. *Dr. Dobb's Journal,* San Mateo, CA, USA, p.18-39, Oct. 1994.

BRUTZMAN, D.; MACEDONIA, M. R.; ZYDA, M. J.; Internetwork infrastructure requirements for virtual environments. Monterey:[s.n.], 1995. Disponível em: <http://www.fakespace.com/products/rave.html>. Acesso em: 15 jul. 2004.

BYRNE, C. M. *The Use of Virtual Reality as Educational Tool.* Washington University. Disponível em <http://www.hitl.washington.edu/publications>. Acesso em: 10 jun. 2005.

CAMACHO, M. de L. A. S. M. *Realidade Virtual e Educação*. Lisboa, Universidade Aberta. Disponível em <http://phoenix.sce.fct.unl.pt/simposio/30.htm>. Acesso em: 10 jun. 2005.

CARDOSO, A. *Uma Arquitetura para Elaboração de Experimentos Virtuais Interativos Suportados por Realidade Virtual Não-Imersiva*. 2002, 163f. Tese (Doutorado em Realidade Virtual) Escola Politécnica da USP, São Paulo, 2003.

CARLSSON, C.; HAGSAND, O. DIVE - A Multi-User Virtual Reality System. In: IEEE Virtual Reality Annual International Symposium. 1993, Washington, DC, USA. *Proceedings of the IEEE VRAIS'93 Conference,* 1993. p.394-400.

CATER, J. P.; HUFFMAN, S. D. Use of the Remote Access Virtual Environment (RAVEN) for Coordinated IVA-EVA Astronaut Training and Evaluation. *Presence,* v.4, n.1, p.103-109, 1995.

CODDELA, C. et al. A Toolkit for Developing Multi-User Distributed Virtual Environments. In: IEEE Virtual Reality Annual International. 1993, Washington, DC, USA. *Proceedings of the IEEE VRAIS'93*, 1993. p. 401-407.

COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T., *Distributed Systems*: Concepts and Design. 4th Edition. Addison-Wesley/Pearson Education, 2005, 944 p.

CRUZ-NEIRA, C. et al. The CAVE Audio Visual Experience Automatic Virtual Environment. *Communication of the ACM*, Jun. 1992. p.64-72.

DEERING, S. Mbone – The Multicast Bone. California Education and Research Federation Network, San Diego, CA, USA, 1993. *Proceedings...* San Diego, CA, USA: CERFnet Seminar, 1993, p.15-24.

DERIGGI JR, F.; SEMENTILLE, A. C.; KUBO, M. M.; KIRNER, C. A Utilização da Plataforma CORBA como Suporte para Aplicações Distribuídas de Realidade Virtual. In: Workshop de Realidade Virtual – WRV'97. 1, 1997, São Carlos-SP. *Anais...* São Carlos-SP: UFSCar, 1997, p.1-10.

DERIGGI JR., F.; KUBO, M. M.; SEMENTILLE, A. C.; SANTOS, S. G.; BREGA, J.R.F.; KIRNER, C. CORBA Platform as Support for Distributed Virtual Environments. In: IEEE Virtual Reality, 3, 1999, Houston, Texas. *Proceedings...* Houston, Texas: IEEE, 1999, p.13-17.

DIZERO, Wagner José; VICENTIN, Verison José e KIRNER, Claudio. *Estudo de Interação para um Sistema de Ensino à Distância Baseado em Interfaces de Realidade Virtual*. Universidade Federal de São Carlos, São Paulo, 1999. Disponível em <http://www.unicamp.br/~ihc99/Ihc99/AtasIHC99/AtasIHC98/Dizero.pdf>. Acesso em: 15 jun. 2004.

ECKHOUSE JR., R. H. et al. Issues in Distributed Processing – An Overview of Two Workshops. *Computer*, New York, USA, v. 11, n. 1, p22-26, dez. 1978.

EL-ANSARY, S.; ONANA, L.; BAND, P.; HARIDI, S. *A Frame work for P2P lookup services based on k-ary search.* Distributed System Laboratory, 2002. Disponível em: <http://www.sics.se/pepito/D2.7/papers/paper1.pdf>. Acesso em: 10 jan. 2005.

ENSLOW JR., P. H. (ed.) Multiprocessors and Parallel Processing. New York, John Wiley & Son, 1974, 340p.

GARCIA, P.; MONTALA, O.; PAIROT, C.; RALLO, R.; SKARMETA, A. G. MOVE: component groupware foundations for collaborative virtual environments. In: Collaborative Virtual Environments. 4, 2002, Boon, Germany. *Proceedings...* New York, USA: ACM Press, 2002, p.55-62.

GIRARDELLI, M. F. *Qual é a diferença entre multidisciplinaridade, interdisciplinaridade e transdisciplinaridade?*. Disponível em <http://www.uol.com.br/novaescola/ed/124_ago99/html/comcerteza_didatica.htm>. Acesso em: 20 maio 2005.

GREENHALGH, C.; PURBRICK, J.; SNOWDON, D. Inside MASSIVE-3: flexible support for data consistency and world structuring. In: INTERNATIONAL CONFERENCE ON COLLABORATIVE VIRTUAL ENVIRONMENTS, 3, 2000, San Francisco, California. *Proceedings...* New York: ACS PRESS, 2000, p.119-127.

HAWKES, R. *A Software Architecture for Modeling and Distributing Virtual Environments.* 1996. 187f. Tese (Doutorado em Ciência da Computação) - University of Edinburg, Edinburg, 1996.

HEBERT, A.; CHEN, A. A new collaborative software package: TeamSpace at Stanford University. In: User Services Conference, 33, 2005, Monterey, CA, USA. *Proceedings of the 33rd annual ACM SIGUCCS conference on User services.* New York, USA: ACM Press, 2005, p109-112.

IONA TECHNOLOGIES LTD. Orbix 2: 2 Programing Guide, 1997.

ISO/IEC 9126.. Software Product Evaluation - Quality Characteristics and Guideline for their Use. *International Standards Organization*, 1991.

JALOTE, P. Fault Tolerance in Distributed Systems*. Prentice Hall PTR*, Englewood Cliffs, New Jersey, 1994.

JANSSEN, B.; SPREITZER, M. *ILU 2.0 Alpha8 Reference Manual.* XEROX Corporation. Disponível em: <ftp://ftp.parc.xerox.com/pub/ilu/2.0/20a8-manual-html/manual_toc.html>. Acesso em: 15 jun. 2005.

JING, Zhang; JIANZHONG, Liu; GANG, Wan; YANBIN, Guo. *Researches on Model of Navigation in CVE.* Disponível em: <http://www.gisdevelopment.net/technology/lbs/pdf/ma04194.pdf>. Acesso em: 13 jun. 2005.

KALAWSKY, R. S. Exploring Virtual Reality Techniques in Education and Training: Technological Issues. *Advanced VR Research Centre*, Lough-borough, University of Tecnology, 1993.

KAMARA, S.; DAVIS, D.; BALLARD, L.; CAUDY, R.; MONROSE, F. An Extensible Platform for Evaluating Security Protocols. In: Annual Simulation Symposium, 38, 2005, Washington, DC, USA. *Proceedings...* Washington, DC, USA: IEEE Computer Society, 2005, p.204-213.

KIRNER, C. MENDES. Sueli B. T. *Sistemas Operacionais Distribuídos - Aspectos gerais e análise de sua estrutura.* Rio de Janeiro: Campus, 1988.

KIRNER, C. et al. Projeto AVVIC: Ambiente Virtual para Visualização Interativa Compartilhada. Disponível em: < http://www.dc.ufscar.br/~grv/avvic.htm>. Acesso em 15 jul. 2004.

KIRNER, C. et al. AVVIC: virtual environment for shared interactive visualization. In: PROTeM-CC PHASE III PROJECTS - INTERNATIONAL EVALUATION, May 1999, Rio de Janeiro. *Proceedings...* Brasília: CNPq, 1999. p.177-190.

KIRNER, C. Realidade Virtual: Dispositivos e Aplicações. VII ESCOLA DE INFORMÁTICA DA SBC REGIONAL SUL, Maio/1999. p.135-158.

KIRNER, C. *Sistemas de Realidade Virtual.* Disponível em: <http://www.dc.ufscar.br/~grv/tutrv>. Acesso em: 18 jan. 2005.

KOTZIAMPASIS, I.; SIDWELL, N.; CHALMERS, A. Virtual reality II: Portals: increasing visibility in virtual worlds. In: Spring Conference on Computer Graphics. 19, 2003, Budmerice, Slovakia. *Proceedings…* New York, USA: ACM Press, 2003, p.257-261.

KOTZIAMPASIS, I.; SIDWELL, N.; CHALMERS, A. Virtual Environments: Seamlessly integrated distributed shared virtual environments. In: Spring Conference on Computer Graphics. 20, 2004, Budmerice, Slovakia. *Proceedings…* New York, USA: ACM Press, 2004, p.138-147.

KUROSE, J. F.; ROSS, K. W.; *Redes de Computadores e a Internet*: Uma Nova Abordagem. 1. ed. São Paulo: Addison Wesley, 2003. 548p.

LAMPORT, R. Lower Bounds for Asynchronous Consensus. *Future Directions in Distributed Computing,* 2003. p. 22-23.

LEAVENS, G. T.; CHEON, Y. Extending CORBA to Specify Behavior with Larch. *Technical Report*. Departament of Computer Science , Iowa State University, Aug. 1993.

LIMA, R. R. *Tutorial CORBA*. Disponível em: <http://www.mundooo.com.br/php/mooartigos.php?pa=showpage&pid=17>. Acesso em: 18 jan. 2004.

LINHARES, S.; GEWANDSZNAJDER, F. *Biologia Hoje.* São Paulo: Editora Érica, 2000. 520p.

LOPES, A. V. *Estrutura de dados para construção de software.* Canoas: Ed. ULBRA, 2002. 640 p.

LUI, J. C. S.; CHAN, M. F. An efficient partitioning algorithm for distributed virtual environment systems. *IEEE Transactions on Parallel and Distributed Systems,* New York, USA, v. 13, n. 3, p.193-211, mar. 2002.

MACEDONIA, M. R.; ZYDA, M. J. *A Taxonomy for Networked Virtual Environments*. Disponível em: <http://doi.ieeecomputersociety.org/10.1109/93.580395>. Acesso em: 18 jan. 2005.

MAFFEIS, S.; SCHMIDT, D. C. Constructing Reliable Distributed Communication Systems with CORBA. *IEEE Communications Magazine,* v. 35, n.2, p.56-61, feb. 1997.

NELSON, V. P. *Remote Procedure Call.* 1981. 192f. Tese (Doutorado em Ciência da Computação) - Department of Computer Science, Carnegie-Mellon University, 1981.

NEVES JUNIOR, I. N. *CORBA*. Disponível em: <http://www.gta.ufrj.br/grad/inon/corba.htm>. Acesso em: 18 jan. 2004.

NPSNET, Naval Postgraduate School. NPSNET Research Group Monterey: [s.n.]. Disponível em: <http://www.npsnet.org/NPSNET-V>. Acesso em: 10 nov. 2004.

NUNES, C. *Uma Aplicação de Técnicas de Realidade Virtual Não-Imersiva no Processo Multidisciplinar de Educação*. 2002. 72f. Dissertação (Mestrado em Ciências – Realidade Virtual) Depto. Laboratório de Computação Gráfica, Faculdade de Engenharia Elétrica, Universidade Federal de Uberlândia, UFU, Uberlândia, 2002.

OMG e o Padrão CORBA. Disponível em: <http://www.corba.hpg.ig.com.br/>. Acesso em: 15 jan. 2004.

OST, L. C.; NEDEL, Luciana. *Realidade Virtual Distribuída. Disponível em:* <http://www.inf.ufrgs.br/~nedel/cmp513/13-distributed-vr-p.pdf>. Acesso em: 12 dez. 2004.

PARDO, R. *Interprocess Communication and Synchronization for Distributed Systems*. 1979. 229f. Tese (Doutorado em Ciência da Computação) - Dept. of Computer and Information Science, The Ohio State University, 1979.

PAULINO, W. R.; BARROS, C. *Os Seres Vivos*. São Paulo: Editora Ática, 2001. 279p.

PAULOVICH, F. V. *Middlewares em Sistemas Distribuídos*. Disponível em: <http://www.dc.ufscar.br/~paulovic/MidSDs.pdf/>. Acesso em: 15 jul. 2004.

PIAGET, J. The Epistemology of interdisciplinary relationships. *INTERDISCIPLINARITY*. 1972. p129.

PINHO, M. S. *Manipulação Simultânea de Objetos em Ambientes Virtuais Imersivos*. 2002. 174f. Tese (Doutorado em Ciência da Computação) – Universidade Federal do Rio Grande do Sul, UFRGS, Rio Grande do Sul, 2002.

PINHO, M. S. *Biblioteca Gráfica OpenGL*. Disponível em: <http://www.inf.pucrs.br/~pinho/CG/Aulas/OpenGL/OpenGL.html>. Acesso em: 18 jul. 2004.

RAJ, G. S. *A Detailed Comparison of CORBA, DCOM and Java/RMI*. Disponível em: <http://my.execpc.com/~gopalan/misc/compare.html>. Acesso em: 2 out. 2004.

RICCIONI, P. R. *Introdução a Objetos Distribuídos com CORBA*. Florianópolis: Visual Books, 2000. 104 p.

ROEHL, B. *Some Thoughts on Behaviour in VR Systems*. Disponível em: <http://sunee.uwaterloo.ca>. Acesso em: 11 jan. 2004.

SEMENTILLE, A. C. *A Utilização da Arquitetura CORBA na Construção de Ambientes Virtuais Distribuídos*. 1999. 186f. Tese (Doutorado em Física Computacional) – Instituto de Física de São Carlos, USP, São Carlos, 1999.

SEMENTILLE, A.C. BREGA, J.R.F., KIRNER, C., KUBO, M.M.. Ambientes Virtuais Distribuídos usando CORBA*:* Um Estudo de Caso. In: III Workshop de Realidade Virtual – WRV´2000, 3, 2000, Gramado-RS. *Proceedings…* Gramado-RS, 2000. p.145-156.

SIEWIOREK, D. P. Architecture of Fault-Tolerance Computers. *Computer,* 17(8):8-18, aug. 1984.

SINGHAL, S. K.; ZYDA, M. *Networked Virtual Environment.* Disponível em: <http://www.npsnet.nps.navy.mil>. Acesso em: 25 jul. 2004.

SIQUEIRA, L. L.; RIBEIRO, M. W. S.; LAMOUNIER, E.; CARDOSO, A. Estudo Comparativo Entre Plataformas de Suporte a Ambientes Virtuais Distribuídos. In: Workshop de Aplicações de Realidade Virtual, 1, 2005, Uberlândia. *Anais...* Uberlândia-MG: UFU, 2005. 1 CD-ROM.

SNOWDON, D. N.; WEST, A. J. Aviary: Design Issues for Future Large-Scale Virtual Environments. *Presence,* v. 3, n. 4, p.228-308, 1994.

SPECTOR, A. Z. *Multiprocessing Architectures for Local Computer Networks*. 1981, 116f. Tese (Doutorado em Ciência da Computação) - Dept. of Computer Science, Stanford University, Stanford, 1981.

STYTZ, M. R. Distributed Virtual Environment. *IEEE Computer Graphics and Applications*, Los Alamitos, CA, USA, v. 16, n. 3, p.19-31, may 1996.

TANEMBAUM, A. S. Distributed Operating Systems. *Prentice Hall International Editions*, New Jersey, 1995.

TEIXEIRA, S. *Delphi 6, o guia do desenvolvedor*. Rio de Janeiro: Campus, 2002.

TOURAINE, D.; BOURDOT, P.; BELLIK, Y.; BOLOT, L. A framework to manage multimodal fusion of events for advanced interactions within virtual environments. In: ACM International Conference Proceeding Series, 23, 2002, Barcelona, Spain. *Proceedings of the workshop on Virtual Environments.* Aire-la-Ville, Switzerland: EUROGRAPH, 2002, p.159-168.

TREICHEL, P. Jr.; KOTZ, J. C. *Química e reações químicas.* V. 1. Rio de Janeiro: Editora LTC, 1998.

VINOSKI, S. CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments. *IEEE Communications Magazine,* Washington, DC, v.14, n.2, p.1-12, feb. 1997.

WANG, J.; KESHAV, S. Efficient and accurate Ethernet simulation.*Technical Report TR99-1749*. Department of Computer Science, Cornell University, May 20, 1999.

WLOKA, M. M. Lag in Multiprocessor VR. *Presence,* v.4, n.1, Spring 1995.

WOO, M. et al. *OpenGL Programming Guide:* the official guide to learning OpenGL, 3.ed. Reading, Massachusetts: Addison Wesley, 1999. 730 p.

WRIGHT, Richard S. Jr.; SWEET, Michael. *OpenGL SuperBible.* 2nd ed. Indianapolis, Indiana: Waite Group Press, 2000. 696 p.

YARDI, S.; HILL, B.; CHAN, S. Decision-making and communication: VERN: facilitating democratic group decision making online. In: International ACM SIGGROUP conference on Supporting group work, 3, 2005, Sanibel Island, Florida, USA. *Proceedings...* New York, USA: ACM Press, 2005, p.116-119.

GLOSSÁRIO

3D - Três dimensões (altura, largura e profundidade), tridimensional. Vide Espaço tridimensional.

Ambiente Virtual – É um ambiente tridimensional simulado, criado por aplicações de realidade virtual, no qual um usuário pode interagir.

Ambiente Virtual Distribuído – É um sistema em que diversos usuários interagem entre si e sem tempo real, sendo que estes podem estar situados em localidades diferentes. Tipicamente, cada usuário acessa seu próprio computador, usando-o para fornecer uma *interface* para o ambiente virtual.

API – *(Application Programming Interface) Interface* de programação de aplicativos. É um conjunto de funções definidas para permitir o acesso a um sistema. Por exemplo, o sistema operacional *Windows* possui centenas de APIs, contidas em bibliotecas de ligação dinâmica (DLLs), às quais um programa tem acesso quando é executado.

Ativação – Preparar um objeto para executar uma operação. Por exemplo, copiando a forma persistente dos métodos e dados armazenados para um espaço de endereçamento executável, a fim de permitir a execução dos mesmos.

Broadcast – É um sistema que entrega um dado pacote para todos os computadores *(hosts)* que estão conectados a ele. Pode ser implementado em *hardware* ou *software*.

Capacete de Realidade Virtual - *Head Mounted Display – HMD*. Um dispositivo que é firmado à cabeça e que é usado para exibir uma cena gerada em um computador. Uma imagem exibida neste capacete provê tipicamente uma visão estéreo ótica (3D) pelo uso dos dois *Liquid Crystal Display (LCD)* ou exibição em *Cathode-Ray Tube* (CRT) pequenas. Nomes de marcas incluem: *EyePhone (VPL Research), Visette (W-Industries), Private Eye (Reflection Technologoes),* dentre outras.

CAVE – (Cave *Automatic Virtual Environment)* É um ambiente virtual alternativo, que utiliza telas de projeção para a visualização, em especial, de aplicações científicas.

Cena - É o mundo virtual que é exibido, em um ambiente virtual.

Classe - É uma entidade que congrega a definição de dados e das operações (funções ou métodos) que manipulam estes dados. Uma variável de uma classe é chamada objeto e conterá campos de dados e funções. A simples definição de uma classe não cria um objeto.

Cliente - É o código ou processo que invoca uma operação em um objeto.

Cliente/Servidor - É um modelo de interação em um sistema distribuído, no qual um programa (ou objeto) em uma localidade envia um pedido a um programa (ou objeto) em outro lugar, e espera por uma resposta. O programa requisitante é chamado cliente; o programa que satisfaz o pedido é chamado de servidor.

Comportamento – No contexto da orientação a objetos, corresponde aos efeitos observáveis de um objeto executando uma operação requisitada, incluindo seus resultados.

Conexão – É o caminho entre dois módulos de protocolo, o qual fornece um serviço de fluxo de dados confiável. Na internet, uma conexão estende-se de uma módulo TCP de um computador à um módulo TCP de outro.

CORBA – *Common Object Request Broker Architecture*. É uma arquitetura criada pela *Object Management Group (OMG),* a qual especifica como objetos distribuídos podem cooperar utilizando uma rede de computadores, sem se preocupar com os sistemas operacionais de clientes e servidores, nem com linguagens de programação. A arquitetura CORBA é uma plataforma completa para objetos distribuídos. Ela estende os limites das redes, linguagens, componentes e sistemas operacionais.

Criação de um objeto - Um evento que causa a existência de um objeto que é distinto de qualquer outro objeto.

Data glove – É uma luva dotada de sensores, a qual executa um sistema de reconhecimento de movimentação de dedos e posição da mão. O sistema traduz a atividade da mão e dos dedos em sinais eletrônicos, que o computador utiliza com objetivos de rastreamento de posição e controle de entrada de dados.

DCE – (*Distributed Computing Environmente*) É um ambiente criado pela *Open Software Foundation (*OSE), que pode servir como plataforma para execução de aplicações distribuídas. Este ambiente é construído no topo de sistemas operacionais, tais como UNIX, *Windows* e VMS. A idéia é que um usuário pode tomar uma coleção de máquinas existente,

adicionar o software DCR, e então ser capaz de executar aplicações distribuídas, sem perturbar as aplicações (não distribuídas) existentes.

Dead-Reckoning – É uma técnica utilizada para minimizar a latência de comunicação em AVDs. Consiste em fazer que um processo execute a completa simulação de um determinado objeto, processando um modelo de comportamento deste objeto, em paralelo, e quando os dois diferem de uma quantidade pré-definida, uma cópia das variáveis de estado reais daquele objeto é enviada para todos os processos-réplica. As subseqüentes são, então, baseadas na última atualização.

Distinto – Diferente; separado; que não se confunde. Especificamente para este trabalho, a palavra distinto foi usada para caracterizar diferentes ambientes virtuais relacionados a áreas ou contextos diferentes.

DLL – (Dynamic Link Libraries) São bibliotecas de dados ou programas, que podem ser chamados ou acessadas por qualquer aplicativo **Windows**. Um arquivo DLL pode usar a extensão dll ou exe.

Escalabilidade – A escalabilidade, para os Ambientes Virtuais Distribuídos, pode incluir diversas medidas. Comumente pode ser mensurada pelo número de entidades que podem, simultaneamente, participar do sistema. Pode, também, ser mensurada pelo número de computadores que podem se conectar simultaneamente a um AVD.

Estado – Representa a variação das propriedades de um objeto, no tempo. Esta variação afeta o comportamento do objeto.

Exceção – Uma condição causada por uma tentativa de executar uma operação inválida.

Extensibilidade – É a característica de uma arquitetura capaz de suportar utilizações imprevistas e se adaptar aos novos requisitos do desenvolvedor. Como exemplo neste trabalho, a inserção de outros ambientes virtuais.

Facilidades Comuns CORBA – São coleções de componentes escritos em IDL, as quais definem as regras de engajamento para os objetos de aplicação. Elas podem ser do tipo horizontal ou vertical.

Facilidade AVD - Um algoritmo com regras criadas para este trabalho com a finalidade de proporcionar funcionalidade ao sistema.

Geometria - A descrição de um objeto em termos de suas dimensões.

Herança de *Interface* - É a construção de uma *interface* pela modificação incremental de outras *interfaces*. A linguagem IDL provém a herança de *interface.*

Herança múltipla – É a construção de uma definição pela modificação incremental de mais do que uma outra definição.

Herança Simples – É a construção de uma definição pela modificação incremental de uma definição.

HMD - vide Capacete de Realidade Virtual.

IDL – (*Interface Definition Language*) É uma linguagem que permite a definição de *interfaces*. A definição de *interface* especifica as operações que um objeto está preparado para executar, os parâmetros de entrada e saída que requerem, e qualquer exceção que possa ser gerada durante sua execução.

Implementação – Uma definição que provém a informação necessária para criar um objeto e permitir ao mesmo fornecer um conjunto de serviços. Uma implementação tipicamente inclui uma descrição da estrutura de dados usada para representar o estado associado com um objeto, bem como, as definições dos métodos que acessam aquela estrutura de dados.

Instância – Um objeto é uma instância de uma *interface* se fornece as assinaturas e as semânticas das operações especificadas para aquela *interface.* Um objeto é uma instância de uma implementação, se seu comportamento é fornecido por aquela implementação.

Interface – É uma lista de operações e atributos que um objeto fornece. Isto inclui as assinaturas das operações, e os tipos dos atributos. Uma definição de *interface* idealmente inclui sua semântica. Um objeto satisfaz uma *interface* se puder ser especificado como o objeto alvo em cada potencial pedido descrito pela *interface.*

Interoperalidade - É a habilidade de trocar funcionalidade e dados interpretáveis entre duas entidades de *software*. Pode ser definida em termos de quatro requisitos: comunicação, geração de pedidos, formato de dados, e semântica. As entidades de *software* requerem um

canal de comunicação com um protocolo de comunicação comum. Atreves deste canal as entidades necessitam ser capazes de formular e transmitir um pedido interpretável para funções ou dados. O resultado do pedido deve retornar para a entidade solicitante. O intercâmbio de dados também implica em um requisito para um formato de dados que possa ser analisado pelas entidades. As entidades devem entender o pedido de dados através de alguma forma de tradução semântica.

Invocação dinâmica – Construir e invocar um pedido cuja assinatura não é conhecida até o momento da execução.

Invocação Estática - É a construção de um pedido em tempo de complicação. Corresponde a chamar uma operação via um procedimento *Stub.*

Latência - É o tempo requerido pela rede para transferir um bit de dado de um ponto a outro. A latência aparece por muitas razões. Primeiro, a transmissão de dados é governada pelo atraso da velocidade da luz. Um dado, fundamentalmente, não pode ser transportado mais rápido do que a velocidade da luz. Segundo, atrasos são introduzidos pelos próprios computadores. Leva tempo para o dado atravessar o sistema operacional, o *hardware* de rede e, então, encontrar a rede; similarmente o dado deve atravessar os mesmos níveis, ao chegar ao computador destino. Terceiro, atrasos são introduzidos pela própria rede. Geralmente os *hosts* emissor e destino não estão ligados diretamente. Ao invés disto, o dado deve passar por diversas interseções de rede, ou roteadores.

Método - É uma implementação de uma operação. É o código que pode ser executado para realizar um serviço requisitado. Os métodos associados com um objeto podem ser estruturados em um ou mais programas.

Middleware - Em um sistema distribuído constituído de uma hierarquia de níveis de abstração, corresponde as camadas intermediárias desta hierarquia, geralmente localizadas entre a camada de transporte e a camada de aplicação.

Modelagem - Criação de uma forma geométrica representando um objeto tridimensional.

Multicast - Uma técnica que permite que cópias de um pacote possa ser passado para um subconjunto selecionado de possíveis destinos.

Multidisciplinar – Relação entre duas ou mais ciências ou setores do conhecimento sem que

as disciplinas envolvidas no processo sejam modificadas.

Objeto - É uma combinação do estado e de um conjunto de métodos que explicitamente incorporam uma abstração caracterizada pelo comportamento dos requisitos relevantes. Um objeto é uma instância de uma implementação e uma *interface.* Um objeto modela uma entidade do mundo real, e é implementada como uma entidade computacional que encapsula estado e operações (internamente implementadas como dados e métodos) e responde aos serviços requisitados.

Objeto de Implementação - Um objeto que serve como uma definição de implementação. Objetos de Implementação residem em um Repositório de Implementação, na arquitetura CORBA.

Objeto distribuído - É um componente de um programa distribuído. Um objeto distribuído possui *interfaces* bem definidas, nas quais podem se comunicar com outros computadores de um sistema distribuído.

Objeto *lnterface* - É um objeto que serve para descrever uma *interface.* Objetos *interface* residem em um Repositório de *lnterface,* na arquitetura CORBA.

ORB - *(Object Request Broker)* É um elemento da arquitetura CORBA, que fornece um meio através do qual clientes podem fazer e receber pedidos e respostas.

Pacote - É uma unidade de dado enviada através de uma rede de comutação de pacotes.

Pedido - Um cliente faz um pedido (ou requisição) quando necessita que uma operação seja executada. Um pedido consiste de uma operação e zero ou mais parâmetros atuais.

Pipeline - É uma técnica usada em processadores avançados, onde o processador inicia a execução de uma segunda instrução antes de a primeira ter sido completada. Isto é, diversas instruções podem estar no *pipeline* simultaneamente, cada uma em um diferente estágio de processamento. O *pipeline* é dividido em segmentos e cada segmento pode executar sua operação concorrentemente com os outros segmentos. Quando um segmento completa uma operação, ele passa o resultado ao próximo segmento no *pipeline* e busca a próxima operação do segmento precedente. Os resultados finais de cada instrução emergem no final do *pipeline* em rápida sucessão.

Portabilidade - Característica do sistema de *software* capaz de ser transferido de um computador para outro sem mudanças.

Proxy - Na arquitetura CORBA, quando um objeto requerido por um cliente é remoto, uma referência ao objeto irá se referir a um objeto "substituto" *(proxy)* do objeto remoto. Um *proxy,* portanto, é um representante local, para um objeto remoto.

Realidade Aumentada – É uma combinação da visão do ambiente real com o ambiente virtual.

Realidade Virtual - Campo da Ciência da Computação que trata das simulações 3D em tempo real, interativas, que imitam o mundo real ou fictício, proporcionando um ambiente virtual com o qual um usuário pode interagir.

Repositório de Implementação - *(lmplementation Repository)* É um local de armazenamento para informações de implementação de objetos.

Repositório de *Interface* - *(lnterface Repository)* É um local para armazenamento de informações de *interface.*

Resultados - É a informação retomada ao cliente, a qual pode incluir valores como informação de estado indicando que condições excepcionais foram levantadas, quando tentou se executar o serviço requisitado.

Serviço de Eventos CORDA - O serviço de eventos permite aos componentes no barramento (ORB) armazenarem, dinamicamente, eventos de seus interesses. O serviço define um objeto bem conhecido chamado de "canal de evento", que coleta e distribui eventos entre componentes.

Servidor - É um processo implementando um ou mais operações em um ou mais objetos.

Sistema Distribuído - É um conjunto de elementos de processamento autônomos, interligados por meio de um subsistema de comunicação de topologia arbitrária. A comunicação entre processos residentes em elementos de processamento diferentes é realizada por meio de troca de mensagens.

Skeleton - É um componente específico de *interface* do ORB, o qual assiste a um adaptador de objetos na passagem dos pedidos para os métodos particulares.

Stub - É um procedimento local que corresponde a uma operação simples, a qual invoca.

Telepresença - A habilidade para agir e interagir em um ambiente distante por tecnologia cibernética. Uma experiência eletrônico-analógica a uma experiência fora-do-corpo.

Tempo de Latência - Intervalo de tempo entre um movimento executado pelo usuário e o resultado deste movimento.

Tempo Real - Resposta simultânea do computador ao usuário com pouco ou nenhum atraso no tempo de resposta, dando a impressão de resposta instantânea.

Textura - É um mapa de bits *(bitmap)* de imagem geralmente criado com o propósito de se aplicar imagens complexas a polígonos simples, a fim de aumentar o desempenho de aplicações gráficas tridimensionais.

APÊNDICE A

MODELO DO QUESTIONÁRIO DE AVALIAÇÃO DO SISTEMA PROTÓTIPO QUE SIMULA O PROCESSO DA FOTOSSÍNTESE

Avaliação do Sistema que Simula o Processo da Fotossíntese

Avaliador: _____ Data da Avaliação: ___/____/2005

Nível de Escolaridade:
( ) Ensino Médio ( ) Ensino Superior ( ) Pós-Graduação

Nome do Curso: _____

Principais finalidades de utilização do computador:

_____

_____

Avalie o Programa que Simula o Processo da Fotossíntese por meio da Multidisciplinaridade

I. Quanto à Finalidade (Eficiência):

( ) Muito útil ( ) útil ( ) pouco útil

(Justificativa)

_____

_____

_____

_____

II. Quanto à Interface (Usabilidade):

( ) Fácil entendimento dos comandos
( ) Médio entendimento dos comandos
( ) Difícil entendimento dos comandos

(Justificativa)

_____

_____

_____

III. Quanto à facilidade de uso (Funcionalidade):

( ) Muito intuitivo
( ) Intuitivo
( ) Pouco Intuitivo

(Justificativa)

_____

_____

_____

IV. Quanto aos recursos do Programa, a experiência proposta (Manutenção):

( ) foi integralmente desenvolvida
( ) foi parcialmente desenvolvida
( ) não foi desenvolvida por completo

(Justificativa)

_____

_____

_____

IV.b. Os objetos disponíveis permitem:

( ) conceber a experiência proposta
( ) conceber parte da experiência proposta
( ) não permitem conceber a experiência

IV.c. A relação entre os objetos disponíveis e suas propriedades:

( ) é simples de ser definida e permite o desenvolvimento proposto
( ) não é simples de ser definida, mas permite o desenvolvimento proposto
( ) não é simples de ser definida e não permite o desenvolvimento proposto

IV.d. ( ) Sugiro inserir novos objetos no experimento, tais como:

_____

_____

IV.e. ( ) Sugiro inserir novas relações entre objetos no experimento, tais como:

_____

_____

_____

_____

V. Conseguiu visualizar a relação da Biologia com a Química por meio dessa experiência?
( ) sim
( ) não
( ) em parte
Justifique:

_____

_____

_____

_____

***VI*. O programa permitiu a aquisição de informações úteis a respeito de como funciona o Processo da Fotossíntese (Confiabilidade):**
( ) sim
( ) não
( ) em parte

Exemplifique:

_____

_____

_____

_____

_____

VII. Observações sobre o programa que achar relevante:

_____

_____

_____

_____

VIII. Sugestões Adicionais:

_____

_____

_____

_____

_____

Assinatura do Avaliador: _____