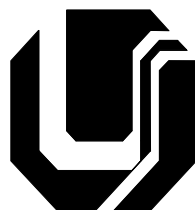


UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA ELÉTRICA
PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA



**ESTUDO COMPARATIVO ENTRE PLATAFORMAS DE
SUPORTE A AMBIENTES VIRTUAIS DISTRIBUÍDOS**

ORIENTADOR: EDGARD LAMOUNIER JÚNIOR, PhD.

CO-ORIENTADOR: ALEXANDRE CARDOSO, Dr.

ORIENTANDO: LUIZ LEONARDO DE SIQUEIRA

**NOVEMBRO
2005**

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

**ESTUDO COMPARATIVO ENTRE PLATAFORMAS DE
SUPORTE A AMBIENTES VIRTUAIS DISTRIBUÍDOS**

Dissertação apresentada por Luiz Leonardo de Siqueira à Universidade Federal de Uberlândia como requisito parcial à obtenção do título de Mestre em Ciências.

Professor Edgard Lamounier Jr, PhD (Orientador).

Professor Alexandre Cardoso, Dr (Co-Orientador).

Professor Marcelo de Paiva Guimarães, Dr, UNASP.

Professor Paulo Roberto Guardieiro, Dr, UFU.

Uberlândia, 24 de novembro de 2005.

ESTUDO COMPARATIVO ENTRE PLATAFORMAS DE SUPORTE A AMBIENTES VIRTUAIS DISTRIBUÍDOS

LUIZ LEONARDO DE SIQUEIRA

Dissertação apresentada por Luiz Leonardo de Siqueira à Universidade Federal de Uberlândia como requisito parcial à obtenção do título de Mestre em Ciências.

Profº Edgard Lamounier Júnior
Orientador

Profº. Darizon Alves de Andrade
Coordenador do Curso de Pós-Graduação

DEDICATÓRIA

Aos meus Pais Hamilton e Dulce

Aos meus irmãos Júnior e Patrícia

As minhas sobrinhas Rebeca e Laura

A minha namorada Carmem

A minha afilhada Júlia

AGRADECIMENTOS

Agradeço primeiramente a Deus, que esteve sempre presente em minha vida ajudando a superar os momentos difíceis e vencer novos obstáculos.

Ao professor e amigo Edgard Lamounier, orientador deste trabalho, pela valiosa contribuição, pela serenidade e compreensão com que sempre demonstrou durante o desenvolvimento deste trabalho.

Ao professor Alexandre Cardoso pelo apoio e amizade na co-orientação desta dissertação.

Ao professor Marcos Wagner pela disponibilidade que demonstrou em ajudar nas horas difíceis vivenciadas durante esta dissertação.

Aos meus Pais, que sempre confiaram em minha capacidade e também todos aqueles que contribuíram de forma direta ou indireta para a realização deste trabalho.

RESUMO

SIQUEIRA, Luiz Leonardo de. Estudo Comparativo Entre Plataformas de Suporte A Ambientes Virtuais Distribuídos, Uberlândia, Faculdade de Engenharia Elétrica - UFU, 2005, XXp.

Esta dissertação apresenta um estudo comparativo do uso da arquitetura CORBA (*Common Object Request Broker Architecture*) e da plataforma .NET como camada intermediária de comunicação entre ambientes virtuais distribuídos. O principal objetivo deste trabalho é desenvolver um protótipo baseado na plataforma .NET e a biblioteca gráfica OPENGL que suporte a comunicação entre dois ou mais ambientes virtuais distintos de tal maneira que ações tomadas em um ambiente alterem a realidade de outros. Para isto foi utilizado como estudo de caso o processo da fotossíntese, tendo dois mundos distintos, um simulando o processo biológico da fotossíntese (Mundo da Biologia) e o outro simulando o processo químico da fotossíntese (Mundo da Química). Dentre os parâmetros utilizados para esta comparação destacam-se: escalabilidade e latência. Além disso, este trabalho foi desenvolvido objetivando o desenvolvimento de ferramentas que possam ser usadas em processos multidisciplinares de Educação. Daí, a busca para o suporte a ambientes virtuais distribuídos.

Palavras-Chave – AVDs, Ambientes Virtuais Distribuídos, Realidade Virtual Distribuída, Sistemas Distribuídos.

ABSTRACT

SIQUEIRA, Luiz Leonardo de. *Estudo Comparativo Entre Plataformas de Suporte A Ambientes Virtuais Distribuídos*, Uberlândia, Faculdade de Engenharia Elétrica - UFU, 2005, XXp.

This study presents a comparative study of the use of the CORBA architecture (Common Object Request Broker Architecture) and the .NET platform as an intermediate communication layer between distributed virtual environments. The main objective of this work is to develop a prototype based of platform .NET and the graphic library OPENGL that allows the communication among two or more distinct virtual environments, in a such way that actions taken in one environment modifies the reality of other environment and vice versa. Thus, the photosynthesis phenomenon has been taken into account as a case study (due to the Biology and Chemistry environments). Scalability and latency are the main parameters used to perform the proposed architecture comparison. Finally, it is important to say this work has been done to provide tools that can support the multidisciplinary education process. Therefore, it aimed to help distributed virtual environments.

Keywords – DVEs, Distributed Virtual Environments, Virtual Reality Distributed, Distributed Systems.

SUMÁRIO

1. INTRODUÇÃO	13
2. SISTEMAS DISTRIBUÍDOS.....	18
2.1. INTRODUÇÃO	18
2.2. O QUE É DISTRIBUIÇÃO?	18
2.3 VANTAGENS E DESVANTAGENS DO SISTEMA DISTRIBUÍDO (SD)	19
2.3.1 <i>Vantagens em relação ao Sistema Central</i>	19
2.3.2 <i>Desvantagens dos Sistemas Distribuídos</i>	20
2.4 MIDDLEWARE	21
2.5. PLATAFORMAS DE DISTRIBUIÇÃO DE SOFTWARE	22
2.5.1 CORBA.....	22
2.5.1.1 Introdução	22
2.5.1.2 Modelo de Objetos da OMG	23
2.5.1.3 Objetos de Serviços	24
2.5.1.4 Facilidades Comuns	26
2.5.1.5 Interfaces de Domínio	27
2.5.1.6 Interface de Aplicação.....	27
2.5.1.7 Objetos de Aplicação.....	28
2.5.2 Componentes da Arquitetura CORBA	28
2.5.2.1 Clientes.....	29
2.5.2.2 ORB (<i>Object Request Broker</i>).....	29
2.5.2.3. Interface ORB	30
2.5.2.4. Interface de Invocação Dinâmica (DII)	30
2.5.2.5. Interface de Esqueleto Dinâmico (DSI).....	30
2.5.2.6. <i>Stub</i>	31
2.5.2.7. <i>Skeleton</i>	31
2.5.2.8. Adaptador de Objeto	31
2.5.2.9 Produtos ORB	31
2.5.3 Plataforma .NET e DCOM.....	32
2.5.3.1 Introdução a plataforma .NET e DCOM	32
2.5.3.2 Estrutura do framework .NET	34
2.5.3.3 CLR – <i>Common Language Runtime</i>	35
2.5.3.4 .NET Framework Class Library	37
2.6 COMPARAÇÃO ENTRE .NET E CORBA.....	37
2.7 BENEFÍCIOS DA PLATAFORMA .NET	38
3. AMBIENTES VIRTUAIS DISTRIBUÍDOS.....	40
3.1. INTRODUÇÃO	40
3.2 SUPORTE À COMUNICAÇÃO DE UM AVD.....	42
3.3 ARQUITETURAS DE SUPORTE À AVDs.....	44
3.3.1 <i>Características do Modelo Cliente-Servidor</i>	44
3.3.2 <i>Características do Modelo Peer-to-Peer</i>	45
3.3.3 <i>Características do Modelo de Computação Distribuída Orientado a Objetos (DOCs)</i>	46
4. ARQUITETURA DO SISTEMA	49
4.1. INTRODUÇÃO	49
4.2 TECNOLOGIAS DE APOIO	49
4.2.1 OPENGL.....	50
4.2.2 Visual Studio .NET.....	52
4.2.3 Linguagem C#.....	52
4.2.4 <i>Genuine Channels</i>	54
4.3 MODELAGEM UML	55
4.4 ARQUITETURA PROPOSTA.....	61
4.4.1 Interface gráfica com o usuário – GUI.....	62
5. DETALHES DA IMPLEMENTAÇÃO.....	69

5.1. INTRODUÇÃO	69
5.1.1. Incorporação da Biblioteca OPENGLE em C#.....	69
5.2 DETALHES DA MODELAGEM.....	70
5.3 DISTRIBUIÇÃO DOS AMBIENTES	74
5.3.1 Implementação do Servidor	75
5.3.2 Implementação do Cliente	76
5.4 CONSIDERAÇÕES FINAIS	78
6. ANÁLISE DOS RESULTADOS	79
6.1 INTRODUÇÃO	79
6.2 DESCRIÇÃO DO AMBIENTE UTILIZADO	79
6.3 ANÁLISE DA LATÊNCIA.....	82
7. CONCLUSÕES E TRABALHOS FUTUROS.....	84
REFERÊNCIAS BIBLIOGRÁFICAS	86

LISTA DE FIGURAS

<i>Figura 1: Arquitetura do Modelo de Referência da OMG. [OMG, 2004]</i>	24
<i>Figura 2. Componentes da arquitetura CORBA. [OMG, 2004]</i>	28
<i>Figura 3. Estrutura do framework .NET.</i>	35
<i>Figura 4. Métodos de Disseminação da Informação [TODESCO, 2000].</i>	43
<i>Figura 5. Arquiteturas Distribuídas [TODESCO, 2000].</i>	44
<i>Figura 6. Diagrama de Caso de Uso do Servidor.</i>	55
<i>Figura 7. Diagrama de Caso de Uso do Mundo da Biologia.</i>	56
<i>Figura 8. Diagrama de Caso de Uso do Mundo da Química.</i>	57
<i>Figura 9. Diagrama de Classes do Mundo da Biologia.</i>	58
<i>Figura 10. Diagrama de Classes do Mundo da Química.</i>	59
<i>Figura 11. Diagrama de Classes do Servidor.</i>	60
<i>Figura 12. Arquitetura Proposta.</i>	61
<i>Figura 13. Interface do Servidor.</i>	63
<i>Figura 14. Mensagens de Aviso.</i>	63
<i>Figura 15. Interface do Mundo da Biologia</i>	64
<i>Figura 16. Interface do Mundo da Química.</i>	65
<i>Figura 17. Mundo da Biologia com 45°C.</i>	66
<i>Figura 18. Início do Processo Químico da Fotossíntese.</i>	67
<i>Figura 19. Inserção do Cianeto no Mundo da Química.</i>	68
<i>Figura 20. Simulação da Morte das Árvores.</i>	68
<i>Figura 21. Parte do Código da Modelagem do Mundo Biologia</i>	71
<i>Figura 22. Parte do Código da Modelagem das Árvores.</i>	72
<i>Figura 23. Código para Geração do Estroma.</i>	73

<i>Figura 24. Código da Interface.....</i>	<i>74</i>
<i>Figura 25. Código de Ativação do Servidor.</i>	<i>75</i>
<i>Figura 26. Código Executado Para Tentar Conectar o Cliente.</i>	<i>76</i>
<i>Figura 27. Código Executado Para Conectar o Cliente com Servidor.</i>	<i>77</i>
<i>Figura 28. Classe QueryPerfCounter.</i>	<i>81</i>
<i>Figura 29. Latência dos Protótipos.....</i>	<i>82</i>
<i>Figura 30. Porcentagem da Latência com Base na Escalabilidade.</i>	<i>83</i>

LISTA DE TABELAS

<i>Tabela 1. Vantagens do Sistema Distribuído</i>	<i>20</i>
<i>Tabela 2. Desvantagens dos Sistemas Distribuídos.</i>	<i>21</i>
<i>Tabela 3. Comparação entre .NET e CORBA. [IONA, 2004].</i>	<i>38</i>
<i>Tabela 4. Benefícios do framework .NET.....</i>	<i>39</i>

CAPÍTULO I

1. INTRODUÇÃO

Realidade Virtual (RV) pode ser definida de forma simples como sendo a forma mais avançada de interface entre o usuário e o computador até agora disponível [HANCOCK, 1995].

Segundo Rodello (2002), apesar da teoria sobre RV não ser muito nova, somente nos últimos anos esta vem apresentando uma grande evolução, onde suas implementações tecnológicas já estão sendo utilizadas por diversas áreas do conhecimento, como por exemplo: medicina, arquitetura, educação, astronomia, simulação, entretenimento entre outros.

A possibilidade de simular situações que de maneira real não seriam possíveis permite um grande avanço em diversos setores militares e empresariais. Na educação, a utilização da RV possibilita a realização de experiências com o conhecimento de forma interativa, permitindo a exploração de ambientes, processos ou objetos, não através de livros, fotos, filmes ou aulas, mas através da manipulação, interação e análise virtual do próprio alvo do estudo.

Multidisciplinaridade 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 multidisciplinaridade 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, 2002].

A multidisciplinaridade é uma ferramenta muito importante no processo da educação, pois permite a substituição de uma concepção fragmentária para uma concepção unitária de um todo, e é neste ponto que se encaixam os Ambientes Virtuais Distribuídos (AVDs).

A motivação para o desenvolvimento de AVDs está no grande crescimento e elevado potencial deste tipo de aplicação, principalmente nas áreas de educação e treinamento. Este tipo de sistema tem permitido que usuários geograficamente dispersos atuem em mundos virtuais compartilhados, usando a rede para melhorar o desempenho coletivo, através da troca de informações [KIRNER,1998].

A maioria dos trabalhos de AVDs são uma distribuição de mundos idênticos, onde a interação em um mundo é vista em tempo real por outros participantes do mesmo. A grande motivação deste trabalho é definir, com base em comparação do protótipo criado, uma melhor plataforma de comunicação entre AVDs distintos para que se possa utilizar no processo de educação interdisciplinar.

O objetivo deste trabalho é comparar a arquitetura CORBA (*Common Object Request Broker Architecture*) e a Plataforma .NET como camada intermediária de comunicação entre ambientes virtuais distintos, utilizando a arquitetura de rede Cliente / Servidor e a biblioteca gráfica OPENG¹ para modelagem dos ambientes. As seguintes metas serão seguidas:

- a) Elaborar um protótipo baseado na Plataforma .NET e na biblioteca gráfica OPENG;
- b) Identificar a arquitetura que apresente maior eficiência na distribuição de ambientes

¹ OPENG é uma biblioteca de rotinas gráficas e de modelagem 2D e 3D. É uma interface (software) para hardware gráfico [Woo 1999].

virtuais distintos;

- c) Efetuar testes entre o protótipo desenvolvido neste trabalho e o protótipo desenvolvido por Ribeiro (2005), baseado na plataforma CORBA, nas mesmas máquinas e utilizando os mesmos parâmetros de testes para evitar discrepâncias nos resultados e validar a comparação teórica.
- d) Avaliar e comparar o desempenho das duas plataformas baseado-se nos parâmetros considerados.

Aplicações de Realidade Virtual (RV) podem ser vistas sob um aspecto bastante amplo e aplicada em várias situações como visto anteriormente. Podemos ter uma única pessoa, usando um único computador, até muitos usuários, usando um sistema distribuído. Os Ambientes Virtuais Distribuídos (AVDs) vêm crescendo e apresentam elevado potencial de aplicação. Esse tipo de sistema permite que os usuários geograficamente dispersos atuem em mundos virtuais compartilhados, usando a rede para melhorar o desempenho coletivo, através da troca de informações.

Existem diversas plataformas que possibilitam a distribuição de mundos virtuais, sendo que a escolha desta plataforma é crucial para um melhor desempenho em rede.

Algumas destas plataformas serão discutidas posteriormente como DCOM (*Distributed Object Component Model*) da Microsoft, CORBA e .NET também da Microsoft.

Um AVD pode ser definido de forma simplificada como um sistema que permite os vários usuários interagirem tanto com o ambiente como entre “eles” em tempo real, mesmo que estes estejam em diferentes localidades geográficas. Os AVDs podem ser classificados em dois tipos, considerando a dimensão do ambiente:

- 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.

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 [MACEDÔNIA, 1994].

A área de aplicação dos AVDs é muito abrangente - treinamento (pilotos, militar), pesquisa, ensino à distância, comércio, cultura, engenharia - mas o maior público e o conseqüente maior investimento destinam-se à área de entretenimento, ou seja, os jogos.

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. Devido à necessidade de comunicação intensiva entre as máquinas do sistema toda vez que há uma atualização de posição, alguns sistemas utilizam a técnica de *dead-reckoning* 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. Normalmente esta técnica é utilizada quando temos ambientes idênticos replicados em máquinas distintas, o que não é o caso deste protótipo, pois vamos utilizar AVDs distintos.

Ambientes virtuais de grande escala como é o caso do sistema de treinamento de guerra NPSNET [MACEDÔNIA, 1994] utilizam a técnica *dead-reckoning*, para diminuir a comunicação pela rede, e com isso não comprometer o desempenho do sistema.

A contribuição deste trabalho é analisar a plataforma CORBA e .NET, para que após coleta dos resultados possamos comparar os resultados e propor em quais aspectos cada plataforma se mostra mais adequada.

Este trabalho está dividido em sete capítulos, que serão detalhados a seguir.

No Capítulo 2 serão citados as plataformas de distribuição mais usadas para sistemas distribuídos as principais características dos sistemas computacionais distribuídos e uma comparação mais detalhada sobre as plataformas .NET e CORBA.

O Capítulo 3 mostra algumas características dos ambientes virtuais distribuídos AVDs.

No Capítulo 4 serão abordadas as ferramentas de apoio utilizadas na construção deste protótipo bem como a arquitetura utilizada.

O Capítulo 5 apresenta alguns detalhes da implementação do sistema com partes de código fonte e a interface do sistema.

No Capítulo 6 será apresentado detalhes dos testes e a análise dos resultados obtidos entre os protótipos em .NET e em CORBA.

Finalmente, o Capítulo 7 apresenta as conclusões e os trabalhos futuros.

CAPÍTULO II

2. SISTEMAS DISTRIBUÍDOS

2.1. Introdução

Este capítulo apresenta as principais características dos sistemas computacionais distribuídos; apresenta de forma detalhada a arquitetura CORBA a plataforma .NET, e uma comparação entre a plataforma .NET com a arquitetura CORBA.

2.2. O que é distribuição?

Segundo TANENBAUM [1995, p. 261]: “Sistema Distribuído é aquele que roda em um conjunto de máquinas sem memória compartilhada, máquinas estas que mesmo assim parecem ser um único sistema coerente”.

Em outras palavras, sistema distribuído é um sistema de software instalado em uma rede, que dá ao sistema alto grau de coesão e transparência.

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, 1994].

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.

2.3 Vantagens e Desvantagens do Sistema Distribuído (SD)

Assim como toda tecnologia que surge, um SD também tem suas vantagens e desvantagens. Acredita-se que os sistemas distribuídos têm a oferecer muito mais em vantagens do que desvantagens, principalmente por vir com a promessa de solucionar grandes problemas, como por exemplo, o alto custo da implantação de sistemas centralizados, dificuldade em desenvolver para ambientes centrais. Entretanto, essas arquiteturas têm a oferecer algumas desvantagens, que hoje ainda podem ser encaradas como empecilho para uma migração para as mesmas. Abaixo, serão descritos alguns pontos dessas vantagens e desvantagens.

2.3.1 Vantagens em relação ao Sistema Central

- **A questão econômica** – é possível apresentar inúmeras vantagens, mas entre as que se destacam está primeiramente a questão econômica.
- **A confiabilidade** – Em geral, os sistemas distribuídos já trazem certa estabilidade e confiabilidade apenas pela sua forma de funcionamento.
- **A velocidade de processamento** - além da questão econômica e da confiabilidade, tem-se também a vantagem na questão da velocidade do sistema. Um sistema distribuído pode atingir velocidade de processamento muito superior a de qualquer sistema centralizado existente, acarretando assim um excelente desempenho do mesmo a um baixo custo, permitindo a execução separada de várias aplicações diferentes ao mesmo tempo.
- **Expansão da capacidade** - uma questão também muito importante diz respeito à melhoria de capacidade: em um ambiente centralizado, com a utilização de um computador de alto desempenho, se for necessário aumentar

a capacidade de processamento, seria preciso a compra de um novo computador de maior capacidade, o que faz voltar à questão econômica, pois isso ocasionaria um custo muito elevado.

Na Tabela 1 estão relacionadas as principais vantagens do sistema distribuído em relação ao sistema central.

Tabela 1. Vantagens do Sistema Distribuído

Economia	Os micros oferecem melhor relação custo/desempenho em relação aos sistemas centralizados.
Velocidade	Um sistema distribuído pode chegar a velocidades que sistemas centralizados existentes não podem.
Distribuição Inerente	As aplicações podem envolver máquinas separadas fisicamente.
Confiabilidade	Se uma máquina der problema, o sistema apesar de perder um pouco do desempenho, continua no ar.
Crescimento	Para o crescimento não se precisa fazer grandes investimentos financeiros

Acredita-se que só esses pontos acima citados já seriam motivos suficientes para uma total migração para ambientes distribuídos, entretanto, existem alguns pontos, que hoje ainda atrapalham na disseminação.

2.3.2 Desvantagens dos Sistemas Distribuídos

Um problema que os ambientes distribuídos sofrem, refere-se às redes: problemas de conexão, tráfego, entre outros, podem comprometer muito o funcionamento de um sistema distribuído, portanto, é fundamental uma rede bem projetada.

O que poderia ser também um grande problema, já foi superado, que é o entendimento funcional do sistema distribuído.

Mas os problemas citados acima têm os seus dias contados, tendo em vista que assim como surgem novas tecnologias para o desenvolvimento de softwares, surgem também novas

formas de transmissão de dados bem mais rápidas, seguras e com maior capacidade do que as atuais. Na Tabela 2 são mostradas alguma desvantagens dos sistemas distribuídos.

Tabela 2. Desvantagens dos Sistemas Distribuídos.

Software	Pouco conhecimento sobre software de sistemas distribuídos pelos usuários
Rede	A rede pode saturar ou cometer erros
Segurança	Acesso fácil pode se aplicar também a dados secretos

2.4 Middleware

O *Middleware* é a designação genérica utilizada para referir os sistemas de software que se executam entre as aplicações e os sistemas operativos. O objetivo do *Middleware* é facilitar o desenvolvimento de aplicações, tipicamente aplicações distribuídas, assim como facilitar a integração de sistemas legados ou desenvolvidos de forma não integrada.

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

- Plataformas de hardware;
- Sistemas operacionais;
- Bibliotecas de comunicação;
- Protocolos de comunicação;
- Formatação de dados;
- Linguagens de programação;
- Modelos de programação.

Suas características são:

O *Middleware* tem que estar disponível em diversas máquinas. As transferências têm que ser confiáveis, ou seja, tem que 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 com a utilização de endereços *Multicast*, por exemplo.

2.5. Plataformas de distribuição de software

Os mais populares paradigmas de objetos distribuídos são: *Distributed Object Component Model* – DCOM da Microsoft, *Common Object Request Broker Architecture* – CORBA da OMG e Java/Remote Method Invocation - Java/RMI da JavaSoft [RAJ, 2004]. Porém o *framework* de objetos distribuídos da Microsoft, desde 2000, é a plataforma .NET [HUANG, 2000]. A seguir, é apresentada uma visão das principais características destes paradigmas.

2.5.1 CORBA

2.5.1.1 Introdução

O padrão CORBA é um modelo proposto pela OMG (*Object Management Group*), objetivando a promover na teoria e na prática a tecnologia de objetos de forma distribuída, ou seja, é uma estrutura comum para o desenvolvimento independente de aplicações, usando técnicas de orientação a objeto em redes de computadores heterogêneas. Visa diminuir consideravelmente os custos, reduzir a complexidade, e proporcionar caminhos para o surgimento de novas aplicações a partir dos conceitos propostos pela OMG. O CORBA, resumidamente, propõe a interoperabilidade local ou remota entre aplicações, independente das linguagens de programação em que foram desenvolvidas e sobre quais plataformas serão executadas.

A OMG é uma organização formada por empresas dos diferentes ramos da informática (venda, desenvolvimento e produção). Inicialmente, somavam-se treze empresas,

hoje já são setecentos e cinquenta, entre elas estão: DEC, Cannon, IBM, Sun, Apple, e outras. Essas empresas trabalham sem fins lucrativos, para promover a criação e elaboração de modelos e padrões que proporcionem a interoperabilidade entre aplicações que usam tecnologia orientada a objeto. Dessa maneira nasceu a tecnologia CORBA (*Common Object Request Architecture Broker*).

A primeira versão do CORBA, a 1.1, surgiu em 1991, momento este, onde se definiu a IDL (*Interface Definition Language*) e a API (*Application Programming Interfaces*). Porém, a interoperabilidade entre os objetos desenvolvido em linguagens de diferentes fabricantes, só veio em 1994, com a segunda versão do CORBA, a 2.0, quando se implementou no mesmo o IOP (*Inter-ORB Protocol*).

2.5.1.2 Modelo de Objetos da OMG

Os objetos distribuídos CORBA são pequenas partes inteligentes de códigos de um sistema maior. Essas pequenas partes estão presentes em algum lugar da rede, e apresentam-se como componentes binários que podem ser acessados por meio de métodos de invocação, os objetos clientes podem ser remotos ou não.

Para criar um objeto, utiliza-se uma linguagem de alto nível para escrever o código do objeto. Pode-se usar C++, Java, Smaltalk e Ada. O objeto é compilado utilizando uma linguagem definida na especificação do CORBA (IDL).

Para a ocorrência das conhecidas características dos objetos distribuídos, o CORBA dispõe de um meio que padroniza os objetos, permitindo sua comunicação: a utilização de interfaces. Cada objeto tem uma interface definida, a qual deve conhecer e saber requisitar um serviço a outro objeto, dessa maneira, não se faz necessário saber detalhes de sua implementação (código fonte) para que o objeto possa ser acessado. Para especificar essa

interface dos objetos, a OMG utiliza a uma IDL, como meio adotado para definir um padrão entre os objetos.

Com a definição das interfaces, o objeto cliente acessa um outro objeto pela execução de uma requisição (evento) ao mesmo, o qual tem suas características conhecidas por todos através de sua interface, que por sua vez é conhecida por ser gerada a partir do padrão IDL, utilizado por todos os objetos da OMG, em virtude disso, se faz possível à interação entre diferentes objetos.

A Figura 1 demonstra melhor a Arquitetura do Modelo de Referência da OMG.

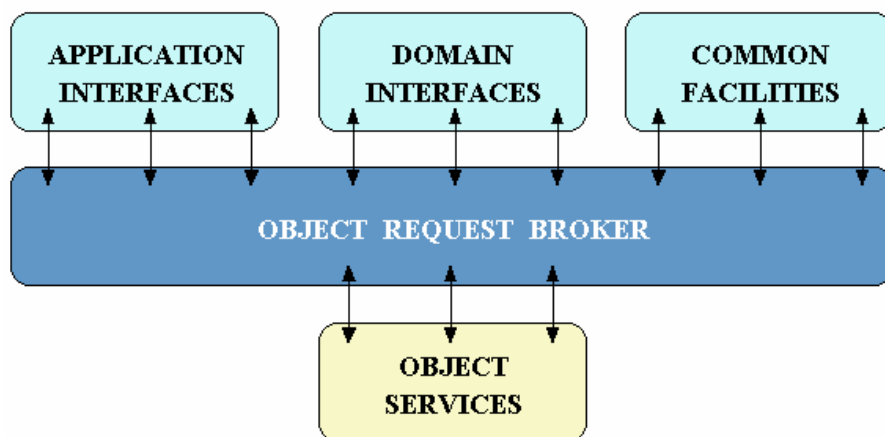


Figura 1: Arquitetura do Modelo de Referência da OMG. [OMG, 2004]

2.5.1.3 Objetos de Serviços

Esse tipo de objeto, nada mais é do que uma coleção de serviços (interfaces) que se apresentam independentes de qualquer domínio de aplicação e que abrangem funções básicas que poderão ser usadas para a implementação de um objeto. Na verdade, os objetos de serviços podem ser vistos como uma adição, um complemento das funções exercidas pelo ORB (*Object Request Broker* – responsável pela comunicação entre o cliente e a implementação do objeto). Existem dezesseis diferentes tipos de objetos de serviço, descritos assim:

- **Gerência do Ciclo de Vida:** define as quatro operações básicas de um objeto: criar, copiar, mover e apagar;
- **Serviços de Persistência:** oferecem uma simples interface de armazenamento de componentes existentes numa variedade de servidores de armazenamento (ex: ODBMSs, RDBMSs e arquivos simples);
- **Serviço de Nome:** permite que os clientes encontrem os objetos baseados nos seus nomes;
- **Serviço de Eventos:** permite o registro e o cancelamento do registro de componentes de acordo com a especificação do evento, é um serviço assíncrono.
- **Serviço de Controle de Concorrência:** permite a criação de um "gerente" capaz de controlar determinadas vantagens para uma determinada execução em depreciação de outra;
- **Serviço de Negociação:** permite que os clientes encontrem os objetos baseados em suas propriedades;
- **Serviços de Relação:** permite que possam ser criadas dinamicamente relações (associações) entre componentes;
- **Serviço de Externalização:** oferece um padrão para que possa adquirir dados externos e internos de um componente através do mecanismo de *stream-like*;
- **Serviços de Pesquisa:** permite busca de operações por objetos;
- **Serviço de Licença:** permite uma "licença", um direito de uso de uma determinada operação sobre um componente, desde que seja justificável a utilização dessa operação sobre tal componente;

- **Serviço de Propriedade:** permite operações de comparações de nomes, valores ou propriedades entre componentes;
- **Serviço de Segurança:** disponibiliza de autenticação, autorização criptografia entre outros serviços de segurança;
- **Serviço de Tempo;**
- **Serviço de "Comércio" (*Trader*);**
- **Serviço de Mudança de Administração;**
- **Serviço de Coleções;**

Uma das grandes vantagens dos objetos de serviços, é que eles permitem que os desenvolvedores não precisem se preocupar com os serviços no nível de sistemas, e sim apenas com seus objetos.

2.5.1.4 Facilidades Comuns

Assim como os objetos de serviços, as "facilidades comuns" também se apresentam como uma coleção de serviços (interfaces), os quais podem ser compartilhados horizontalmente no nível de usuário final, diferentemente dos objetos de serviços que possuem seus serviços direcionados ao sistema (ORB). Outra diferença quanto aos objetos de serviços, é que os serviços oferecidos por essa interface (facilidades comuns) não são considerados fundamentais. Existem duas categorias de Facilidades Comuns: horizontal e vertical.

➤ Horizontal

- Interface usuário: permite editar serviços semelhantes aos promovidos pelo *OpenDoc* e *OLE*;

- Administração da Informação: permite a combinação de documentos armazenados e o intercâmbio de dados;
- Administração do Sistema: permite definir interfaces para administrar, instruir, configurar, instalar, operar e reparar os componentes de objetos distribuídos;
- Administração de Tarefas: permite *workflow*, longas transações, scripts, *rules*, e e-mail.

➤ Vertical

Está relacionada à criação e definição de interfaces IDL vinculadas aos segmentos do mercado vertical, em outras palavras, voltada a questões como saúde, vendas, finanças.

Como exemplo de uma facilidade pode-se citar o DDCF (*Distributed Document Component Facility*), trata-se de uma facilidade comum de documento composto baseado no padrão *OpenDoc*, o DDCF permite a apresentação e troca de objetos baseados em modelos de documentos, facilitando assim a ligação de um objeto planilha em um relatório.

2.5.1.5 Interfaces de Domínio

Compreendem premissas parecidas com as dos objetos de serviços e das facilidades comuns, mas estas interfaces estão orientadas para o domínio de uma aplicação específica.

2.5.1.6 Interface de Aplicação

Interfaces desenvolvidas especificamente para uma determinada aplicação. O que juntamente ao fato da OMG não produzir aplicações, faz com que não haja uma padronização para essas interfaces.

2.5.1.7 Objetos de Aplicação

Trata-se de componentes desenvolvidos especificamente para aplicações do usuário final. Esses objetos correspondem às noções tradicionais das aplicações. Dentro deles, temos os objetos de negócio, que correspondem aos objetos que formam as aplicações compatíveis com o CORBA. A partir de um conjunto de objetos de negócio se é possível construir uma aplicação. Outro ponto importante, é que ele (Objeto de Negócio) manipula processos de negócios que estão presentes no mundo real. Devido a todas essas características, os objetos de aplicação não são padronizados pela OMG.

Visando o gerenciamento da complexidade da aplicação, os objetos de negócio se concentram mais nas interfaces e suas implementações.

2.5.2 Componentes da Arquitetura CORBA

A arquitetura CORBA é um conjunto complexo de partes que possuem suas particularidades e especificações, que têm como objetivo comum, promover um ambiente que atenda às especificações definidas pela OMG para o CORBA. A Figura 2 mostra a descrição das partes da família CORBA.

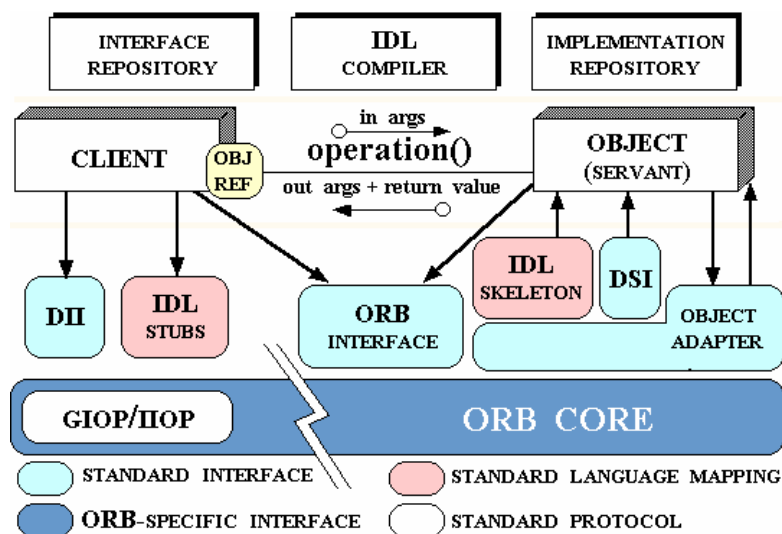


Figura 2. Componentes da arquitetura CORBA. [OMG, 2004]

2.5.2.1 Clientes

Cliente é toda entidade (programa) responsável por invocar uma determinada operação sobre uma implementação de objeto. Como uma das características do CORBA, os detalhes utilizados (como o adaptador de objeto, ou o ORB utilizado) para que seja feito esse acesso aos serviços de um determinado objeto é transparente e dispensável do conhecimento do cliente, precisando o mesmo apenas saber interagir com a interface do tal objeto. Essa invocação feita pelo cliente, que para o mesmo aparenta ser local, poderá ser feita utilizando-se o *stub*, (o já conhecido conjunto de rotinas de bibliotecas que são invocadas como procedimentos e que são geradas no momento da compilação da interface), ou poderá ser feita utilizando-se um conjunto de rotinas de invocações feitas dinamicamente através da interface DII (Interface de Invocação Dinâmica).

2.5.2.2 ORB (*Object Request Broker*)

É responsável pela comunicação, de forma transparente, entre o cliente e a implementação do objeto desejado. O ORB é a principal arma na simplificação da programação em ambientes distribuídos, já que o mesmo libera os programadores de realizarem o árduo trabalho de programar a baixo nível os detalhes das invocações dos métodos. O ORB exerce como principal função, achar a implementação do objeto que atenda as necessidades da invocação de um determinado cliente, após isso, fica responsável pela ativação (caso não esteja ativado) do mesmo, passar as informações necessárias para o objeto e devolver (quando for necessário) para o cliente o resultado desta operação. Relembrando que todos esses passos são feitos de forma transparente para o cliente.

2.5.2.3. Interface ORB

A interface ORB foi definida, dentro da especificação CORBA, como uma "interface abstrata" para o ORB, servindo como uma biblioteca de ajuda para os serviços locais do mesmo (ORB). Onde ainda, por esta interface se tratar de uma entidade lógica, pode ser implementada de diversas maneiras, portanto, a interface ORB vem justamente para desacoplar às aplicações dos detalhes de cada uma dessas implementações de ORB. Dentre as funcionalidades oferecidas por essa interface, temos a conversão de “referências a objetos” para *strings* e vice-versa, provê também a criação de listas de argumentos para as requisições que são realizadas através da DII.

2.5.2.4. Interface de Invocação Dinâmica (DII)

Permite ao cliente, por meio do descobrimento dos métodos em tempo de execução, um acesso direto aos mecanismos de requisição residentes em um ORB e também que realizem uma sincronização chamada não-bloqueada (operações separadas de envio e recebimento) e chamada unidirecionais (só de envio). Neste tipo de invocação, esse processo é feito de forma dinâmica sem necessitar passar pela interface específica da IDL para que seja ativado (*Stub*).

2.5.2.5. Interface de Esqueleto Dinâmico (DSI)

Essa interface realiza uma função semelhante a realizada pela DII, a diferença básica, é que todas as atividades realizadas pela DSI são feitas no servidor. A principal característica da DSI é permitir que ORB transmita, apesar de não conhecer em tempo de execução o tipo de objeto utilizado, requisições para uma implementação de objeto.

2.5.2.6. Stub

Está localizado no cliente, tem como característica promover interfaces estáticas para criar e enviar requisições correspondentes dos serviços desejados do cliente para o servidor. O *Stub* é criado utilizando-se um compilador IDL.

5.2.2.7. Skeleton

Está vinculado diretamente com o *Stub*, e tem como função fornecer a interface através da qual o método recebe as requisições.

2.5.2.8. Adaptador de Objeto

Chamados também de AO, tem como principal função auxiliar o ORB no despacho de requisições para os objetos e na ativação de um determinado objeto. O AO é considerado o núcleo da comunicação do ORB, pois associa implementações de um objeto com o ORB. Podemos encontrar vários tipos de adaptadores de objetos, pois é possível especializá-los para uma determinada função, como exemplo de um AO especializado temos o OODB, que são adaptadores para objetos, adaptadores persistentes e de bibliotecas para objetos não-remotos, entretanto, independentemente de existir os AOs especializados, sempre será encontrado um "padrão" denominado BOA (*Basic Object Adapter*), que nada mais é do que um adaptador padrão presente em todos os ORB's.

2.5.2.9 Produtos ORB

Abaixo estão alguns produtos ORB disponíveis no mercado, como:

- ChorusORB (Sun) ;
- Component Broker/DSOM (IBM);
- ILU (Xerox Parc);

- Netscape Internet Service Broker (Netscape);
- Object Director (Fujitsu);
- omniORB2 (AT&T Laboratories);
- ORB Plus (HP);
- RPC-ORB (Nortel);
- VisiBroker (Borland).

2.5.3 Plataforma .NET e DCOM

2.5.3.1 Introdução a plataforma .NET e DCOM

Antes de falarmos da plataforma .NET, vamos deixar claro que .NET *remoting* e DCOM são apenas semelhantes e não idênticas.

As duas tecnologias são baseadas em diferentes áreas. COM, DCOM e COM+, são baseadas na arquitetura COM (VB6, ATL, C++, MFC), enquanto .NET *Remoting* é baseada no *framework* .NET [ASADUZZAMAN, 2003].

O modelo de objetos distribuídos de propriedade da Microsoft é conhecido como DCOM [ROGERSON, 1997]. Tal arquitetura transporta o modelo de objetos da Microsoft, conhecido como *Common Object Model* (COM), para o ambiente distribuído. Assim como CORBA, a plataforma DCOM permite que programas desenvolvidos em diferentes linguagens possam se comunicar. Com este fim, objetos distribuídos devem implementar uma ou mais interfaces que encapsulam tabelas de ponteiros para as entidades que realmente implementam os métodos remotos. Em DCOM, cada interface possui um identificador único, que também é usado para identificar o componente que a implementa.

A comunicação entre objetos, em DCOM, é feita via representantes locais, ou *proxies*, assim como em CORBA. Porém, a nomenclatura utilizada em DCOM não é a mesma

que a utilizada em CORBA: os representantes de objetos remotos do lado cliente de uma aplicação são chamados *proxies* e os representantes do lado servidor são chamados *stubs*.

Tais componentes trocam mensagens por meio de um mecanismo de RPC implementado em sistemas operacionais da família Windows. Assim como CORBA, DCOM permite que chamadas remotas sejam executadas estática e dinamicamente. Chamadas estáticas são realizadas por meio de *proxies*, os quais são gerados automaticamente. Chamadas dinâmicas, por sua vez, são implementadas via bibliotecas de tipos que permitem a descoberta, durante a execução da aplicação, da assinatura dos métodos que fazem parte de uma determinada interface.

A portabilidade da arquitetura DCOM é penalizada pelo fato de vários dos serviços que ela fornece serem parte do sistema operacional Windows. A Microsoft tem procurado argumentar que esta plataforma é independente de qualquer sistema operacional, contudo, implementações robustas de DCOM existem somente para Windows [COSTA, 2002].

O modelo DCOM serviu de base para o desenvolvimento da tecnologia .NET [OBERMEYER et al. (2001)], que procura integrar os serviços tradicionalmente fornecidos pelos sistemas operacionais da Microsoft, com aplicações implementadas em um vasto conjunto de linguagens de programação diferentes.

Além de integrar programas implementados em diferentes linguagens, .NET também facilita o acesso de aplicações à Internet. Com este objetivo, a plataforma .NET utiliza um protocolo de comunicação baseado na linguagem de marcação XML [MACDONALD, 2003]. Tal protocolo, denominado SOAP [SNELL, 2001], ou *Simple Object Access Protocol*, permite que os parâmetros e o resultado de uma chamada remota sejam conduzidos pela Internet por meio de mensagens codificadas em XML [MACDONALD, 2003].

A plataforma .NET permite a interação entre programas escritos em linguagens diferentes por meio de um ambiente de execução conhecido como *Common Language Runtime*, ou CLR. Tal arquitetura define um sistema de tipos normalizado, para o qual são fornecidos mapeamentos para diversas linguagens de programação. Dado um componente de software qualquer, este pode ser traduzido para CLR, desde que exista um mapeamento entre a linguagem na qual tal componente foi implementado e o sistema de tipos mencionado [COSTA, 2002].

Nem toda linguagem de programação possui um mapeamento perfeito para o sistema de tipos de CLR. Assim, o mapeamento de alguns conceitos de linguagens não orientadas por objetos não é uma questão trivial. Suponha, por exemplo, que os métodos de uma interface remota sejam mapeados para programas separados implementados por alguma linguagem de script, como *Perl*. Neste caso, não é simples garantir que todos os métodos compartilhem um estado comum, como seria possível caso eles fossem implementados pela mesma classe C++. Devido a este fato, nem todos os programas desenvolvidos sobre um sistema operacional da família Windows podem interagir via a plataforma .NET. Outra questão que compromete a portabilidade desta plataforma é o fato dela ser extremamente dependente de sistemas operacionais da Microsoft.

Existem tentativas de desenvolver versões do .NET para outros sistemas operacionais, como por exemplo, o projeto Mono [SCHONIG, 2003], que visa criar uma instância aberta do .NET para o sistema *Linux*. Tal projeto, contudo, ainda encontra-se em estágio pouco desenvolvido.

2.5.3.2 Estrutura do framework .NET

O *framework* .NET está dividido em duas grandes camadas : a *Common Language Runtime* (CLR) e o *.NET Framework Class Library* (Figura 3) . A CLR é responsável por

tarefas como o código em execução, o gerenciamento de memória e *threads*. Já o *.NET Framework Class library* é um rico conjunto de classes orientadas a objeto que permite o desenvolvimento de uma vasta gama de aplicações desde console e GUI até aplicações Web através do uso de ASP.NET e *Web Services*. Esta estrutura abre possibilidade de se conseguir um código portátil entre plataformas já que bastaria reprogramar a CLR para uma outra plataforma e executar o mesmo código *framework*.

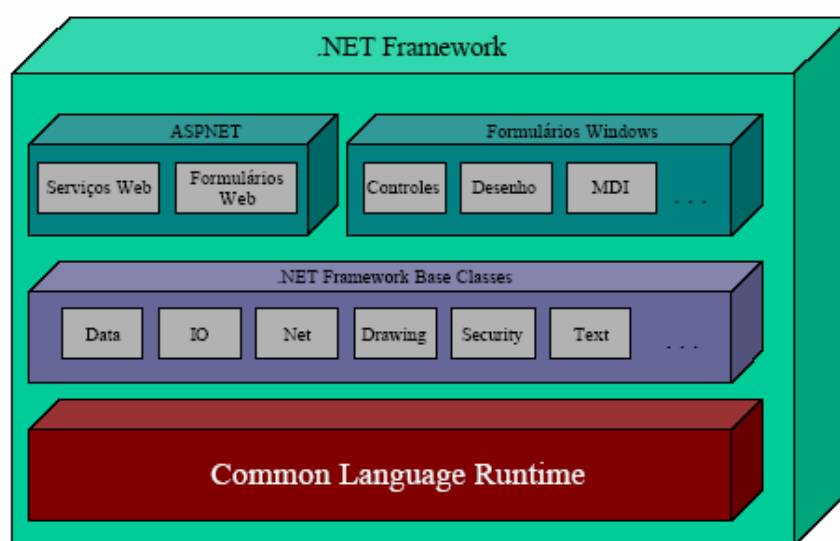


Figura 3. Estrutura do framework .NET.

2.5.3.3 CLR – *Common Language Runtime*

O *runtime* da *.NET Framework* é responsável pelo gerenciamento do código executado e também disponibiliza serviços básicos que facilitam a integração e o desenvolvimento entre linguagens distintas. O código gerado por um compilador para ser executado pela CLR é chamado de código gerenciável.

Segundo a Microsoft este código gerenciável executa debaixo de um “contrato de cooperação” com o *runtime*. Os compiladores devem disponibilizar uma *metadata* que servirá de repositório de informações para que o CLR possa garantir serviços como gerenciamento de memória, integração entre linguagens e controle do tempo de vida dos objetos.

O *runtime* com o conhecimento da *metadata* controla o uso das classes e objetos do código. Ele implementa um sistema de controle de memória chamado *garbage collection* que monitora o uso da memória quando se cria um objeto. Caso o código não implemente todos os procedimentos de limpeza necessários antes de finalizar um objeto este mecanismo de limpeza se encarregará de finalizar todo o processo liberando a memória utilizada.

Uma característica muito importante do CLR é a definição de um sistema comum de tipos. Desta maneira todas as linguagens que geram código para o CLR utilizam a mesma definição de tipos de dados possibilitando a execução de códigos entre linguagens. Como exemplo se poderia escrever uma classe em uma determinada linguagem e utilizá-la em outra. Até mesmo uma instância de uma classe poderia ser passada para um método de uma outra classe escrita em uma linguagem diferente. É fácil notar que será natural o trabalho entre equipes mesmo que usem linguagens de desenvolvimento diferentes. Este fato é um avanço importante sobre a tecnologia COM que mesmo focado na componentização e reutilização de código nem sempre possibilitava uma coerência quando se trabalhava com linguagens de programação diferentes.

A *metadata* contém informações como dependência de módulos, e uso de recursos necessários para a execução de um componente. Esta rica informação permite assegurar que o correto ambiente de execução está ou não disponível na máquina que executará o código.

Poderia fazer uma analogia da CLR com a máquina virtual Java (Virtual Machine). Em teoria a CLR pode ser portada para outras plataformas e o código seria executado sem mudanças. Contudo ela é mais abstrata e oferece suporte para outras linguagens e a convivência entre elas através de um bem cuidado suporte a tipos como foi explicado anteriormente.

2.5.3.4 .NET Framework Class Library

A *.NET Framework Class Library* é um grande conjunto de tipos de dados, interfaces e classes reutilizáveis que possibilitam o acesso a todas as funcionalidades do sistema onde as aplicações e componentes são construídos. Disponibiliza tipos de dados de onde nosso código pode derivar. Além disso, implementa um conjunto de interfaces onde podemos desenvolver a nossa própria coleção de classes.

Desta maneira conseguimos que o código gerado por terceiros e por nós mesmos estejam fortemente ligado ao *Framework* e com isso se melhora a compatibilidade entre terceiros e linguagens de desenvolvimento.

Sobre a *.NET Framework Class Library* podemos desenvolver :

- Serviços Windows 2000 e NT;
- Aplicações GUI (*Windows Forms*);
- Aplicações de console;
- Aplicações ASP.NET (*Web Forms*);
- *Web Services*.

2.6 Comparação entre .NET e CORBA

As arquiteturas, .NET (mais especificamente *.NET Remoting*) e CORBA são padrões de *frameworks* para computação distribuída baseada em objetos. Ambas compartilham algumas metas em comum, como:

- Possibilitam a interoperabilidade entre aplicações escritas em linguagens distintas;

- Permitem modificações na implementação de objetos escritos em linguagens particulares sem a necessidade de alterar os outros objetos que foram escritos em linguagens diferentes.

A Tabela 3 mostra algumas características entre .NET e CORBA.

Tabela 3. Comparação entre .NET e CORBA. [IONA, 2004].

.NET	CORBA
.NET <i>metadata</i> e Microsoft <i>Intermediate Language</i> (MSIL) prove uma definição independente de linguagens à interface de objetos .NET.	A Linguagem de definição de Interface (IDL) prove uma definição independente de linguagens à interface de objetos CORBA.
.NET <i>metadata</i> é armazenado em um .NET <i>assembly</i> .	IDL é armazenada em um repositório de Interface.
<i>Common Type System</i> especifica tipos mapeados para várias linguagens.	Padroniza a IDL como mapeamento para várias linguagens
CLR (<i>Common Language Runtime</i>) usa <i>metadata</i> para requisições entre aplicações distribuídas.	O <i>Runtime ORB (Object Request Broker)</i> usa a IDL para requisições entre aplicações distribuídas.
.NET <i>Remoting</i> usam canais de comunicação como, por exemplo, protocolo binário da Microsoft sobre o TCP.	Usa protocolos padrão, como por exemplo, IIOP (<i>Internet Inter-ORB Protocol</i>) sobre o TCP/IP.

2.7 Benefícios da plataforma .NET

A plataforma .NET, foi desenvolvida para prover soluções para problemas comuns enfrentados por desenvolvedores de software.

A Tabela 4 mostra como a plataforma .NET resolve alguns problemas relacionados ao desenvolvimento de software.

Tabela 4. Benefícios do framework .NET.

Problemas relacionados com o desenvolvimento de software	Solução .NET
Interoperacionalidade, integração e extensibilidade das aplicações são difíceis e dispendiosas.	O uso do XML - um modelo acessível controlado pelo <i>World Wide Web Consortium</i> (W3C) - remove barreiras ao uso de dados e à integração de software.
A competição entre os proprietários das tecnologias de software dificulta a integração.	A Microsoft .NET é construída com base em modelos acessíveis e abarca todas as linguagens de programação.
A utilização pelo o usuário final não é simples nem atrativa. Os usuários finais não conseguem facilmente transmitir dados de uma aplicação para outra nem trabalhar com base nesses dados.	O XML facilita o fluxo de informações e o software da .NET dá aos usuários a possibilidade de trabalharem com os dados recebidos.
Os usuários finais não têm controle sobre suas informações e dados pessoais quando trabalham na Rede, criando problemas de privacidade e segurança que dificultam a recepção de software <i>Web-based</i> .	A Microsoft .NET provê um conjunto de serviços essenciais que permite ao usuário ter controle de suas informações pessoais e bloquear o acesso a elas.

CAPÍTULO III

3. AMBIENTES VIRTUAIS DISTRIBUÍDOS

3.1. Introdução

A Realidade Virtual (RV), como mencionado anteriormente proporciona um nível mais elevado de interface entre homem e máquina, podendo ser caracterizada como uma junção de três idéias básicas: imersão, interação e navegação [MORIE, 1994]. Analisando estas idéias de forma isolada, elas não são de exclusividade de RV, porém, neste caso elas coexistem.

Imersão é a característica que proporciona ao usuário um sentimento de presença dentro de um ambiente sintético, gerado pelo computador, seja através do uso de dispositivos de E/S atípicos, como um capacete de visualização, luvas, ou através de salas especiais de projeção, como a CAVE [CRUZ-NEIRA et al., 1992].

O sistema tem que ser capaz de interagir com o usuário, detectando suas ações e refletindo-as dentro do mundo virtual. A latência do sistema (retardação), isto é, o tempo que decorre entre a ação do usuário e a reflexão desta ação no ambiente virtual, tem que estar dentro de um limite aceitável, pois caso contrário, a sensação de imersão se perderá. Este limite tem que ser menor que 100ms para as aplicações que exigem um sincronismo maior entre os objetos, como aviões em voo sincronizado, e 300ms para as demais [PULLEN, 1995].

Ambientes virtuais distribuídos são sistemas que permitem que vários usuários, fisicamente dispersos, interajam entre si em tempo real.

Os principais requisitos para aplicações de Ambientes Virtuais Distribuídos são listados a seguir [ARAUJO, 1996] [VAN DAN, 1993]:

- Taxas rápidas de atualização do ambiente virtual;
- Latência máxima;
- Tratamento de múltiplos dispositivos de E/S (capacetes, óculos, dispositivos de rastreamento de movimentos, dispositivos de tato de força, e etc.);
- Simulação de um número grande de objetos com comportamento complexo;
- Representação realista do mundo virtual;
- Distribuição dos dados em redes de longa distância;
- Suporte a tipos diferentes de unidades de dados, com requisitos diferentes de confiabilidade na entrega;
- Suporte à comunicação por difusão seletiva de várias fontes para vários destinos;
- Suporte à integração e sincronização de informações multimídia à ambientes virtuais;
- Suporte aos recursos computacionais e redes heterogêneas.

Os problemas são maiores e os requisitos mais rígidos quando o número de usuários participantes é muito grande [PULLEN, 1997] [HOLBROOK, 1995] [ARAUJO, 1997]:

- Gerenciamento de até milhares de usuários de grupos, que possa sustentar centenas de trocas de grupos por segundo através de associação / desassociação de um número grande de participantes e/ou objetos móveis simulados, várias vezes, no decorrer da simulação;

- Suporte à alocação dinâmica de recursos de rede para os grupos de difusão seletiva, através de mecanismos de reserva de recursos;
- Suporte a diferentes níveis de qualidade de serviço, como o número de quadros por segundo da apresentação da simulação, áudio, vídeo, segurança, privacidade, etc.;
- Comunicação por difusão seletiva confiável, que passa a ser um desafio quando as aplicações envolvem um número muito grande de participantes.

Analisando os requisitos citados acima podemos perceber que a complexidade de um Ambiente Virtual Distribuído (AVD) cresce de acordo com sua escalabilidade. Com isso a camada de comunicação e a arquitetura utilizada para distribuição são cruciais para obtermos melhor desempenho em rede.

3.2 Suporte à Comunicação de um AVD

Com o aumento da capacidade de comunicação das redes e de processamento dos computadores possibilita a existência dos AVDs. Os tipos de comunicação que suportam a comunicação entre AVDs são (Figura 4):

- Difusão ponto-a-ponto (*unicast*): requer o estabelecimento de uma conexão ou caminho entre uma estação com todas as outras estações da rede, perfazendo um total de $N(N-1)/2$ conexões virtuais para um determinado grupo, onde N é o número de estações pertencentes ao grupo. Por exemplo, para um grupo de 1000 membros, as 1000 estações teriam que endereçar e enviar, separadamente, 999 pacotes idênticos [MACEDONIA , 1995];
- Difusão (*broadcast*): é um fluxo de dados único dirigido a todas as estações da rede. Este tipo de comunicação requer que todas as estações examinem

todos os pacotes, mesmo se a informação não for endereçada especificamente para aquela estação. Isso acarreta uma queda de desempenho, já que o computador tem que interromper suas operações para executar tarefas de recepção no nível do sistema operacional;

- Difusão seletiva (*multicast*): fornece serviços de entrega de mensagens no esquema de um-para-muitos e muitos-para-muitos, para aplicações nas quais há a necessidade de comunicação entre várias estações simultaneamente. A difusão seletiva proporciona a formação de grupos de tamanhos arbitrários, que se comunicam em rede, através de uma única transmissão feita pelo emissor. Por exemplo, em uma teleconferência a difusão seletiva permite que uma estação envie voz e vídeo simultaneamente para um grupo de estações.

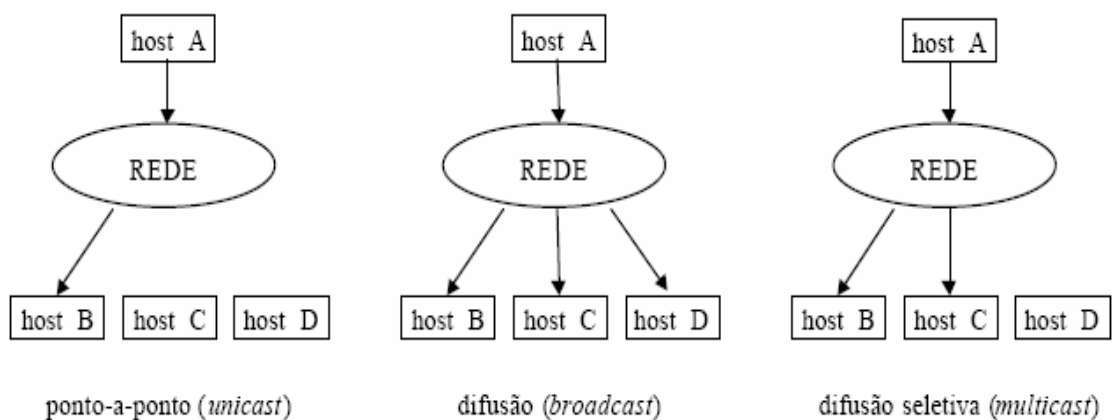


Figura 4. Métodos de Disseminação da Informação [TODESCO, 2000].

O paradigma da comunicação ponto-a-ponto domina os sistemas de comunicação baseados na arquitetura atual da Internet e no modelo de referência da OSI [DIOT, 1997]. Esse tipo de comunicação tem características inadequadas às necessidades das aplicações que usam difusão seletiva. Na comunicação ponto-a-ponto, um participante inicia e controla a transmissão dos dados, enquanto que na comunicação por difusão seletiva qualquer

participante pode decidir associar-se a um grupo ou deixar esse grupo, ou seja, cada participante define os parâmetros de controle de sua sessão de comunicação.

A comunicação por difusão seletiva suporta transferência de informações entre um conjunto de participantes, e está se tornando cada vez mais relevante em ambientes distribuídos, pois para alguns tipos de aplicações distribuídas, como AVDs e aplicações multimídia, é a única maneira de controlar a transmissão de dados para membros de um determinado grupo [ALMEROTH, 1996].

3.3 Arquiteturas de Suporte à AVDs

Os modelos que suportam a comunicação entre AVDs são mostrados na Figura 5.

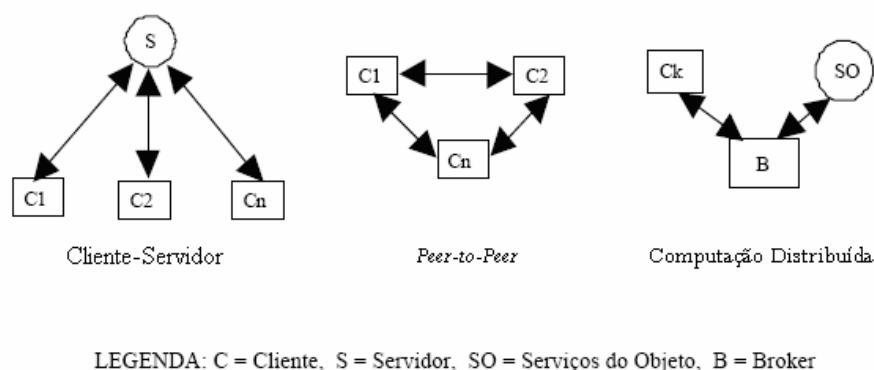


Figura 5. Arquiteturas Distribuídas [TODESCO, 2000].

Algumas características das arquiteturas são listadas abaixo [TODESCO, 2000].

3.3.1 Características do Modelo Cliente-Servidor

Prós:

- Controle Centralizado;
- Baixo grau de conectividade;

- Um diretório com os endereços dos clientes pode ser mantido no servidor, e o cliente só precisa conhecer o endereço do servidor;
- Controle das versões de software é simplificado, já que a cópia principal é mantida no servidor, podendo ser acessado por todos os clientes.

Contras:

- Se o servidor falhar, todo o sistema fica sem operação;
- Como o servidor é o único elo de comunicação entre os clientes, este pode se tornar um gargalo, limitando o crescimento do sistema;
- Não suporta bem a dispersão geográfica, já que um cliente geograficamente distante pode não ter opção de comunicação, que não com aquele servidor;
- Não escala bem, isto é, o servidor suporta um número máximo de clientes, embora existam técnicas para superar esta dificuldade.

3.3.2 Características do Modelo Peer-to-Peer

Prós:

- Falha em um cliente particular afeta apenas a participação do usuário ligado àquele cliente, sem afetar a operação do sistema como um todo;
- Elimina o problema de gargalo distribuindo a coordenação das atividades entre os clientes;
- Dependendo da topologia adotada, as distâncias entre os clientes podem ser reduzidas;
- Escala bem, isto é, pode suportar um número crescente de clientes;

Contras:

- A distribuição da coordenação das atividades, entre os clientes comunicantes, acarreta uma cooperação através de protocolos de comunicação, o que leva a uma maior complexidade de *software*, custos mais altos de desenvolvimento, e menor desempenho em cada cliente;
- Cada cliente deve conhecer os endereços dos outros clientes que possuem células coincidentes com as suas, para que possam se comunicar diretamente;
- Maior complexidade no gerenciamento de entrada/saída de novos clientes;
- O controle de versões de *software* é mais difícil de ser mantido nesta abordagem distribuída.

3.3.3 Características do Modelo de Computação Distribuída Orientado a Objetos (DOCs)

Prós:

- Os objetos são independentes da aplicação, o que permite que sejam mais facilmente modificados e incorporados ao sistema;
- Promove o reuso de código, portabilidade e interoperabilidade entre sistemas heterogêneos;
- Os objetos são auto contidos (possuem comportamento, suportam interação e possuem relacionamentos) e capazes de executar ações, o que, na construção de mundos virtuais, é importante, pois os objetos podem conter conceitos, como, por exemplo, gravidade, já embutidos, sem que estes conceitos precisem ser desenvolvidos na aplicação;
- O conceito de orientação a objeto reduz a quantidade de código que precisa ser escrito de forma geral, através dos conceitos de herança, em que um

objeto pode herdar características de outros objetos, e polimorfismo, que permite que um comando genérico possa ser emitido a diferentes objetos, e cada objeto responde àquele comando de forma dependente de seu tipo;

- Como os objetos do mundo real podem ser mapeados de forma abstrata sobre um objeto na computação orientada a objeto, sistemas complexos, que incluem sistemas de Realidade Virtual, podem ser mais facilmente desenvolvidos;
- Ao solicitar um serviço, o cliente não precisa se preocupar com a localização do provedor de serviços - a interação é feita de forma transparente;

Contras:

- DOCs como o CORBA, apresentam baixo desempenho - a execução da solicitação de um serviço pode levar vários milissegundos o que, somado à latência gerada pelo resto do sistema, pode inviabilizar a interatividade e sentimento de presença, características fundamentais em um sistema de RV.

Os modelos acima apresentam as suas vantagens e desvantagens. Por exemplo: o modelo cliente-servidor apresenta pontos interessantes no suporte às aplicações de jogos distribuídos (controle de cópia ilegal, dificulta as trapaças no jogo, evita ações ilegais, etc.) [GAUTIER, 1997]. Porém para aplicações de AVDs, onde o crescimento do sistema é almejado sem a necessidade de um controle efetivo da conexão dos usuários (requisito importante para aplicações de RV em educação), o modelo *peer-to-peer* é o mais recomendado. Uma arquitetura híbrida, que combina os dois tipos de modelo pode ser uma alternativa interessante. Como esta aplicação foi desenvolvida como ferramenta de auxílio a educação, utilizamos o modelo Cliente-Servidor para prover um maior controle durante a

utilização do mesmo, como por exemplo, quando um aluno desconecta do sistema o professor pode ver e chamar a atenção do aluno que está desatento.

CAPÍTULO IV

4. ARQUITETURA DO SISTEMA

4.1. Introdução

Analizando as soluções propostas pelo *framework* .NET, vide Tabela 4, foi possível perceber que o *framework* .NET seria uma boa escolha para ser utilizada como camada intermediária de comunicação entre ambientes virtuais distintos.

Para o desenvolvimento deste protótipo foram utilizadas algumas ferramentas de apoio que serão discutidas nos próximos itens.

4.2 Tecnologias de apoio

Novas tecnologias têm sido criadas para dar suporte ao desenvolvimento de aplicações em Realidade Virtual (RV). Atualmente, com o avanço dos equipamentos de hardware e o barateamento dos mesmos se torna possível a execução de aplicações de RV em computadores pessoais (PCs), que antigamente se fazia somente em estações de trabalho desenvolvidas para este fim.

Somente a evolução de componentes físicos e custos mais acessíveis não seria uma solução em longo prazo, pois o aumento da capacidade de processamento das máquinas está ligado diretamente ao aumento das necessidades dos usuários. Com isso novas tecnologias são desenvolvidas para prover mais eficiência e eficácia às aplicações de computação gráfica. As tecnologias de apoio utilizadas neste trabalho serão discutidas no próximo item.

4.2.1 OPENGL

OpenGL é uma interface de software para dispositivos de hardware. É uma biblioteca gráfica de modelagem e exibição tridimensional e portátil para vários sistemas operacionais. Seus recursos permitem ao usuário criar objetos gráficos com qualidade próxima à de um raytracer, de modo mais rápido que este último, além de incluir recursos avançados de animação, tratamento de imagens e texturas.

A biblioteca *OpenGL* (*Open Graphics Library*) foi introduzida em 1992 pela *Silicon Graphics* (www.sgi.com), no intuito de conceber uma API (Interface de Programação de Aplicação) gráfica independente de dispositivos de exibição. Com isto, seria estabelecida uma ponte entre o processo de modelagem geométrica de objetos, situadas em um nível de abstração mais elevado, e as rotinas de exibição e de processamento de imagens implementadas em dispositivos (hardware) e sistemas operacionais específicos. As função utilizada pela *OpenGL* para desenhar um ponto na tela, por exemplo, possui os mesmos nome e parâmetros em todos os sistemas operacionais nos quais *OpenGL* foi implementada, e produz o mesmo efeito de exibição em cada um destes sistemas.

Diante das funcionalidades providas pela *OpenGL*, tal biblioteca tem se tornado um padrão amplamente adotado na indústria de desenvolvimento de aplicações. Este fato tem sido encorajado também pela facilidade de aprendizado, pela estabilidade das rotinas, pela boa documentação disponível [NEIDER, 2000] e pelos resultados visuais consistentes para qualquer sistema de exibição concordante com este padrão. É possível encontrar na Internet diversos sites que tratam sobre *OpenGL*; o mais popular é a sua página oficial: www.opengl.org. Neste endereço encontram-se disponíveis diversos artigos e *links* sobre o assunto, fóruns de discussão, informações sobre a evolução desse padrão, além de uma série de programas utilitários.

Diversos jogos, aplicações científicas e comerciais tem utilizado *OpenGL* como ferramenta de apresentação de recursos visuais, principalmente com a adoção deste padrão por parte dos fabricantes de placas de vídeo destinadas aos consumidores domésticos.

Entre os recursos gráficos disponíveis pela *OpenGL*, podem ser destacados os seguintes:

- Modos de desenho de pontos (escrita/xor/transparência);
- Ajuste de largura de linhas;
- Aplicação de transparência;
- Ativação/desativação de *serrilhamento* (*aliasing*);
- Mapeamento de superfícies com textura;
- Seleção de janela de desenho;
- Manipulação de fontes/tipos de iluminação e sombreamento;
- Transformação de sistemas de coordenadas;
- Transformações em perspectiva;
- Combinação de imagens (*blending*).

Diversos desenvolvedores de software possuem suas próprias implementações do *OpenGL*. Entre as implementações mais populares, destacam-se aquelas providas pela Microsoft, a biblioteca compartilhada dinâmica **opengl32.dll**, pela própria SGI, **opengl.dll**, e o pacote Mesa3D (www.mesa3d.org). Esta última é uma implementação *OpenGL conformant*, ou seja, possui funcionalidades semelhantes às do *OpenGL*, além de ser completamente gratuita e funcionar nos sistemas operacionais Windows 9x/NT, MS-DOS e Unix, facilitando ainda mais a portabilidade de aplicações.

As implementações do OpenGL geralmente provêem bibliotecas auxiliares, tais como a GLU (OpenGL Utility library), utilizada para realizar tarefas comuns, tais como manipulação de matrizes, geração de superfícies e construção de objetos por composição.

4.2.2 Visual Studio .NET

Para o desenvolvimento de aplicações .NET não é necessário a utilização do Visual Studio .NET (VS.NET). Para isso, é necessário simplesmente de um editor de texto, como o bloco de notas e um compilador .NET, como os compiladores C#.

Porém, neste trabalho foi utilizado o VS.NET, pois ele nos proporciona um ambiente de desenvolvimento integrado (IDE), onde podemos desenvolver, depurar, testar e disponibilizar aplicações .NET. Temos também o *Intellisense* que completa o código para nós, otimizando o processo de desenvolvimento de software. Outra característica que influenciou na escolha do VS.NET foi a facilidade de utilização da ferramenta, bem como a organização dos arquivos gerados automaticamente.

4.2.3 Linguagem C#

A linguagem C# foi desenvolvida pela Microsoft em conjunto com a plataforma .NET. Embora exista inúmeras linguagens que podemos utilizar com a plataforma .NET, utilizamos neste trabalho a linguagem C# devido a inúmeras vantagens como:

- Foi desenvolvida para funcionar na plataforma .NET, sem preocupações de compatibilidade com código já existente;
- O compilador C# foi o primeiro a ser desenvolvido;
- A maioria das classes do .NET *Framework* e até mesmo o compilador *JScript* foram desenvolvidos em C#;
- Primeira linguagem "orientada a componentes" da família C/C++;

- Segue os padrões de programação orientada a objetos (POO), onde tudo deriva de um ancestral comum, no caso *System.Object*, não existem atributos e métodos *stand-alone*, tudo é declarado dentro do escopo da classe, contudo é possível declarar tipos *struct* e *enum* fora do escopo de classes;
- Fortemente *tipada*, o que ajuda a evitar erros oriundos de uma manipulação imprópria de tipo e/ou atribuições;
- Suporte à COM, COM+ ou outras DLLs escritas por linguagens que geram código não gerenciado;
- É *case-sensitive*, ou seja, faz diferenciação entre maiúsculas e minúsculas, por exemplo, *net* é completamente diferente de *NET*;
- Programas implementados em C# rodam sob um ambiente gerenciável, o que significa que todo o controle de memória é feito pelo *.NET Framework* e não diretamente pelo programador, reduzindo assim falhas na programação enquanto a alocação e liberação de um objeto na memória.

Embora muitas pessoas tenham participado de seu desenvolvimento, é justo dizer que o C# é obra de uma pessoa, o Anders Hejlsberg, hoje um “*Distinguished Engineer*” na Microsoft.

O C# inclui idéias de várias linguagens de programação, mas é patente a influência das duas outras principais linguagens com as quais Anders trabalhou anteriormente: o Pascal do *Delphi* e o Java. Existem também claras influências do C++ e *Smalltalk*.

Por ser uma linguagem padronizada, a linguagem C# não se limita a ter apenas um fornecedor de IDE's (Ambientes de desenvolvimento integrado). Várias empresas já estão distribuindo suas IDE's para C#, algumas de forma comercial, outras como OpenSouce ou simplesmente gratuitas. É possível inclusive desenvolver aplicações em C# apenas usando o

bloco de notas e as ferramentas de linha de comando do Microsoft .NET Framework SDK

v1.1. Algumas das mais conhecidas são:

- Microsoft Visual Studio .NET 2003;
- Borland Delphi 2005;
- ASP .NET WebMatrix;
- #develop (*SharpDevelop*);
- Eclipse for C#.

Com base nas características citadas acima, optamos por utilizar a linguagem C# no desenvolvimento deste protótipo.

4.2.4 Genuine Channels

Genuine Channels pode ser definido como um *Middleware* que combina o desempenho de soluções baseadas em *socket*, a conveniência do *.NET Remoting* e a rica funcionalidade dos *Web Services*[GENUINECHANNELS].

O *Genuine Channels* é uma variante do *.NET Remoting* que possui inúmeros recursos na área de segurança, o mais interessante é que o cliente consegue detectar quando o servidor foi reiniciado e restabelece a comunicação.

Esta ferramenta prove alguns benefícios como:

- Condições para comunicação estável;
- Sistema seguro de canais e eventos de rede;
- Suporte a *IP Multicasting*.

Com base nas características citadas acima, utilizaremos esta ferramenta como *middleware* para distribuição deste protótipo, conseguindo com isso uma comunicação mais estável e maior facilidade de implementação.

4.3 Modelagem UML

Na Figura 6 podemos observar o diagrama de caso de uso do sistema do servidor, onde o usuário é o autor que interage com o sistema para ativar o servidor.

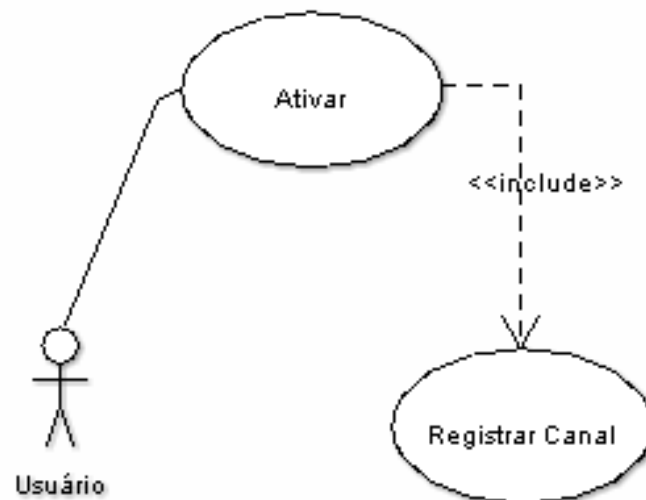


Figura 6. Diagrama de Caso de Uso do Servidor.

Na Figura 7 mostramos o diagrama de caso de uso do mundo da Biologia.

O usuário é o ator, que interage com o sistema chamado de *mundo da biologia*.

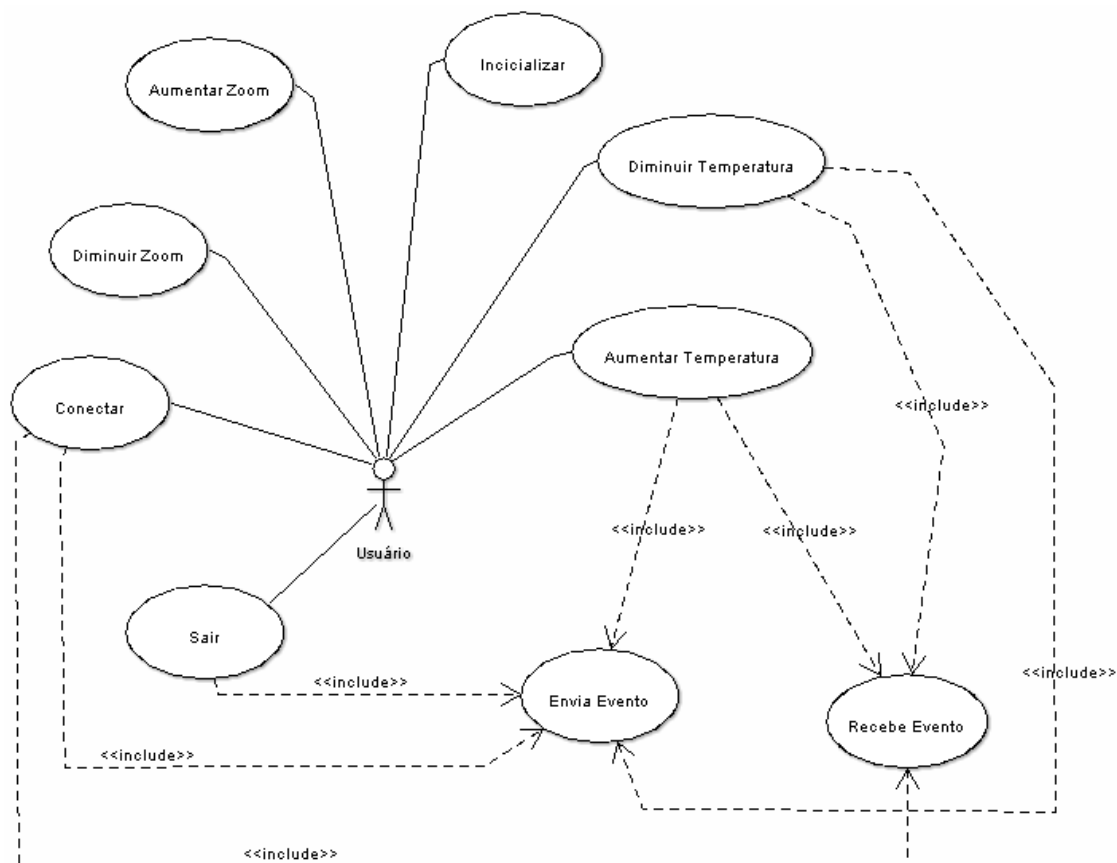


Figura 7. Diagrama de Caso de Uso do Mundo da Biologia.

A Figura 8 mostra o diagrama de caso de uso do mundo da Química. Mais uma vez o usuário do sistema é o ator do diagrama.

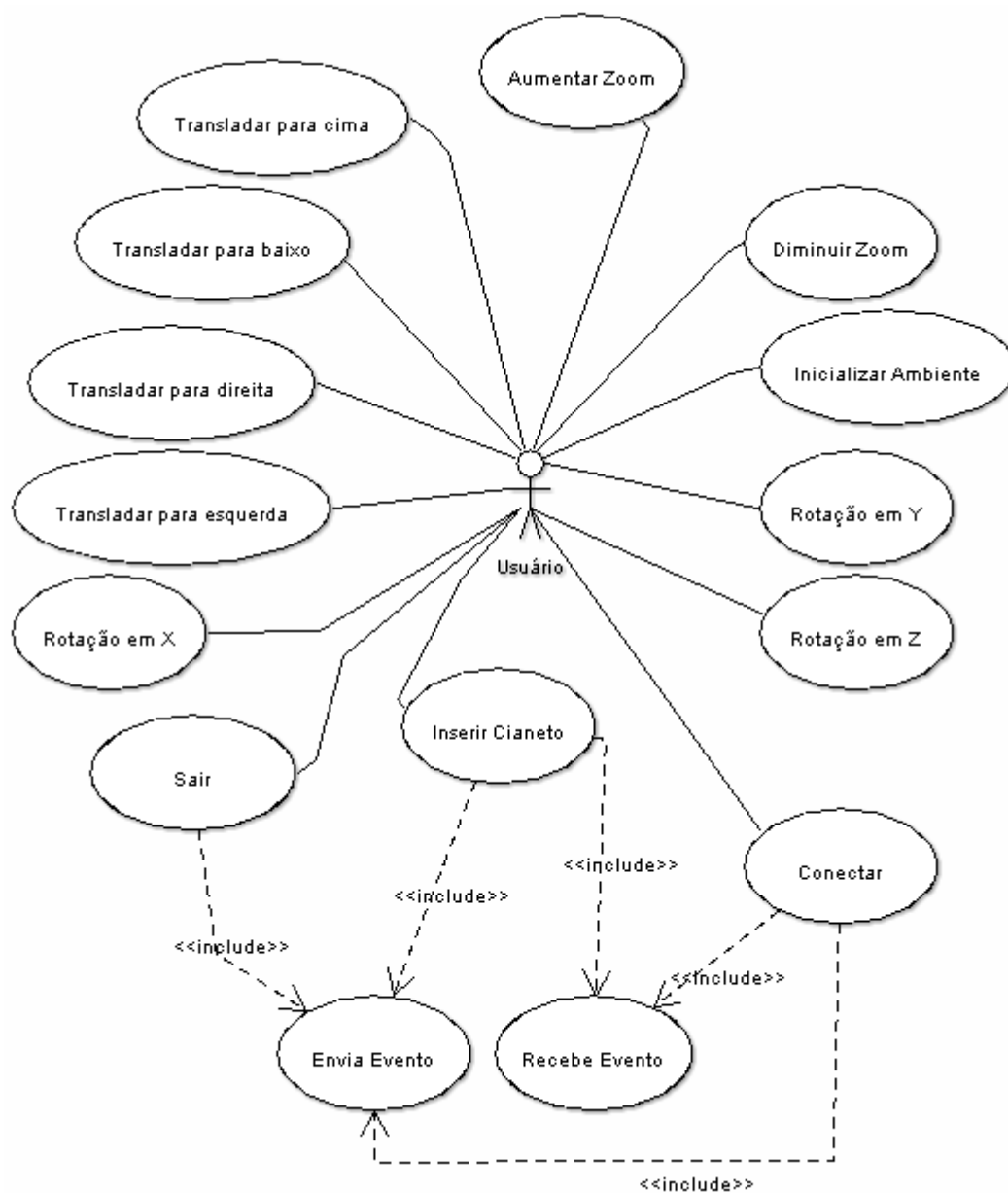


Figura 8. Diagrama de Caso de Uso do Mundo da Química.

A Figura 9 mostra o diagrama de classes do Mundo da Biologia.

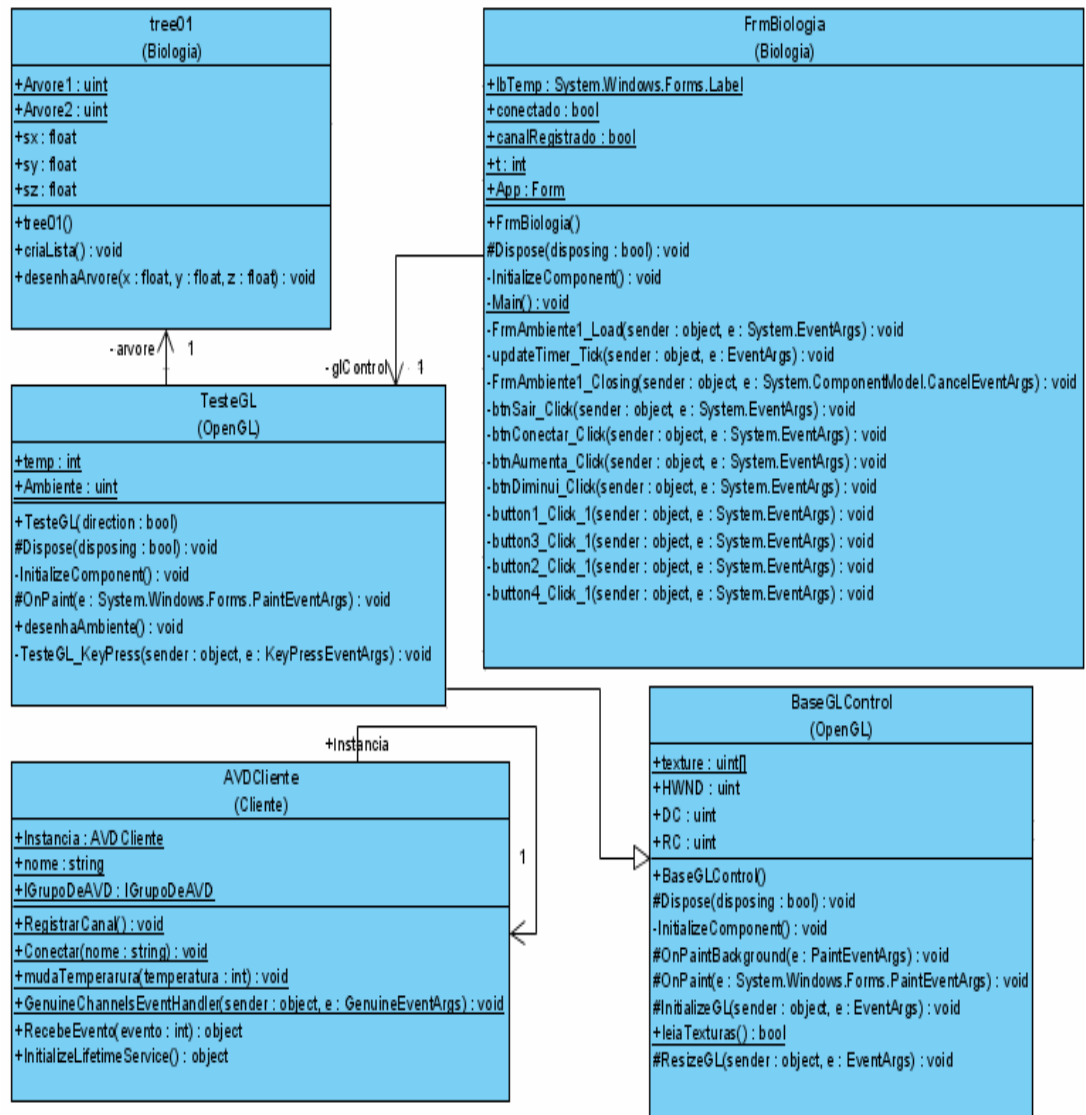


Figura 9. Diagrama de Classes do Mundo da Biologia.

A Figura 10 mostra o diagrama de classes do Mundo da Química.

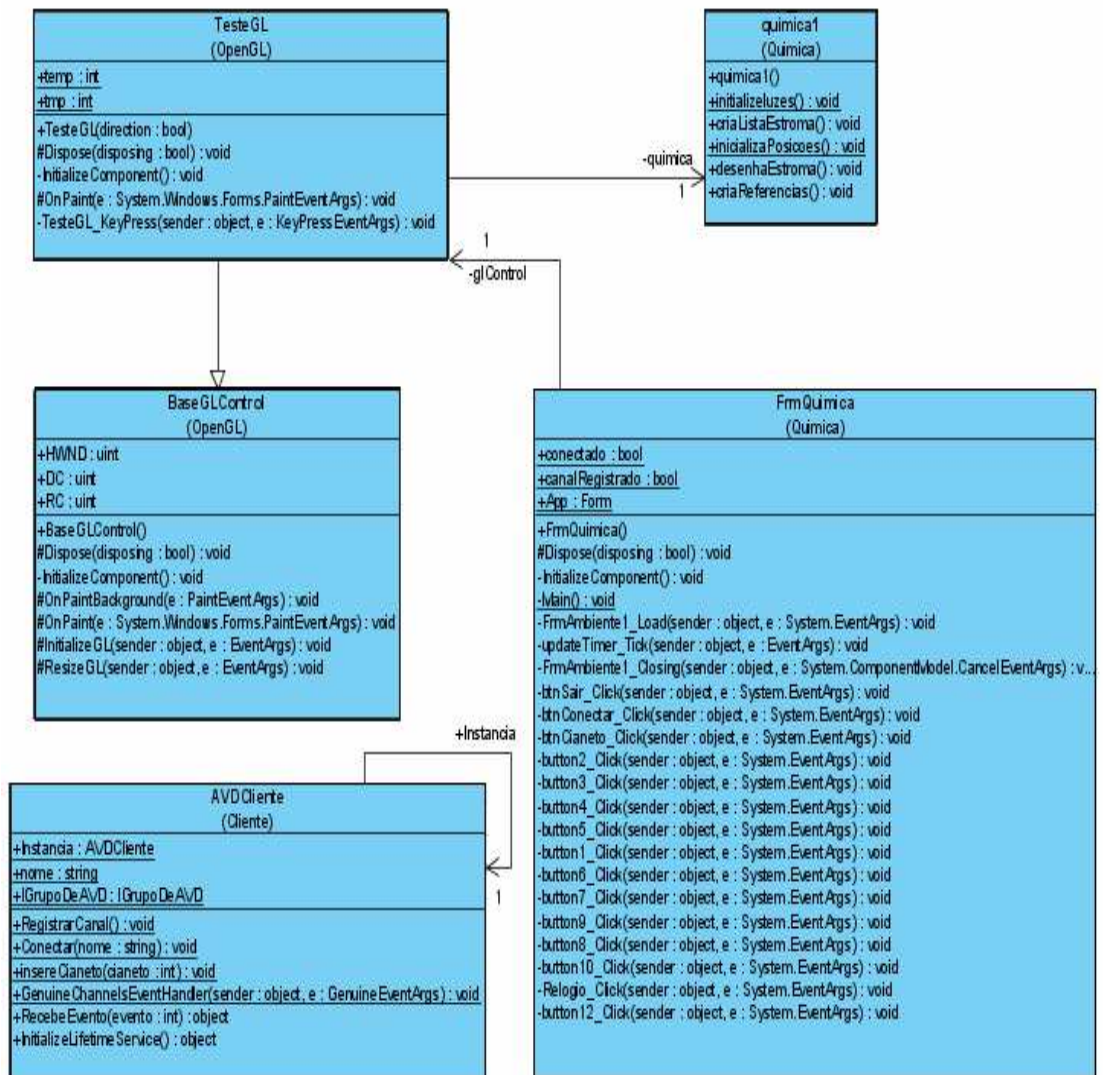


Figura 10. Diagrama de Classes do Mundo da Química.

Na Figura 11 podemos observar o diagrama de classes do Servidor.

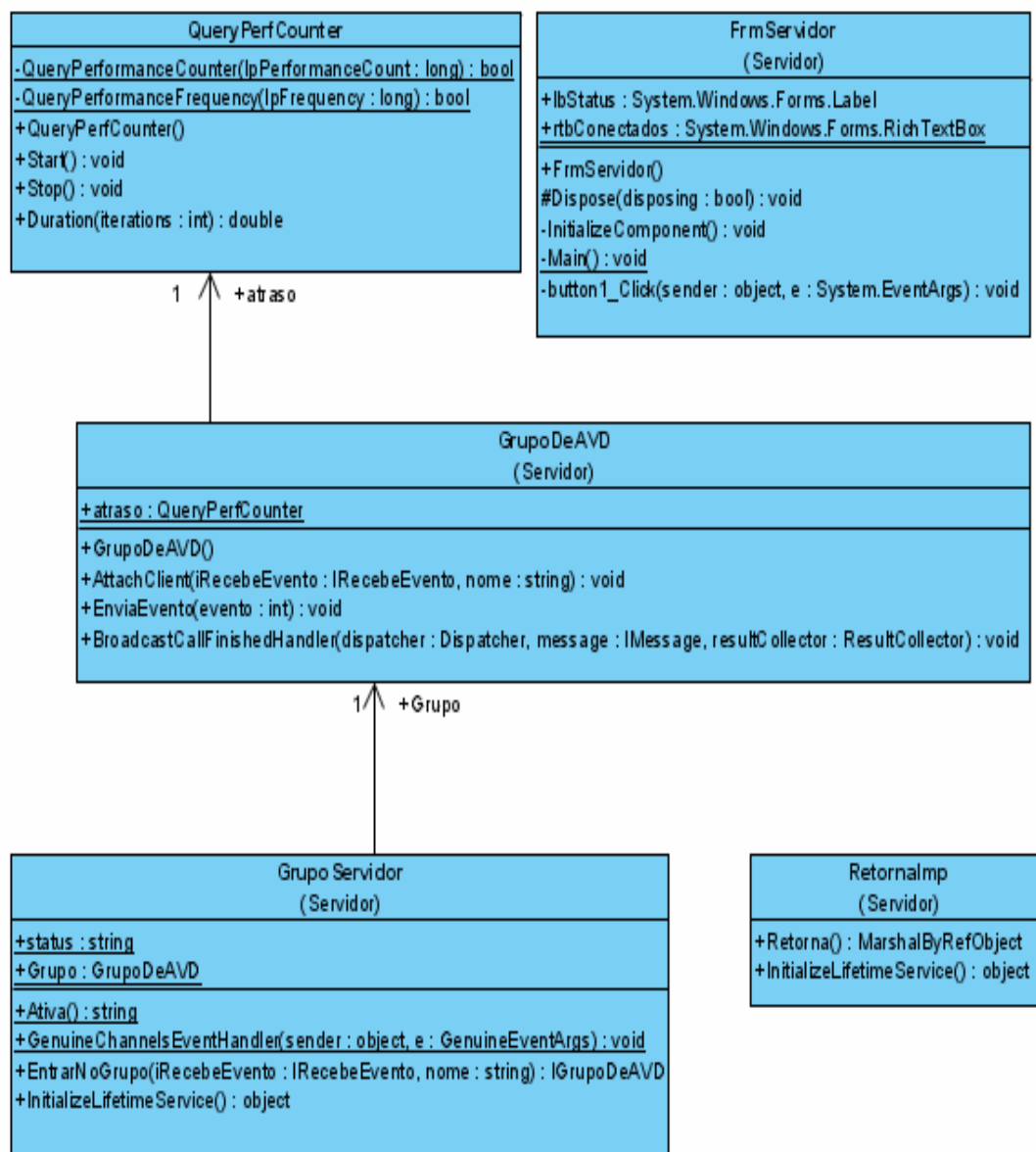


Figura 11. Diagrama de Classes do Servidor.

4.4 Arquitetura Proposta

A arquitetura proposta para o sistema é mostrada na Figura 12.

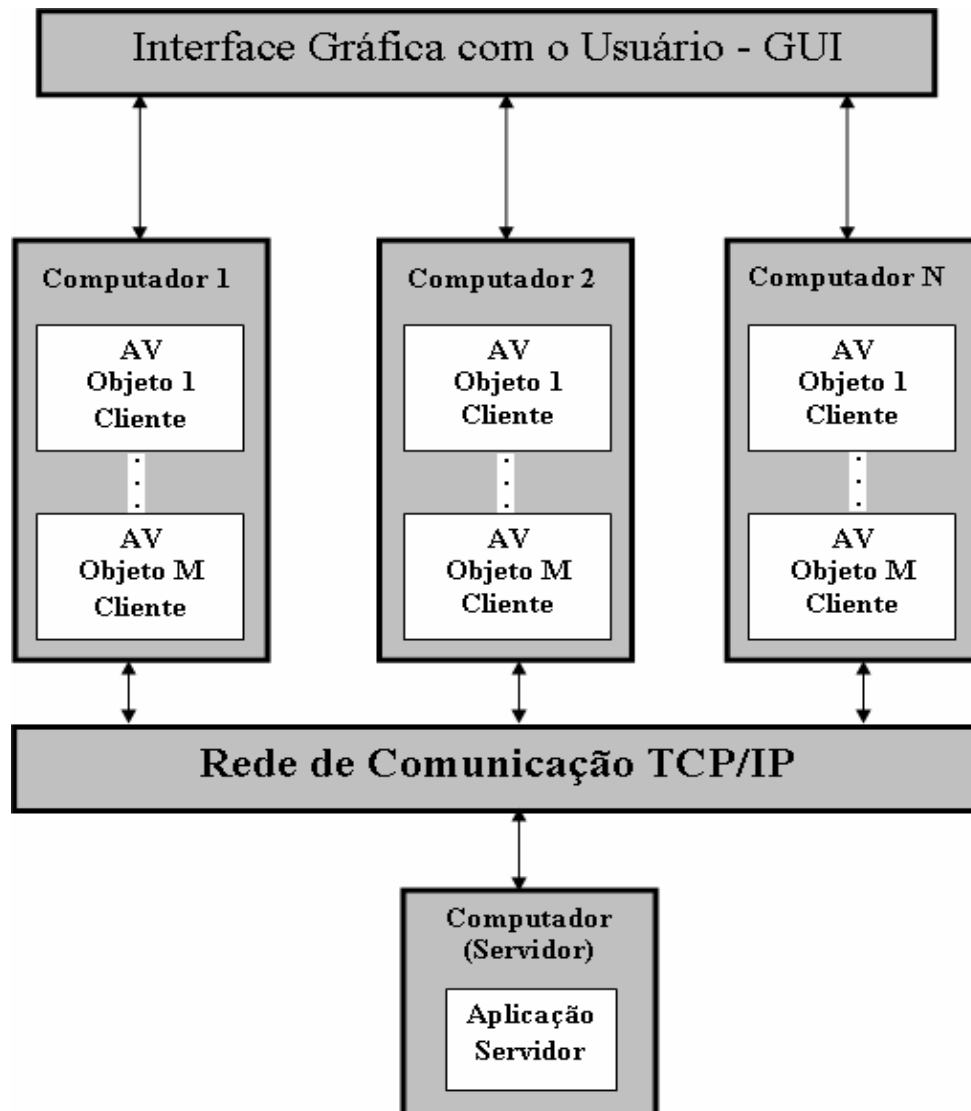


Figura 12. Arquitetura Proposta

A Figura 12 mostra a existência de N computadores, onde cada computador pode hospedar M ambientes virtuais. Todos ambientes virtuais são clientes do servidor. Podemos ter ambientes virtuais distintos, como por exemplo, simular o processo da fotossíntese decomposto em dois mundos distintos como o processo biológico (Ambiente Biologia) e o

processo químico (Ambiente Química). Com isso esta arquitetura pode ser usada para propiciar a multidisciplinaridade.

Cada ambiente virtual pode ser executado separadamente, estando conectado ou não ao servidor. Mas se não estiver conectado ao servidor não enviará eventos em rede para os ambientes que estão conectados.

No caso de mais de dois ambientes conectados os eventos disparados em cada ambiente serão enviados e interpretados de forma distinta em cada um dos ambientes conectados ao servidor. Isto ocorre nos dois sentidos de comunicação.

Para que haja a comunicação entre os ambientes, o servidor tem que estar ativo, e os clientes conectados a ele através de um canal de comunicação.

Os clientes têm que saber onde se encontra o servidor para estabelecer um canal de comunicação entre os ambientes e o servidor, isto é feito, e será mostrado em maiores detalhes no próximo capítulo, com um arquivo de configuração incorporado na aplicação.

Para uma comparação fiel dos resultados foi utilizado neste protótipo, como no trabalho do [RIBEIRO et al, 2005], a API *OPENGL* para modelagem dos ambientes virtuais.

4.4.1 Interface gráfica com o usuário – GUI

Na Figura 13 pode-se observar a interface proposta para o servidor deste protótipo. Mostrando duas pessoas conectadas, sendo Leonardo e Maria.

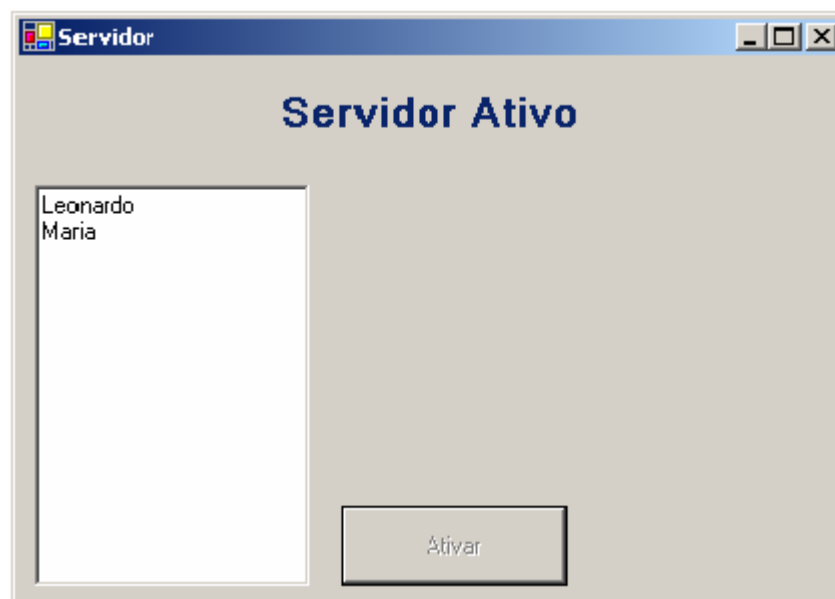


Figura 13. Interface do Servidor

Após clicar em *Ativar*, a mensagem *Servidor ativo* será mostrada na parte superior da janela.

Algumas mensagens de aviso podem ser observadas na Figura 14.

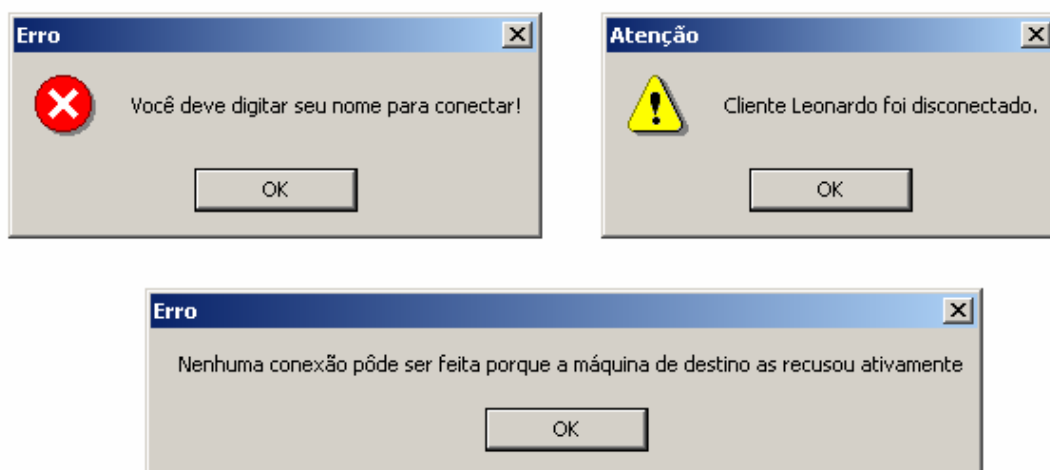


Figura 14. Mensagens de Aviso.

A primeira mensagem é uma mensagem de erro, pois houve uma tentativa de conexão ao servidor sem que um nome para a conexão fosse digitado.

A segunda é disparada na máquina onde está executando o servidor, mostrando que o cliente “Leonardo” foi desconectado. Esta mensagem é interessante, pois se este protótipo for utilizado como ferramenta de auxílio a educação, o professor terá controle dos alunos conectados no sistema. Se um aluno se desconectar por algum motivo, o professor poderá chamar a atenção do mesmo.

A última mensagem é mostrada quando se tenta estabelecer uma conexão com o servidor, mas o servidor não está ativo, ou não foi encontrado.

A Figura 15 mostra a interface criada para o ambiente que simula o processo Biológico da fotossíntese chamado de Mundo da Biologia.

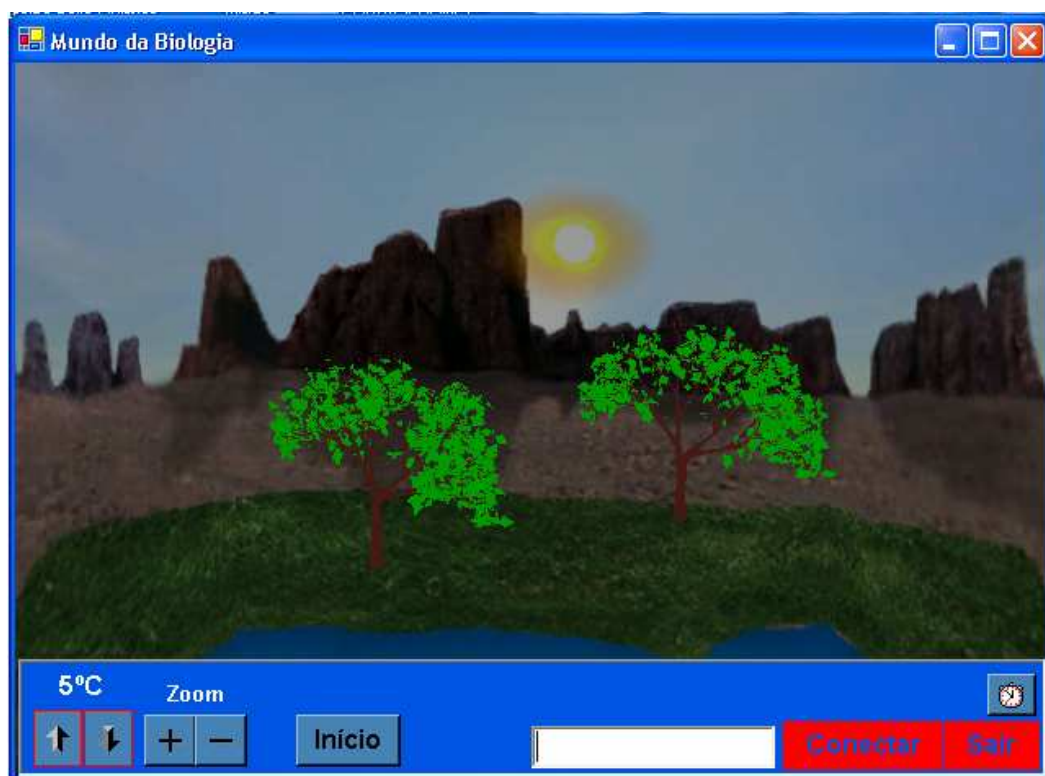


Figura 15. Interface do Mundo da Biologia

Para se conectar é necessário que o servidor esteja ativo, o cliente conheça o identificador do servidor na rede IP (endereço IP do servidor), digitar um nome e depois clicar em conectar. Se tudo proceder corretamente à caixa de texto e o botão conectar irão

desaparecer. Isto quer dizer que o ambiente da Biologia se conectou ao servidor e está pronto para enviar e receber eventos através da rede de comunicação.

A Figura 16 mostra a interface do outro ambiente virtual que simula o processo químico da fotossíntese que chamamos de Mundo da Química.

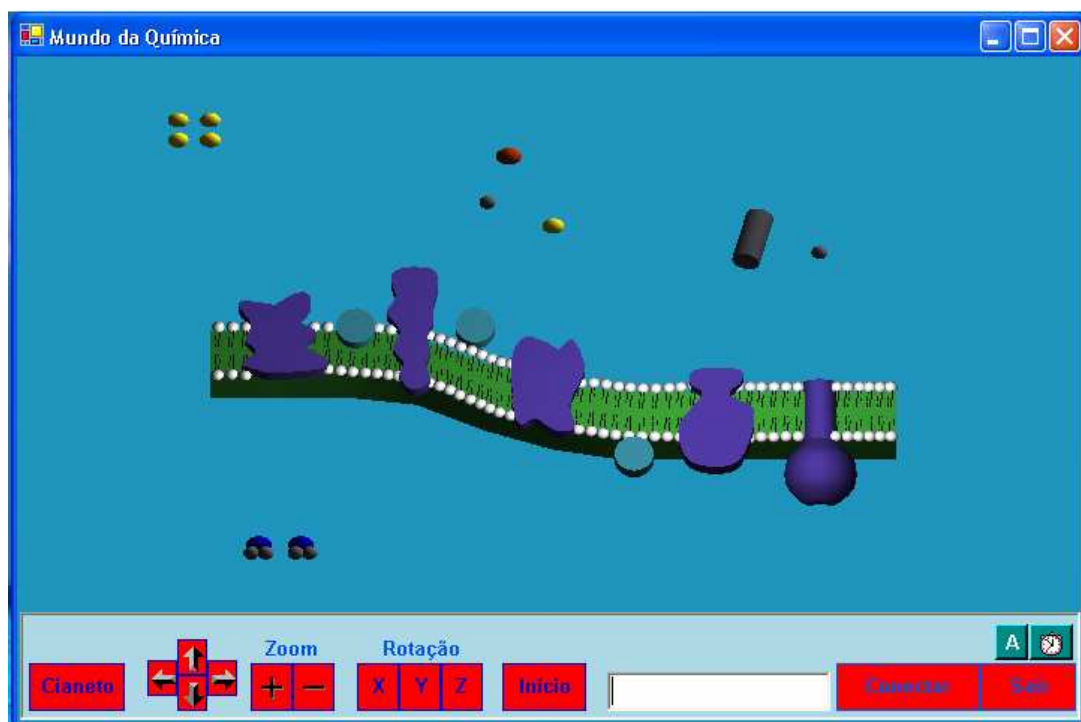


Figura 16. Interface do Mundo da Química

Os procedimentos para conectar são os mesmos citados no mundo da Biologia.

Após o servidor estar ativo e pelo menos dois mundos conectados poderá observar a interação entre eles em rede.

Quando clicamos no botão de aumento de temperatura no mundo da Biologia iremos observar um aumento no tamanho das árvores e um aumento na luminosidade do ambiente, como visto na Figura 17.



Figura 17. Mundo da Biologia com 45°C.

Comparando com a Figura 15 podemos observar o efeito, que será disparado igualmente em todos os ambientes de simulação do processo Biológico da Fotossíntese que estão conectados ao servidor.

Ao mesmo tempo em que este evento é disparado nos mundos da Biologia, será iniciado o processo Químico da Fotossíntese em todos os mundos da química conectados no servidor. A Figura 18 mostra o início do processo químico da fotossíntese.

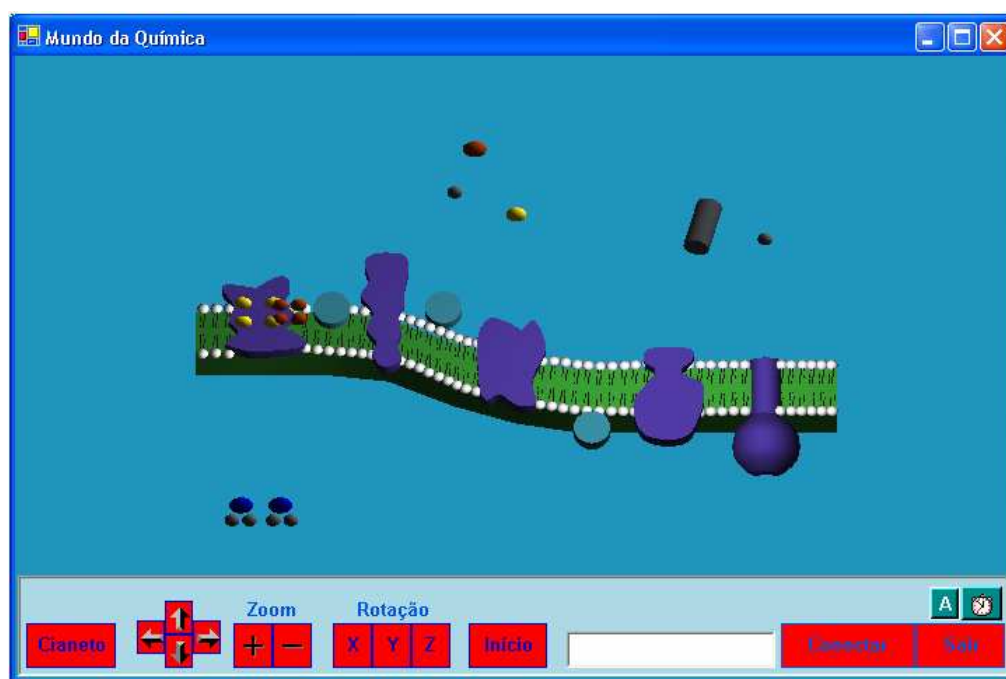


Figura 18. Início do Processo Químico da Fotossíntese.

Nas Figuras 17 e 18 podemos observar o efeito gerado pelo aumento da temperatura no mundo da Biologia e no mundo da Química. Porém existe também o caminho contrário, onde uma ação no mundo da química gera um efeito nos mundos da química e também nos mundos da biologia. Este efeito pode ser visto com a inserção da molécula de cianeto no processo químico da fotossíntese, que interrompe o processo químico e mata as árvores.

Na Figura 19, pode-se observar a molécula de cianeto sendo incorporada no processo químico da fotossíntese.

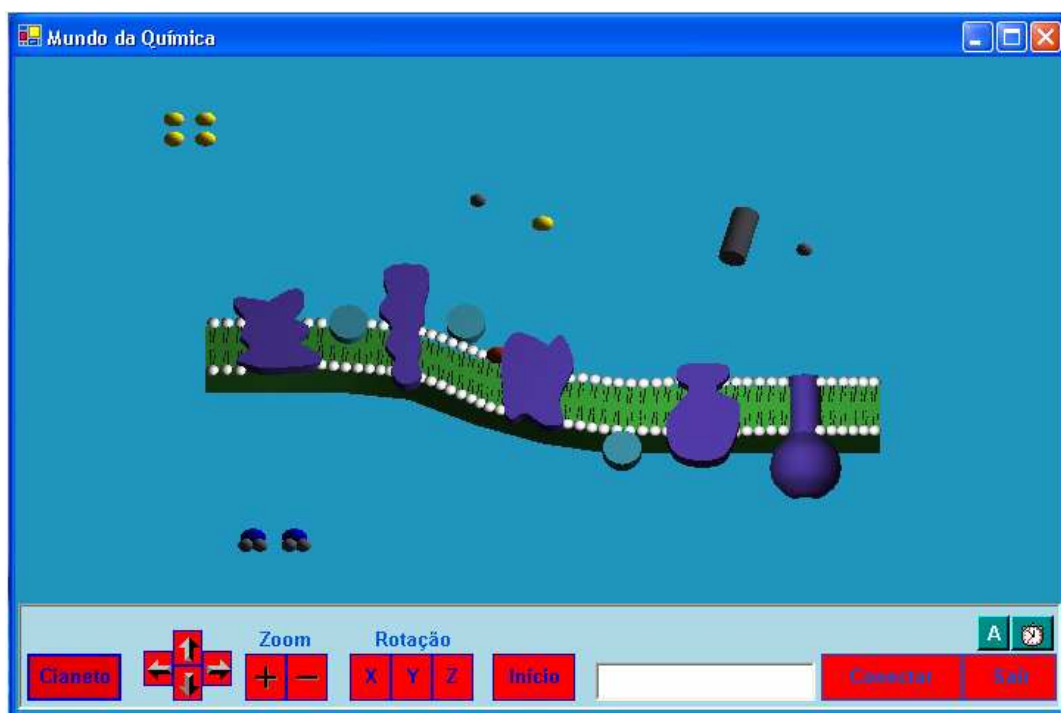


Figura 19. Inserção do Cianeto no Mundo da Química.

Ao mesmo tempo em que se insere cianeto no processo químico podemos observar que as árvores do mundo da Biologia perdem as folhas, simulando assim a morte das mesmas.

A Figura 20 mostra esta reação.

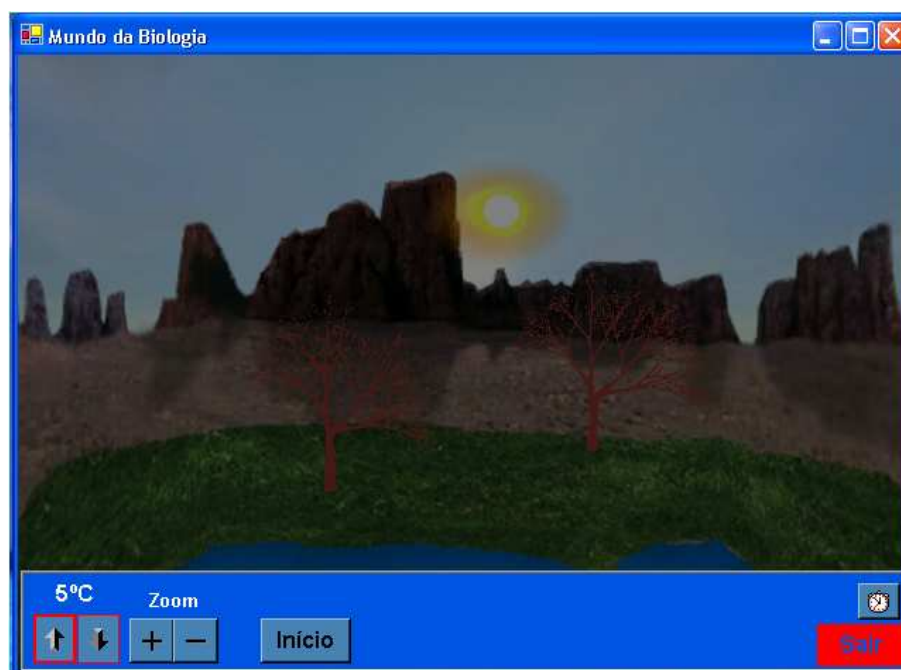


Figura 20. Simulação da Morte das Árvores.

CAPÍTULO V

5. DETALHES DA IMPLEMENTAÇÃO

5.1. Introdução

Este capítulo apresenta detalhes da implementação deste sistema que foi dividido em três etapas, sendo:

- Incorporação da biblioteca gráfica *OPENGL* em aplicações .NET;
- Desenvolvimento da modelagem;
- Desenvolvimento do mecanismo de comunicação utilizando a biblioteca *Genuine Channels*.

Para distribuição dos ambientes foi utilizada a arquitetura de rede Cliente/Servidor.

5.1.1. Incorporação da Biblioteca OPENGL em C#

Utilizar a biblioteca *OPENGL* em aplicações .NET não é tão simples como utilizá-la em linguagens como *Delphi*, C++, Java, etc. Onde simplesmente você utiliza uma diretiva como *#include opengl* em C++ e pronto, todas funcionalidades da biblioteca estão disponíveis para você utilizar em suas aplicações.

Para utilizar esta biblioteca em C# existem vários *wrappers* (empacotador) para incorporar esta biblioteca em aplicações .NET, como:

- TAO disponível em: <http://www.taoframework.com/>;
- CSGl disponível em: <http://csgl.sourceforge.net/>

Para esta aplicação foi escolhido o empacotador do Colin Fahey [COLIN], devido a sua simplicidade e eficiência.

5.2 Detalhes da Modelagem

É importante deixar claro que a modelagem destes experimentos foi à mesma utilizada por Ribeiro (2005), para que não houvesse discrepância nos resultados obtidos, tendo em vista que o esforço computacional de renderização interfere na coleta do *clock* da máquina.

Na Figura 21 podemos verificar parte do código para modelagem do ambiente da Biologia que foi feito com um cubo de seis faces e texturas renderizadas em cada face, dando a idéia de um ambiente.

```
...
GL.glBegin(GL.GL_QUADS);
public void desenhaAmbiente()
{
    GL.glNewList(Ambiente, GL.GL_COMPILE);
    GL.glBindTexture(GL.GL_TEXTURE_2D, OpenGL.BaseGLControl.texture[0]);
    GL.glBegin(GL.GL_QUADS);
        GL.glTexCoord2f(0.0f, 0.0f); GL.glVertex3f(-1.01f, -1.01f, 1.0f);
        GL.glTexCoord2f(1.0f, 0.0f); GL.glVertex3f( 1.005f, -1.005f, 1.0f);
        GL.glTexCoord2f(1.0f, 1.0f); GL.glVertex3f( 1.005f, 1.005f, 1.0f);
        GL.glTexCoord2f(0.0f, 1.0f); GL.glVertex3f(-1.005f, 1.005f, 1.0f);
    GL glEnd();

    GL.glBindTexture(GL.GL_TEXTURE_2D, OpenGL.BaseGLControl.texture[1]);
    GL.glBegin(GL.GL_QUADS);
        GL.glNormal3f( 0.0f, 0.0f, -1.0f);
        GL.glTexCoord2f(1.0f, 0.0f); GL.glVertex3f(-1.005f, -1.005f, -1.0f);
        GL.glTexCoord2f(1.0f, 1.0f); GL.glVertex3f(-1.005f, 1.005f, -1.0f);
        GL.glTexCoord2f(0.0f, 1.0f); GL.glVertex3f( 1.005f, 1.005f, -1.0f);
        GL.glTexCoord2f(0.0f, 0.0f); GL.glVertex3f( 1.005f, -1.005f, -1.0f);
    GL glEnd();

    GL.glBindTexture(GL.GL_TEXTURE_2D, OpenGL.BaseGLControl.texture[2]);
    GL.glBegin(GL.GL_QUADS);
        GL.glNormal3f( 0.0f, 1.0f, 0.0f);
        GL.glTexCoord2f(0.0f, 0.0f); GL.glVertex3f(-1.005f, 1.0f, -1.005f);
        GL.glTexCoord2f(1.0f, 0.0f); GL.glVertex3f(-1.005f, 1.0f, 1.005f);
        GL.glTexCoord2f(1.0f, 1.0f); GL.glVertex3f( 1.005f, 1.0f, 1.005f);
        GL.glTexCoord2f(0.0f, 1.0f); GL.glVertex3f( 1.005f, 1.0f, -1.005f);
    GL glEnd();
}
```

```

GL.glBindTexture(GL.GL_TEXTURE_2D, OpenGL.BaseGLControl.texture[3]);
GL.glBegin(GL.GL_QUADS);
GL.glNormal3f( 1.0f, 0.0f, 0.0f);
GL.glTexCoord2f(1.0f, 0.0f); GL.glVertex3f( 1.0f, -1.005f, -1.005f);
GL.glTexCoord2f(1.0f, 1.0f); GL.glVertex3f( 1.0f, 1.005f, -1.005f);
GL.glTexCoord2f(0.0f, 1.0f); GL.glVertex3f( 1.0f, 1.005f, 1.005f);
GL.glTexCoord2f(0.0f, 0.0f); GL.glVertex3f( 1.0f, -1.005f, 1.005f);
GL.glEnd();

GL.glBindTexture(GL.GL_TEXTURE_2D, OpenGL.BaseGLControl.texture[4]);
GL.glBegin(GL.GL_QUADS);
GL.glNormal3f(-1.0f, 0.0f, 0.0f);
GL.glTexCoord2f(0.0f, 0.0f); GL.glVertex3f(-1.0f, -1.005f, -1.005f);
GL.glTexCoord2f(1.0f, 0.0f); GL.glVertex3f(-1.0f, -1.005f, 1.005f);
GL.glTexCoord2f(1.0f, 1.0f); GL.glVertex3f(-1.0f, 1.005f, 1.005f);
GL.glTexCoord2f(0.0f, 1.0f); GL.glVertex3f(-1.0f, 1.005f, -1.005f);
GL.glEnd();

GL.glBindTexture(GL.GL_TEXTURE_2D, OpenGL.BaseGLControl.texture[5]);
GL.glBegin(GL.GL_QUADS);
GL.glNormal3f( 0.0f, -1.0f, 0.0f);
GL.glTexCoord2f(0.0f, 1.0f); GL.glVertex3f(-1.005f, -1.0f, -1.005f);
GL.glTexCoord2f(0.0f, 0.0f); GL.glVertex3f( 1.005f, -1.0f, -1.005f);
GL.glTexCoord2f(1.0f, 0.0f); GL.glVertex3f( 1.005f, -1.0f, 1.005f);
GL.glTexCoord2f(1.0f, 1.0f); GL.glVertex3f(-1.005f, -1.0f, 1.005f);
GL.glEnd();
GL.glEndList();
...

```

Figura 21. Parte do Código da Modelagem do Mundo Biologia

Para modelagem deste cubo (Mundo da Biologia) foi utilizado a constante *GL_QUADS*, que desenha um quadrado com base em quatro vértices, com isso desenhamos todas as faces do cubo com base nos vértices de cada lado.

Na Figura 22 podemos verificar parte do código da modelagem das árvores do mundo da Biologia.

```

...
GL.glNewList(Arvore1, GL.GL_COMPILE);
GL.glBegin(GL.GL_TRIANGLES);
for (int j=0; j<19834; j++)
{
//desenha tronco
GL.glNormal3f(normals2_0[face_indicies2_0[j,3],0],
normals2_0[face_indicies2_0[j,3],1], normals2_0[face_indicies2_0[j,3],2]);

GL.glVertex3f(vertices2_0[face_indicies2_0[j,0],0],
vertices2_0[face_indicies2_0[j,0],1],
vertices2_0[face_indicies2_0[j,0],2]);

GL.glNormal3f(normals2_0[face_indicies2_0[j,4],0],
normals2_0[face_indicies2_0[j,4],1], normals2_0[face_indicies2_0[j,4],2]);

```

```

GL.glVertex3f(vertices2_0[face_indicies2_0[j,1],0],
vertices2_0[face_indicies2_0[j,1],1],
vertices2_0[face_indicies2_0[j,1],2]);

GL.glNormal3f(normals2_0[face_indicies2_0[j,5],0],
normals2_0[face_indicies2_0[j,5],1], normals2_0[face_indicies2_0[j,5],2]);

GL.glVertex3f(vertices2_0[face_indicies2_0[j,2],0],
vertices2_0[face_indicies2_0[j,2],1],
vertices2_0[face_indicies2_0[j,2],2]);
}
GL.glEnd();
GL.glEndList();

GL.glNewList(Arvore2, GL.GL_COMPILE);
GL.glBegin(GL.GL_TRIANGLES);
//desenha folhas
for (int j=0; j<3197; j++)
{
GL.glNormal3f(normals2_1[face_indicies2_1[j,3],0],
normals2_1[face_indicies2_1[j,3],1], normals2_1[face_indicies2_1[j,3],2]);

GL.glVertex3f(vertices2_1[face_indicies2_1[j,0],0],
vertices2_1[face_indicies2_1[j,0],1],
vertices2_1[face_indicies2_1[j,0],2]);

GL.glNormal3f(normals2_1[face_indicies2_1[j,4],0],
normals2_1[face_indicies2_1[j,4],1], normals2_1[face_indicies2_1[j,4],2]);

GL.glVertex3f(vertices2_1[face_indicies2_1[j,1],0],
vertices2_1[face_indicies2_1[j,1],1],
vertices2_1[face_indicies2_1[j,1],2]);

GL.glNormal3f(normals2_1[face_indicies2_1[j,5],0],
normals2_1[face_indicies2_1[j,5],1], normals2_1[face_indicies2_1[j,5],2]);

GL.glVertex3f(vertices2_1[face_indicies2_1[j,2],0],
vertices2_1[face_indicies2_1[j,2],1],
vertices2_1[face_indicies2_1[j,2],2]);
}
GL.glEnd();
GL.glEndList();
...

```

Figura 22. Parte do Código da Modelagem das Árvores.

Para modelagem deste ambiente foi utilizado a constante *GL_TRIANGLES*, onde especificamos três vértices para cada face.

Esta modelagem está dividida em duas partes, sendo a primeira a construção do Tronco e a segunda a construção das Folhas.

As árvores foram construídas em 3D MAX e depois convertidas em código OPENGL. Este código fornece as matrizes utilizadas para construção da árvore, que como podemos observar na Figura 22 estão sendo utilizadas.

A Figura 23 mostra parte do código para geração do Mundo da Química, que foi gerado também em 3D MAX e convertido para OPENGL.

```
GL.glNewList(Estroma, GL.GL_COMPILE);
GL.glBegin(GL.GL_TRIANGLES);
for (int i=1; i<242; i++)
{
    vert = vertices[i];
    norm = normals[i];
    fac = faces[i];
    qtdfaces = maxFace[i];
    for (int j=0; j<qtdfaces; j++)
    {
        GL.glMaterialfv(GL.GL_FRONT_AND_BACK, GL.GL_AMBIENT_AND_DIFFUSE,
            color[cores[i]]);
        GL.glNormal3f(norm[fac[j,3],0], norm[fac[j,3],1], norm[fac[j,3],2]);

        GL.glVertex3f(vert[fac[j,0],0], vert[fac[j,0],1], vert[fac[j,0],2]);
        GL.glNormal3f(norm[fac[j,4],0], norm[fac[j,4],1], norm[fac[j,4],2]);
        GL.glVertex3f(vert[fac[j,1],0], vert[fac[j,1],1], vert[fac[j,1],2]);
        GL.glNormal3f(norm[fac[j,5],0], norm[fac[j,5],1], norm[fac[j,5],2]);
        GL.glVertex3f(vert[fac[j,2],0], vert[fac[j,2],1], vert[fac[j,2],2]);
    }
}
GL.glEnd();
GL.glEndList();
```

Figura 23. Código para Geração do Estroma.

Para geração do estroma foram utilizadas 723 matrizes de ponto flutuante, obtidas na conversão de 3D MAX em OPENGL. O estroma é construído com 241 objetos, onde cada objeto utiliza três matrizes.

Como foi utilizada apenas uma classe para alocar todas estas matrizes e tendo a necessidade de criar um laço que utilizem todas elas, foi necessário criar outras matrizes de referências e até vetores de matrizes de referências.

Com base nestas mesmas matrizes foram criados os demais objetos do mundo da Química, como moléculas de elétrons, cianeto heterogêneos, oxigênios, ou seja, todas as moléculas envolvidas no processo químico da biologia.

5.3 Distribuição dos ambientes

Para distribuição destes ambientes foi inicialmente desenvolvida uma Interface, que é utilizada pelos clientes e pelo servidor. O nome do arquivo é *Interface.dll* e está *dll* é incorporada tanto no cliente quanto no servidor para prover a comunicação entre eles.

Na Figura 24 podemos ver as interfaces implementadas para esta comunicação.

```
namespace Interface
{
    public interface IRecebeEvento
    {
        object RecebeEvento(int evento);
    }

    public interface IGrupoServidor
    {
        IGrupoDeAVD EntrarNoGrupo(IRecebeEvento iRecebeEvento, string
nome);
    }

    public interface IGrupoDeAVD
    {
        void EnviaEvento(int evento);
    }
}
```

Figura 24.Código da Interface.

Como podemos observar na Figura 24, foi utilizado três interfaces para comunicação do sistema, sendo a interface *IRecebeEvento* implementa um objeto que é utilizado nos clientes para receber um evento do servidor. Repare que utilizamos um *int* para comunicação entre os mundos. Está não é a melhor opção, pois utilizamos 32 *bits* para representar seis informações (5°C, 15°C, 25°C, 35°C, 45°C e Cianeto), com isso estamos utilizando de forma ineficiente a largura de banda da rede. Nós optamos na utilização do *int* para que a comparação de latência entre os sistemas não ficasse incoerente, pois o sistema do Ribeiro (2005) utiliza um tipo de dado da IDL chamado ANSII *char* que tem 32 *bits*.

A interface *IGrupoServidor* é utilizada para os clientes se conectarem no servidor, e a interface *IGrupoDeAVD* utilizada também pelos clientes para enviar eventos para os clientes conectados.

5.3.1 Implementação do Servidor

Como foi mostrado anteriormente na Figura 13, o servidor possui um botão com o nome de *Ativar*. Quando nos clicamos neste botão, estaremos ativando o servidor, ou seja, criaremos um canal de comunicação para os clientes.

A Figura 25 mostra parte do código executado ao clicarmos no botão *Ativar*.

```
...
public static string status;
public static string Ativa()
{
    try
    {
        //configura o servidor
        System.Configuration.ConfigurationSettings.GetConfig("DNS");
        GenuineGlobalEventProvider.GenuineChannelsGlobalEvent += new
        GenuineChannelsGlobalEventHandler(GenuineChannelsEventHandler);
        RemotingConfiguration.Configure("Servidor.exe.config");
        GrupoServidor.Grupo = new GrupoDeAVD();
        // liga o servidor
        RemotingServices.Marshal(new GrupoServidor(), "GrupoServidor.rem");
        RemotingServices.Marshal(new RetornaImp(), "Retorna.rem");
        status = "Servidor Ativo";
    }
    catch(Exception ex)
    {
        status = (ex.Message);
    }
    return status;
}
...
```

Figura 25. Código de Ativação do Servidor.

Este código faz com que o servidor crie um canal de comunicação com os clientes. Os dados de configuração deste canal estão descritos no arquivo de configuração que tem o mesmo nome do executável mais *.config*. A vantagem de se utilizar este tipo de arquivo é que com ele podemos mudar a porta de comunicação utilizada, por exemplo, sem a necessidade de re-compilarmos a aplicação. No final deste código temos o retorno de uma *string* chamada

status, se ela estiver preenchida com “Servidor Ativo”, significa que a ativação do canal foi realizada com sucesso, senão será mostrada a mensagem da exceção que ocorreu.

5.3.2 Implementação do Cliente

A implementação dos clientes é semelhante no âmbito da distribuição. A diferença está na modelagem bem como na animação gerada pelo recebimento de determinado evento.

Na Figura 26 mostramos o código executado quando clicamos no botão conectar.

```
...
if(tbNome.Text == "")
{
    MessageBox.Show("Você deve digitar seu nome para
conectar!", "Erro", MessageBoxButtons.OK, MessageBoxIcon.Error);
    tbNome.Focus();
}
else if (!conectado)
{
    if (!canalRegistrado)
        Cliente.AVDCliente.RegistrarCanal();
    Cliente.AVDCliente.Conectar(tbNome.Text);
    if (conectado)
    {
        tbNome.Visible = false;
        btnConectar.Visible = false;
    }
    else
    {
        tbNome.Text = "";
        tbNome.Focus();
    }
}
...
```

Figura 26. Código Executado Para Tentar Conectar o Cliente.

Como podemos observar na Figura 26, o primeiro passo é verificar se o usuário digitou um nome para criação do canal com o servidor. Depois que o usuário digitou um nome vamos verifica se o usuário já está conectado, senão estiver ele registra um canal e depois estabelece a conexão com o servidor.

Na Figura 27 podemos ver o código *RegistrarCanal()* e *Conectar(nome)*.

```
...
//Código executado para registrar o canal de comunicação
public static void RegistrarCanal()
{
    try
    {
        System.Configuration.ConfigurationSettings.GetConfig("DNS");
        GenuineGlobalEventProvider.GenuineChannelsGlobalEvent += new
        GenuineChannelsGlobalEventHandler(GenuineChannelsEventHandler);
        RemotingConfiguration.Configure("Biologia.exe.config");
        Biologia.FrmBiologia.canalRegistrado = true;
    }
    catch(Exception ex)
    {
        MessageBox.Show(ex.Message, "Erro");
    }
}

//Código executado para Conectar o Cliente com o Srevidor
public static void Conectar(string nome)
{
    try
    {
        AVDCliente.nome = nome;
        //Liga o cliente ao Servidor
        IRetorna iRetorna = (IRetorna)
        Activator.GetObject(typeof(IGrupoDeAVD),
        ConfigurationSettings.AppSettings["RemoteHostUri"] + "/Retorna.rem");
        CourtCollection.Attach("/Chat/GlobalRoom", AVDCliente.Instancia,
        iRetorna.Retorna());
        IGrupoServidor iGrupoServidor = (IGrupoServidor)
        Activator.GetObject(typeof(IGrupoServidor),
        ConfigurationSettings.AppSettings["RemoteHostUri"] +
        "/GrupoServidor.rem");
        AVDCliente.IGrupoDeAVD =
        iGrupoServidor.EntrarNoGrupo(AVDCliente.Instancia, AVDCliente.nome);
        Biologia.FrmBiologia.conectado = true;
    }
    catch(Exception ex)
    {
        MessageBox.Show(ex.Message, "Erro");
        Biologia.FrmBiologia.conectado = false;
    }
}
```

Figura 27. Código Executado Para Conectar o Cliente com Servidor.

Nós temos no cliente, semelhante ao lado do servidor, um arquivo de configuração que também possui o mesmo nome do executável mais *.config*, este arquivo contém, dentre outras informações, o IP do servidor bem como a porta utilizada para comunicação. Após o registro do canal é feita a ligação do cliente com o servidor.

5.4 Considerações Finais

Mostramos neste capítulo uma visão geral do modelo proposto para distribuição do protótipo. Mostramos parte do código do Servidor do Cliente e a Interface utilizada para comunicação. Toda camada de comunicação foi desenvolvida baseada na biblioteca *GenuineChannel* que é uma variante do *.NET Remoting*, como foi comentado anteriormente.

No próximo capítulo vamos descrever o ambiente utilizado para realização dos testes de latência, bem como o modelo de testes adotado.

CAPÍTULO VI

6. ANÁLISE DOS RESULTADOS

6.1 Introdução

Neste capítulo, apresenta os resultados coletados nos testes realizados com o protótipo desenvolvido nesta dissertação com o protótipo desenvolvido por Ribeiro (2005).

A análise realizada abrange os seguintes aspectos:

- a) Obtenção da latência de comunicação exibida pelos protótipos, através da mensuração dos tempos de resposta dos sistemas;
- b) A identificação das tendências quanto à escalabilidade dos protótipos.

6.2 Descrição do Ambiente Utilizado

Para realização dos testes foi utilizado um laboratório composto por 20 máquinas, com a seguinte configuração básica:

- Processador *AMD Sempron 2300+*;
- 512 MB de memória RAM DDR 400MHz;
- *Hard Disk* de 80GB (Conexão Serial);
- Placa de vídeo *on-board* com memória compartilhada;
- Sistema Operacional Windows XP Professional.

A rede utilizada foi uma LAN *Fast Ethernet* de 100 Mbps comutada.

Neste sistema a latência será calculada considerando o tempo gasto por um pacote de dados para percorrer o tráfego de rede que possui a origem em um cliente, em seguida enviado para o servidor e finalmente replicado em todos os outros clientes conectados, inclusive o cliente de origem.

Para tentarmos reduzir o número de variáveis envolvidas neste processo de medição, foram utilizadas as mesmas máquinas para os testes deste protótipo e o desenvolvido por Ribeiro (2005).

No momento da realização dos testes a rede estava sendo utilizada unicamente para este fim.

Inicialmente, tentamos realizar os testes de latência sincronizando os relógios dos clientes com o comando *"net time \\Nome do servidor /set /yes"* e posteriormente conectávamos no servidor, cinco, dez, quinze e vinte clientes respectivamente. Com isso verificávamos a diferença de tempo entre a maior e menor latência dos clientes conectados e tínhamos uma média na latência. Mas este tipo de teste se mostrou ineficiente, pois depois de certo tempo os relógios não estavam mais totalmente sincronizados e geravam dados inconsistentes.

Infelizmente, a plataforma .NET não possui um canal de eventos para testes de latência. Por isto foi utilizado *QueryPerformanceCounter* e *QueryPerformanceFrequency* para capturar o relógio em alta precisão.

Utilizamos o seguinte método: quando um cliente, após conectar, clica no botão relógio que envia um evento (*int*) para o servidor e posteriormente aos clientes conectados, capturamos no servidor o relógio local com o método *Start()*, e após envio para todos os cliente utilizamos o método *Stop()*. Depois utilizamos o método *Duration(1)* com o parâmetro

1 que indica apenas uma interação e retorna a diferença de tempo em nanosegundos. Porém, para apresentarmos os resultados em milisegundos dividimos o resultado por 1.000.000.

O código da classe *QueryPerfCounter* é mostrado na Figura 28.

```
// QueryPerfCounter.cs
using System;
using System.ComponentModel;
using System.Runtime.InteropServices;
public class QueryPerfCounter
{
    [DllImport("KERNEL32")]
    private static extern bool QueryPerformanceCounter(out long
    lpPerformanceCount);

    [DllImport("Kernel32.dll")]
    private static extern bool QueryPerformanceFrequency(out long lpFrequency);
    private long start;
    private long stop;
    private long frequency;
    Decimal multiplier = new Decimal(1.0e9);
    public QueryPerfCounter()
    {
        if (QueryPerformanceFrequency(out frequency) == false)
        {
            // Frequency not supported
            throw new Win32Exception();
        }
    }
    public void Start()
    {
        QueryPerformanceCounter(out start);
    }
    public void Stop()
    {
        QueryPerformanceCounter(out stop);
    }
    public double Duration(int iterations)
    {
        return (((double)(stop - start)* (double) multiplier) / (double)
        frequency)/iterations);
    }
}
```

Figura 28. Classe QueryPerfCounter.

6.3 Análise da Latência

A latência em redes de comunicação pode variar muito devido a inúmeros fatores, como o tipo de tráfego de dados, congestionamento da rede, protocolos de comunicação, etc. Além disso, está implicitamente considerado a latência do sistema operacional na passagem dos dados entre a aplicação e interface de rede.

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, 1995].

Para a realização desta análise, as mensagens de eventos foram enviadas 100 vezes para os nós (computadores) que possuíam um ambiente cada.

A Figura 29 mostra os dados referentes à latência de comunicação com base no aumento de clientes conectados entre o protótipo .NET e CORBA [RIBEIRO et al, 2005].

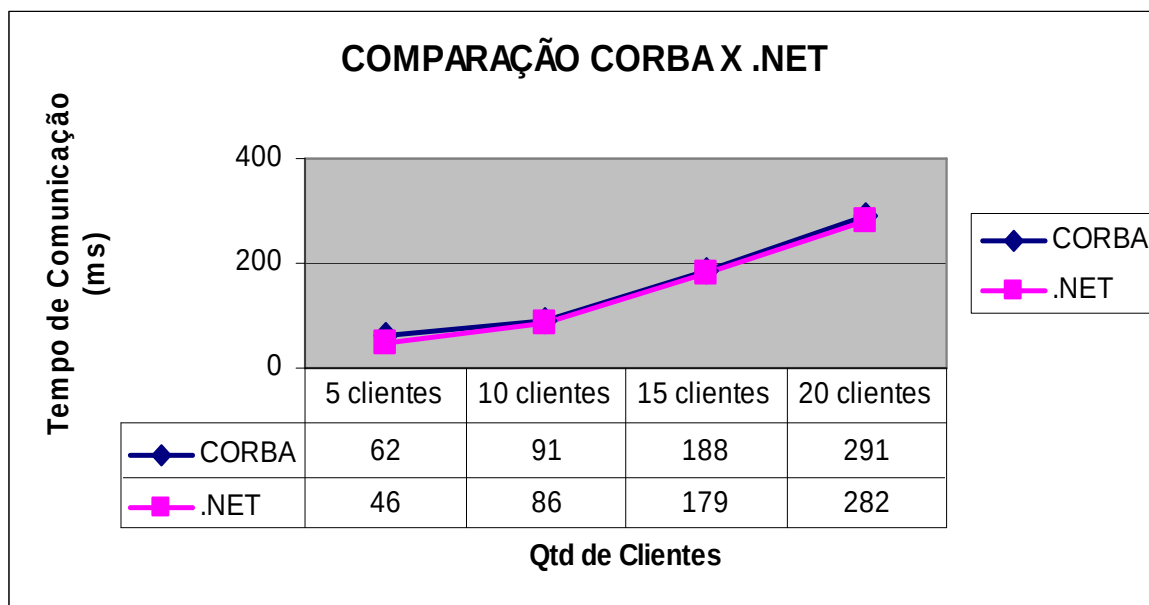


Figura 29. Latência dos Protótipos.

Pode-se observar na Figura 29 que os resultados obtidos foram muito próximos com apenas uma pequena vantagem do sistema .NET.

Na Figura 30 podemos analisar o crescimento da latência em porcentagem com base no aumento do número de clientes conectados.

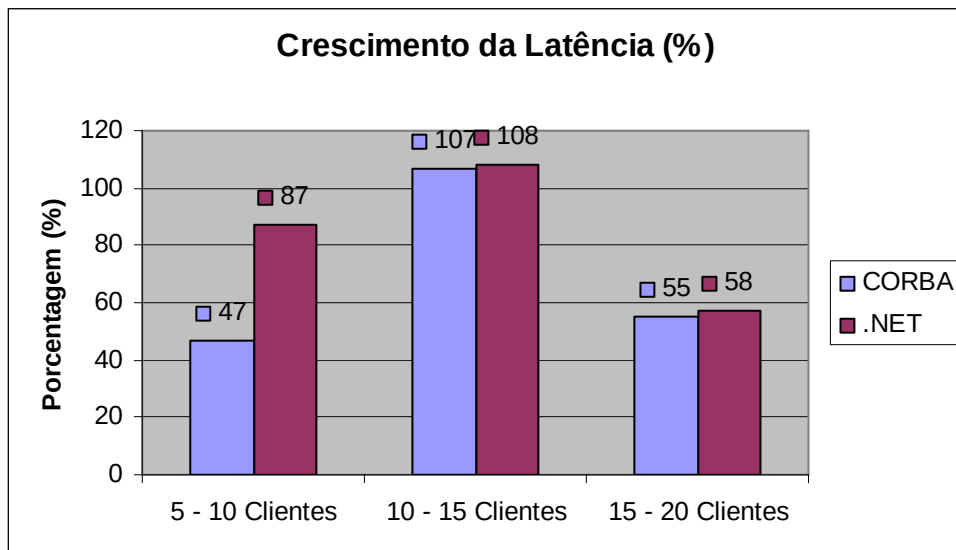


Figura 30. Porcentagem da Latência com Base na Escalabilidade.

Podemos observar que a escalabilidade do sistema CORBA foi melhor que a do sistema .NET no aumento de 5 para 10 clientes e semelhante nos demais casos, pois teve um crescimento muito próximo de latência em porcentagem com base no aumento de clientes.

CAPÍTULO VII

7. CONCLUSÕES E TRABALHOS FUTUROS

Este capítulo tem o objetivo de apresentar as principais características observadas com base nos testes e resultados coletados. Também será apresentada uma proposta de trabalhos futuros.

Pode-se observar com as pesquisas feitas que a maioria dos estudos em AVDs são na distribuição de ambientes virtuais idênticos. Neste trabalho, visou-se o uso de AVDs distintos e com comunicação nos dois sentidos (de um AVD para outro e vice-versa).

Ambas as plataformas .NET e CORBA se comportaram com eficiência como camada intermediária de comunicação entre AVDs.

Apesar do protótipo .NET ter sido um pouco melhor em termos de latência, com base em cinco, dez, quinze e vinte clientes, se mostrou com um aumento grande de latência no aumento de 5 para 10 clientes e semelhante nos demais casos.

O protótipo desenvolvido se mostrou eficiente em AVDs e podendo ser utilizado em vários outros tipos de sistemas distribuídos, necessitando poucas alterações na sua estrutura.

O sistema desenvolvido por Ribeiro (2005) possui algumas facilidades, pois não há necessidade de estar mudando um arquivo de configuração para que cliente encontre servidor. Para isto foi utilizado o ORB Visibroker da Borland que segue todas as especificações CORBA com exceção do serviço de nomeação ou localização de objetos. Já o protótipo .NET proposto neste trabalho existe esta necessidade.

O sistema proposto permite comunicação de tipos de dados menores que 32 bits como *char de 16 bits*, já a arquitetura CORBA utiliza a IDL que possui, por exemplo, um tamanho mínimo do tipo *char* de 32 bits, que pode ser inconveniente em aplicações como a citada neste trabalho, pois desta forma se faz à utilização não eficiente da largura de banda da rede. Tendo em vista que não são necessários 32 bits para especificar 6 opções, sendo 5 opções para representar o mundo da Biologia (5°C, 15°C, 25°C, 35°C e 45°C) e 1 opção para o mundo da Química (Cianeto).

Como continuações deste trabalho, pode-se citar os seguintes tópicos:

- Realizar testes de latência na Internet com um protocolo de comunicação HTTP.
- Criar novas arquiteturas de distribuição para testar o desempenho das arquiteturas.
- Criar mecanismo para que o cliente possa encontrar de forma automática o servidor, sem a necessidade de utilizar um arquivo de configuração que contenha o IP do servidor.
- Calcular o esforço computacional da aplicação para renderizar estes ambientes e desenvolver novas técnicas para aperfeiçoar esta modelagem.
- Realizar testes junto a professores e alunos para verificar a utilidade do sistema.

REFERÊNCIAS BIBLIOGRÁFICAS

- ALMEROTH, K. C., Ammar, M. H., Multicast Group Behavior in the Internet's Multicast Backbone (Mbone), IEEE HPDC 96, IEEE Press, Syracuse, August, 1996, pp.209-216.
- ARAÚJO, R. B., Especificação e Análise de um Sistema Distribuído de Realidade Virtual, Tese de Doutorado, EPUSP, Julho, 1996.
- ASADUZZAMAN, Ahm - Forget the DCOM Pain and Use Remoting or Web Services – Documento disponível on-line em: <http://www.devarticles.com/c/a/Web-Services/Forget-the-DCOM-Pain-and-Use-Remoting-or-Web-Services/>, 29/09/2003. Acesso em 12/08/2005.
- COLIN, P. Fahey., C# OpenGL Wrapper. Documento on-line disponível em: <http://www.colinfahey.com/opengl/csharp.htm>, acesso em 15/03/2005.
- COSTA, Fábio M. e KON, Fabio. Novas Tecnologias de Middleware: Rumo à Flexibilização e ao Dinamismo. Simpósio Brasileiro de Redes de Computadores, 2002.
- COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T., *Distributed Systems: Concepts and Design*. Addison-Wesley, 1994.
- CRUZ-NEIRA, C. et al. - The CAVE Audio Visual Experience Automatic Virtual Environment - Communication of the ACM - 35(6):64-72, June 1992. Documento on-line: <http://portal.acm.org/citation.cfm?id=129892>, acesso em 25/08/2005.
- DIOT, C., Dabbous, W., Crowcroft, J., Multipoint Communication: A Survey of Protocols, Functions, and Mechanisms, IEEE JSAC, Vol.15, No.3, April, 1997, pp.277-290.
- GAUTIER, L., DIOT, C., Mimaze, a Multiuser Game on the Internet, Rapport de Recherche, No.3248, Setembro, 1997.

GENUINECHANNELS, Documento on-line: <http://www.genuinechannels.com/> acesso em 21/05/2005.

GIRARDELLI, Maria de Fátima. *Qual é a diferença entre multidisciplinaridade, Multidisciplinaridade e transdisciplinaridade?*. Manaus-AM. Disponível em (http://www.uol.com.br/novaescola/ed/124_ago99/html/comcertezadidatica.htm), maio/2002.

HANCOCK, D. - Viewpoint: Virtual Reality in Search of Middle Ground, IEEE Spectrum, 32(1):68, Jan 1995.

HOLBROOK, H. W., Singhal, S. K.; Cheriton, D. R., Log-Based Receiver- Reliable Multicast for Distributed Interactive Simulation, in SIGCOMM 95, Boston, August 1995, pp.328-341.

HUANG, Wei *Comparison of Microsoft .NET and CORBA*. Documento on-line: <http://www.iit.edu/~huanwei/doc/corba-net.html>, 2000

IONA Technologies, .NET Connector Programmer's Guide. Documento on-line Documento on-line: <https://www.iona.com/support/docs/orbix/6.1/develop/dotNET/dotnet.pdf>, 2004. Acesso em 24/08/2005.

KIRNER, Cláudio. *Sistemas de Realidade Virtual*. Documento on-line, <http://www.dc.ufscar.br/~grv/>, 1998. Acesso em 22/07/2005.

KIRNER, Cláudio. Realidade Virtual: Dispositivos e Aplicações. VII Escola de Informática da SBC Regional Sul, 10 a 14 Maio/1999, pp. 135-158.

MACEDONIA, M. R.; ZYDA, M. J. A Taxonomy for Networked Virtual Environments. *IEEE multimedia*, 1995.

MACDONALD, Matthew. Microsoft .NET Distributed Applications: Integrating XML Web Services and .NET Remoting. Microsoft Press, 2003.

- MORIE, J.F. - Inspiring the Future: Merging Mass Communication – Art. Entertainment and Virtual Environments - Computer Graphics – 28(2):135-138, May 1994. Documento on-line: <http://portal.acm.org/citation.cfm?id=178951.178973>, acesso em 22/08/2005.
- NEIDER, Jackie, OpenGL programming Guide, 2000.
- OBERMEYER, Piet and HAWKINS, Jonathan. Microsoft .NET Remoting: A Technical Overview. Technical Report 013, Microsoft Corporation, 2001. Documento on-line
Documento on-line: <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndotnet/html/hawkremoting.asp>. Acesso em 18/08/2005.
- OMG e o Padrão CORBA. Documento on-line. <http://www.corba.hpg.ig.com.br/>, 2004.
- PULLEN, J. M., Laviano, V. P. A Selectively Reliable Transport Protocol for DIS, 13rd Workshop on Standards for Interoperability of DIS, 1995.
- PULLEN, J.M. et al. Limitations of Internet Protocol Suite for Distributed Simulation in the Large Multicast Environments, Working Internet Draft <draft-ietf-lsma-limitations-01.txt>, March, 1997.
- RAJ, Gopalan Suresh. *A Detailed Comparison of CORBA, DCOM and Java/RMI*. Documento on-line: <http://my.execpc.com/~gopalan/misc/compare.html>, 2004.
- RIBEIRO, Marcos Wagner de Souza; LAMOUNIER, Edgard Jr.; CARDOSO, Alexandre. Uma Proposta de Arquitetura na Distribuição de Ambientes Virtuais Multidisciplinares. I Workshop de Aplicações de Realidade Virtual, Faculdade de Engenharia Elétrica, Universidade Federal de Uberlândia, Novembro de 2005.
- RODELLO, Ildeberto;. Um Ambiente Virtual para Auxiliar o Ensino de Química em Escolas de Ensino Fundamental. In: XXII Congresso da Sociedade Brasileira de Computação. Anais do VIII WIE. Florianópolis, SC, 2002.
- ROGERSON, Dale. Inside COM. Microsoft Press, 1997.
- SCHONIG, Hans-Jurgen; GESCHWINDE, Ewald. Mono Kick Start. Sams, 2003.

- SNELL, James; TIDWELL, Doug; and KULCHENKO, Pavel. Programming Web Services with SOAP. O'Reilly & Associates, 2001.
- TANEMBAUM, A. S. *Distributed Operating Systems*. Prentice Hall International Editions, 614p, New Jersey, 1995.
- TODESCO, Glauco. Criação de Ambientes Virtuais Multi - Usuário Através do Padrão MPEG – 4, Dissertação de Mestrado, Universidade Federal de São Carlos, Agosto de 2000.
- VAN DAN, A., VR as a Forcing Function: Software Implications of a new Paradigm, IEEE Symposium on Research Frontiers in Virtual Reality, October, 25-26, CA, 1993.
- WOO, Mason; NEIDER, Jackie; DAVIS, Tom; SHREINER, Dave. *OpenGL Programming Guide: the official guide to learning OpenGL, version 1.2*. 3rd ed. Reading, Massachusetts: Addison Wesley, 1999. 730 p.