UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO

(Bacharelado)

ANÁLISE COMPARATIVA ENTRE AS ESPECIFICAÇÕES DE OBJETOS DISTRIBUÍDOS DCOM E CORBA UTILIZANDO O AMBIENTE DE DESENVOLVIMENTO DELPHI

TRABALHO DE CONCLUSÃO DE CURSO SUBMETIDO À UNIVERSIDADE REGIONAL DE BLUMENAU PARA A OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA COM NOME EQUIVALENTE NO CURSO DE CIÊNCIAS DA COMPUTAÇÃO — BACHARELADO

RUBENS BÓSIO

BLUMENAU, JUNHO/2000.

2000/1-64

ANÁLISE COMPARATIVA ENTRE AS ESPECIFICAÇÕES DE OBJETOS DISTRIBUÍDOS DCOM E CORBA

RUBENS BÓSIO

ESTE TRABALHO DE CONCLUSÃO DE CURSO FOI JULGADO ADEQUADO PARA OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA DE TRABALHO DE CONCLUSÃO DE CURSO OBRIGATÓRIA PARA OBTENÇÃO DO TÍTULO DE:

BACHAREL EM CIÊNCIAS DA COMPUTAÇÃO

_____

Prof. Marcel Hugo — Orientador na FURB

_____

Prof. José Roque Voltolini da Silva — Coordenador do TCC

BANCA EXAMINADORA

_____

Prof. Marcel Hugo

_____

Prof. Maurício Capobianco Lopes

_____

Prof. Wilson Pedro Carli

AGRADECIMENTOS

Agradeço ao meu orientador Prof. Marcel Hugo, pela confiança e calma transmitida durante o desenvolvimento do trabalho.

Um agradecimento à minha esposa Selone, pelo apoio ao abrir mão da minha companhia nos diversos finais de semana utilizados para que eu pudesse concluir este trabalho.

Aos meus pais: Salvador e Teresa, pelo grande apoio a mim oferecido durante toda a minha vida para que eu pudesse chegar a essa condição de formando.

Agradeço também a todos os colegas da FURB que proporcionaram um ambiente de amizade e de solidariedade (além das cachaçadas), durante os difíceis anos de universidade.

Finalmente, um agradecimento a todos os colegas de trabalho da Senior Sistemas.

Sumário

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

RESUMO

Este trabalho apresenta duas das principais tecnologias de objetos distribuídos. De um lado tem-se o DCOM da Microsoft, que vem com uma série de serviços disponibilizados pelos sistemas operacionais desta empresa, simplificando o desenvolvimento de objetos distribuídos e, ao mesmo tempo, vinculando esta tecnologia à plataforma Windows. De outro lado o CORBA, definido pelo *Object Management Group* (OMG) com uma das principais características de ser independente de plataforma, de hardware ou de linguagem de programação. Como forma de demonstrar as tecnologias, foi feito um protótipo de uma ferramenta de consulta a banco de dados, desenvolvida em duas versões, cada uma com uma das duas tecnologias. Como objetivo do trabalho, é feita uma comparação entre as tecnologias, em vários aspectos, como performance e facilidade de implementação.

ABSTRACT

This work presents two of the main distributed object technologies - DCOM and CORBA. The Microsoft DCOM that comes with plenty services provided by the Operating Systems from this company, making the distributed object systems development easier and, at the same time, linking this technology to the Windows platform. CORBA is defined by the Object Management Group - OMG - with one of the main characteristics of being platform and hardware and language independent. As a way of studying these two technologies, a software tool prototype to consult a database was developed in two versions where each one carries on the two different distributed object technologies.

1 INTRODUÇÃO

Percebe-se hoje que, quando se fala em orientação a objetos, não se trata de nenhuma novidade na área de desenvolvimento. Esta tecnologia para desenvolvimento de sistemas já está bastante difundida.

Outro ponto bastante explorado no desenvolvimento de sistemas é o de processamento distribuído, onde as tarefas de um sistema são divididas entre vários computadores.

De acordo com estas necessidades e tendências, surge atualmente o que se chama de Objetos Distribuídos, que vem unir as duas tecnologias acima citadas. O objetivo agora é, cada vez mais, aumentar a interoperabilidade, de forma a facilitar a comunicação entre sistemas, desde um mesmo computador, até a comunicação entre sistemas em pontos diferentes via internet ([MIC2000]).

Dentro dessas necessidades, surgem no mercado duas frentes em especificação de Objetos Distribuídos: CORBA e DCOM.

Estas duas especificações estão se destacando no mercado, pois uma foi desenvolvida por um forte grupo de empresas que se uniu para encontrar uma solução em objetos distribuídos, caso do CORBA e a outra, por ter sido desenvolvida e lançada por uma empresa que é líder em vários segmentos do mercado de software, caso do DCOM.

O CORBA (*Common object request broker architecture*), desenvolvido em 1989 pelo OMG (*Object Management Group*), permite que objetos invoquem métodos em objetos distribuídos em redes, como se estes fossem locais, independentemente de plataforma ([CAP1999]).

O DCOM (*Microsoft® Distributed COM*) é uma extensão do COM, para suportar comunicação entre objetos em diferentes computadores, desde redes locais até a Internet ([MIC2000]). Esta tecnologia, por vir da Microsoft, depende da plataforma que, no caso, é o Windows.

Pretende-se com este TCC, fazer um protótipo de um sistema de acesso remoto a um banco de dados onde serão enviados comandos SQL de um objeto para outro que será

encarregado de executá-lo e retornar os dados resultantes. Esta implementação será feita para cada uma das duas tecnologias, de forma a possibilitar uma comparação de seus comportamentos, analisando sua execução e o próprio trabalho de desenvolvimento.

Para a elaboração do trabalho, pretende-se utilizar a ferramenta de desenvolvimento Delphi 5.0, pelo fato de ser uma ferramenta bastante conhecida, além de suportar as tecnologias CORBA e DCOM, aqui estudadas.

A análise será feita orientada a objetos utilizando a linguarem UML com a ferramenta *Rational Rose*.

A UML é a linguagem padrão para especificar, visualizar, documentar e construir artefatos de um sistema e pode ser utilizada com todos os processos ao longo do ciclo de desenvolvimento e através de diferentes tecnologias de implementação. Buscou-se unificar as perspectivas entre os diversos tipos de sistemas e fases de desenvolvimento de forma que permitisse levar adiante determinados projetos que antes não eram possíveis pelos métodos existentes ([FUR1998]).

1.1 OBJETIVOS

O objetivo principal deste trabalho é fazer um protótipo de uma ferramenta de consulta a banco de dados de forma remota para que se possa fazer um comparativo entre as principais características das duas tecnologias de desenvolvimento de Objetos Distribuídos: DCOM e CORBA.

1.2 ESTRUTURA

Este trabalho foi estruturado em sete capítulos de maneira a apresentar primeiramente os objetivos do trabalho e a sua estrutura no capítulo 1.

Em seguida, no capítulo 2, alguns conceitos com relação à orientação a objetos, sistemas distribuídos e objetos distribuídos, serão apresentados de uma forma superficial.

Os capítulos 3 e 4 demonstram as tecnologias estudadas, CORBA e DCOM respectivamente, mostrando suas arquiteturas, características e seus funcionamentos.

No capitulo 5, serão apresentados o protótipo, com toda a definição, diagrama de classes, caso de uso e diagrama de seqüência, mostrando a estrutura utilizada para o desenvolvimento da aplicação e as principais características da ferramenta Delphi, utilizada para o desenvolvimento do protótipo, com relação a CORBA e DCOM.

No capitulo 6 será feita uma comparação entre estas tecnologias, de forma a visualizar as diferenças de nomenclatura utilizadas, as limitações e as diferenças quanto à facilidade de desenvolvimento e uma seqüência de teste para a comparação da performance de cada tecnologia.

No capítulo 7, serão feitas as considerações finais sobre o trabalho feito, comentando as conclusões, dificuldades encontradas e sugestões para trabalhos futuros.

2 OBJETOS DISTRIBUÍDOS

Este capítulo tem o objetivo de fornecer uma fundamentação sobre Objetos Distribuídos, comentando também tecnologias como sistemas distribuídos e orientação a objetos.

2.1 SISTEMAS DISTRIBUÍDOS

Não se pode negar que a influência da filosofia Cliente/Servidor *(Client/Serve*r) em todas as empresas que utilizam informática foi imensa. Hoje o ambiente cliente/servidor está presente em praticamente todas estas empresas contribuindo com um grande passo a partir de sistemas de *mainframe* centralizados ([MAI1997]).

Ultimamente observa-se alguns autores alertando para a “Nova Onda” de cliente/servidor, ou até mesmo em alguns casos, a “Segunda Geração” de cliente/servidor. Estes autores buscam chamar a atenção de profissionais de desenvolvimento de software para o fato de que, agora que o ambiente cliente/servidor já está presente na maioria das empresas, é importante se preparar para o próximo passo ([MAI1997]).

Basicamente esta segunda geração de cliente/servidor, é um ambiente onde cliente não tem mais um papel único de ser apenas cliente, e o servidor também não apresenta mais uma função única, por exemplo, de apenas servir dados. Atualmente nas empresas tem-se um ambiente onde as estações são computadores poderosos e com sistemas operacionais poderosos de 32-bits, multitarefa, *multithreading*, que suporta vários protocolos de rede nativamente e com uma interface gráfica intuitiva e de fácil uso, provendo mais poder ao seu usuário. Da mesma forma, os servidores são computadores ainda mais poderosos e que, para prover todos os serviços solicitados pelos clientes, utiliza-se de chamadas a outros servidores, espalhados pela rede. Portanto cliente não é mais apenas cliente e servidor não é mais apenas um servidor, todos os computadores na rede corporativa podem assumir todos os papéis. Na verdade, este tipo de ambiente não é novo, e já tem nome: “Sistemas Distribuídos” ([MAI1997]).

A “segunda geração”, ou “nova onda” de cliente/servidor, nada mais é do que a realização, por parte de grandes empresas, de que hoje é completamente possível trabalhar em um ambiente distribuído, transparente, executando tarefas em paralelismo real, obtendo um

melhor desempenho no resultado das aplicações. Este tipo de ambiente já é uma realidade, e não faz parte mais de artigos que prevêm as tendências da computação. Isso não é mais uma tendência, é uma realidade ([MAI1997]).

2.2 ORIENTAÇÃO A OBJETOS

Além de ambientes distribuídos, outra realidade que está presente na computação moderna é a Orientação a Objetos (OO). Não existe um profissional que esteja envolvido ativamente com a área de desenvolvimento de software que não tenha observado a influência da orientação a objetos. Na nova era da computação, os métodos e processos de desenvolvimento de software, tão famosos e estáveis no passado, já não mais se aplicam para as soluções dos problemas atuais. Não se está mais em um ambiente apenas voltado para o processamento centralizado, com *mainframe*s e terminais burros modo caractere e apenas acessos a dados. A computação mudou, o profissional atualizado precisa apenas olhar para os lados para observar isso. As redes corporativas, a integração entre máquinas de diferentes arquiteturas e sistemas operacionais, a Internet e outras tecnologias modernas já estão presentes em praticamente todas as empresas. Para se desenvolver software em um novo ambiente como este, são necessários métodos e processos mais adequados, num novo paradigma, que é a Orientação a Objetos ([MAI1997]).

2.2.1 VISÃO COMERCIAL

Com a vinda de linguagens com suporte ao conceito de orientação a objeto e o crescimento da Internet, os gerentes estão novamente empolgados com a possibilidade de usar a tecnologia de objetos de software e de ultrapassar limites com aplicações grandes e complexas, constituídas por uma união de vários módulos previamente desenvolvidos em forma de objetos, conseguindo assim a entrega mais rápida de soluções, a um custo mais baixo ([MIC2000]).

Uma estrutura de objetos para construir aplicações traz economias no desenvolvimento de software por ([MIC2000]):

a) acelerar o desenvolvimento: o programador pode construir aplicações mais rapidamente, unindo componentes já prontos para determinadas funções;

b) diminuir custos de integração: fornecendo um conjunto comum de interfaces para programas de fornecedores diferentes, significa menos trabalho para integrar objetos em soluções completas;
c) melhorar a flexibilidade no desenvolvimento: fica mais fácil a customização de uma solução de software para áreas diferentes de uma companhia, simplesmente mudando alguns dos objetos na aplicação;
d) diminuir custos de manutenção: isolar funções em objetos pequenos é uma forma barata e eficiente para atualizar partes de um sistema sem gerar impacto na aplicação inteira.

Uma arquitetura de objetos distribuídos aplica estes benefícios em uma escala maior de aplicações multi-usuário ([MIC2000]).

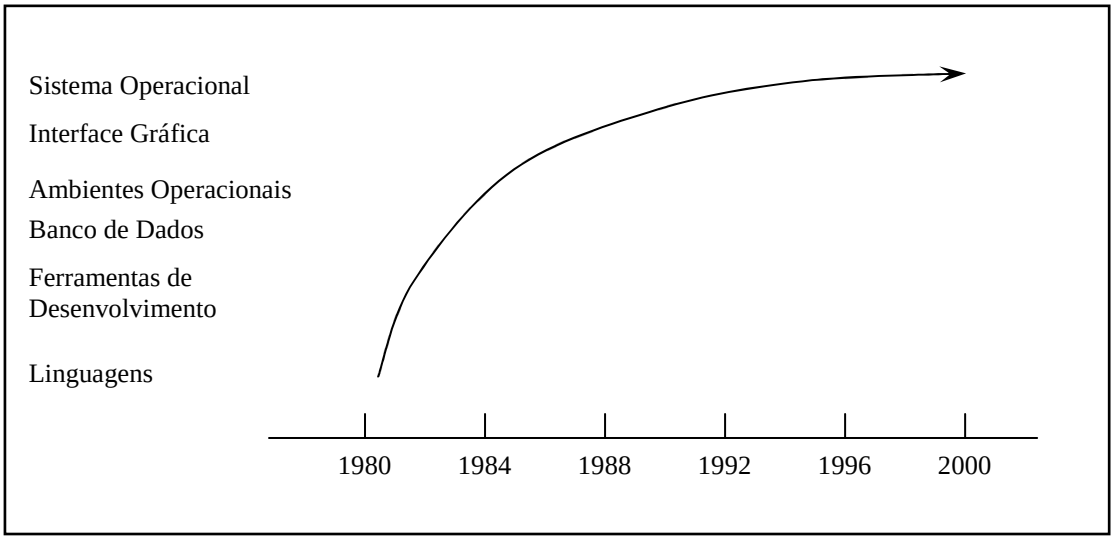
Aplicações feitas com componentes são mais fáceis de depurar. Por exemplo, considere-se o "*Bug do Milênio*" que em muitas organizações foi motivo de grandes despesas para as alterações nos sistemas. Na verdade, a causa de tantas despesas para alteração não foi a data, mas sim, a estrutura dos sistemas. Se fosse um sistema orientado a objetos, onde haveria um único componente de data utilizado em todo o sistema, o trabalho para alteração seria bem menor e muito mais barato ([MIC2000]).

2.2.2 VISÃO DO DESENVOLVEDOR

Hoje, mesmo que um profissional da área de desenvolvimento de software queira desenvolver um novo sistema sem usar a orientação a objetos, vai encontrar muitas dificuldades, pois praticamente todas as ferramentas de ponta atualmente usadas no processo de desenvolvimento de software são otimizadas para trabalhar usando-se os conceitos de orientação a objetos. Há muito tempo, vários autores já vêm fazendo previsões sobre a influência da orientação a objetos nas várias áreas da computação. Se forem observados os acontecimentos destas áreas nos últimos anos, verifica-se que todas as previsões se transformaram em realidade ([MAI1997]).

A Figura 1 dá uma idéia de como a Orientação a Objetos foi influenciando as diversas áreas da computação ao longo dos últimos 20 anos.

Figura 1 - Influência da Orientação a Objetos nas áreas da computação



Fonte: [MAI1997]

2.2.2.1 OO E LINGUAGENS DE PROGRAMAÇÃO

A primeira influência da orientação a objetos ocorreu através das linguagens de programação. Durante os anos 80, parece que houve uma febre entre criadores de linguagens de programação e praticamente todas as linguagens passaram embutir os conceitos de OO. Linguagens como BASIC, Pascal, C e até mesmo COBOL e LISP passaram a embutir os conceitos de OO (que se transformaram em linguagens como: Object Pascal, C++, Object COBOL e CLOS). Até mesmo as linguagens proprietárias e baseadas em *script*s, de ferramentas gráficas de prototipação hoje são consideradas orientadas a objetos (apesar de haver uma grande discussão envolvendo os conceitos: “baseado” em objetos e “orientado” a objetos) ([MAI1997]).

Hoje os compiladores mais vendidos no mercado são os compiladores otimizados para linguagens OO, e também os mais vendidos. Isto é uma evidência de que a orientação a objetos está realmente sendo usada ([MAI1997]).

2.2.2.2 OO E BANCOS DE DADOS

Outra área importante que vem recebendo a influência da OO há algum tempo é a área de Bancos de Dados. A tentativa de integração entre estas duas áreas identificou um problema, que é conhecido como o problema da impedância *(impedance mismatch*), ou seja, o problema de se armazenar objetos em bancos de dados organizados em tabelas relacionais. A busca da solução deste problema deu origem a uma imensa nova indústria, a dos Bancos de Dados Orientados a Objetos (ou OODBMSs, *Object Oriented Data Base Management Systems*). Hoje existem produtos que permitem armazenar objetos como objetos, solucionando o problema da impedância ([MAI1997]).

Outra área que se demonstra em forte crescimento é a dos Bancos de Dados Objeto-Relacionais (ou ORDBMSs, *Object-Relational Data Base Management System*s). Esta área tem potenciais de ser ainda maior que a indústria de OODBMSs, e tudo indica, que com o advento de novas necessidades de armazenamento advindas da orientação a objetos e da Internet, esta venha a ser a principal tecnologia para os bancos de dados do futuro. Recentemente pode-se observar os últimos movimentos das grandes empresas que dominam a área de bancos de dados, como Oracle, Sybase e Informix, que pretendem transformar seus produtos principais em bancos de dados objeto-relacionais ([MAI1997]).

Os agressivos avanços das grandes empresas da área de banco de dados embutindo os conceitos de OO em seus produtos principais, mostram mais uma evidência da aceitação da orientação a objetos ([MAI1997]).

2.2.2.3 OO E A INTERNET

Observa-se também a influência da OO na Internet. Hoje, para se desenvolver software para a Internet (ou uma intranet) existem duas principais tecnologias disponíveis: ActiveX e Java, e as duas são tecnologias que usam os conceitos de orientação a objetos. Portanto os desenvolvedores de software que pretendem estar presentes no mercado promissor de desenvolvimento de software para a Internet, devem ter conhecimento de OO ([MAI1997]).

2.3 OBJETOS DISTRIBUÍDOS

A união destas duas importantes tecnologias (Sistemas Distribuídos e Orientação a Objetos) dá origem à área dos Objetos Distribuídos ([MAI1997]).

Objetos Distribuídos são um paradigma da computação que permite a objetos serem distribuídos através de uma rede heterogênea, e permitir a cada componente a total interoperabilidade e acesso a outros objetos. Para uma aplicação escrita em um ambiente de Objetos Distribuídos, objetos remotos são acessados como se estivessem na máquina que efetua as requisições. Um objeto distribuído é essencialmente um componente, que pode interoperar com outros objetos distribuídos através de sistemas operacionais, redes, linguagens, aplicações, ferramentas e equipamentos diversos. Existe uma tendência de se construir sistemas computacionais abertos utilizando objetos distribuídos ([CAP1999]).

Objetos Distribuídos são basicamente usar a tecnologia de orientação a objetos em um ambiente distribuído. Como estas duas tecnologias já estão se transformando em realidade atual na grande maioria das empresas, será uma evolução natural. Os Objetos Distribuídos começaram a sua trajetória através do *middleware* (a parte do software responsável pela intercomunicação entre os vários componentes distribuídos em uma rede), mas hoje está invadindo todas as áreas, inclusive a Internet ([MAI1997]).

Uma arquitetura de objetos é sem dúvida uma arquitetura cliente/servidor. Os objetos servidores são aqueles que oferecem um serviço para o resto do sistema, e os objetos clientes são aquelas aplicações que se utilizam destes serviços ([SIE1996]).

Além disso, esses objetos distribuídos possuem as mesmas características principais dos objetos das linguagens de programação: encapsulamento, polimorfismo e herança, tendo, dessa forma, as mesmas principais vantagens: fácil reusabilidade, manutenção e depuração, só para citar algumas ([CAP1999]).

Cada objeto distribuído não opera sozinho. A princípio ele é construído para trabalhar com outros objetos e, para isso, precisa de uma espécie de barramento. Tais barramentos fornecem infra-estrutura para os objetos, adicionando novos serviços que podem ser herdados durante a construção do objeto, ou mesmo em tempo de execução para alcançar altos níveis de colaboração com outros objetos independentes ([CAP1999]).

Já existem vários modelos de objetos distribuídos disponíveis no mercado, como: DSOM *(Distributed System Object Mode*l) da IBM, PDO *(Portable Distributed Object*s) da NeXT, DCOM *(Distributed Component Object Mode*l) da Microsoft, CORBA *(Common Object Request Broker Architectur*e) da OMG, e outros em nível de pesquisas acadêmicas (como Emerald, Arjuana, ILU, etc). Dentre estes concorrentes, dois estão se transformando nos principais padrões de mercado (seja padrão de fato ou de jure). O DCOM da Microsoft e o CORBA da OMG ([MAI1997]).

3 CORBA

3.1 OBJECT MANAGEMENT GROUP (OMG)

A OMG *(Object Management Grou*p, ou Grupo de Gerenciamento de Objetos) é uma organização sem fins lucrativos que tem como objetivo difundir a área de orientação a objetos. Sua missão é descrita como: "... estabelecer guias e especificações de gerência de objetos para a indústria, visando prover uma base comum para o desenvolvimento de aplicações distribuídas". A OMG foi fundada em 1989 e conta hoje com mais de 750 empresas membro, entre elas, IBM, HP, Sun, DEC, Microsoft, Netscape ([MAI1997]).

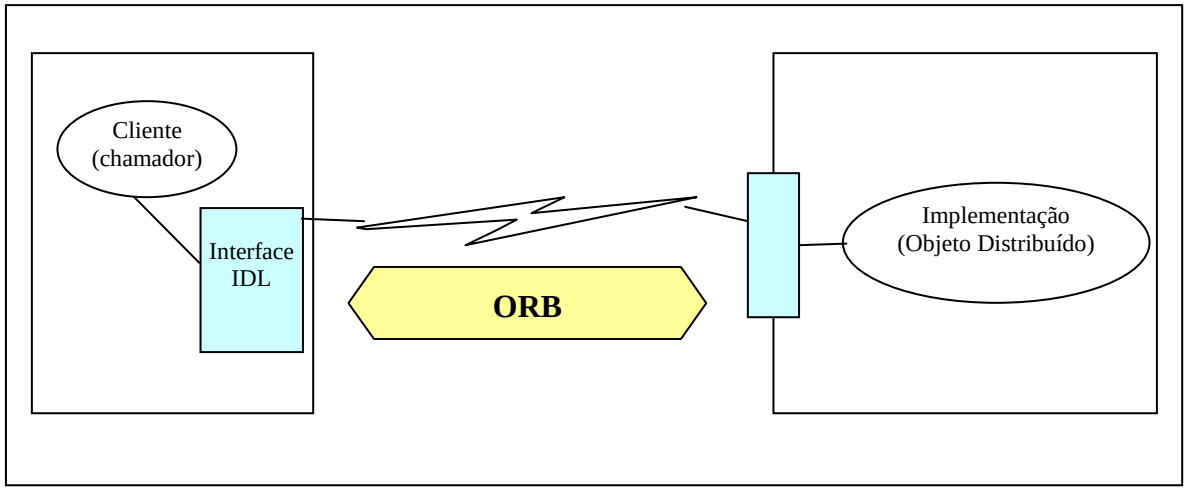
A principal contribuição da OMG até o momento é o padrão CORBA *(Common Object Request Broker Architectur*e, ou Arquitetura de Corretor de Requisições de Objetos Comum). CORBA é um padrão criado pelas empresas membro e endossado pela OMG que define como objetos devem interoperar em um ambiente distribuído ([MAI1997]).

3.2 VISÃO GERAL

Uma das características mais importantes deste padrão é a existência de uma linguagem para a definição de interfaces (IDL, *Interface Definition Languag*e) que é uma linguagem, também padronizada pela OMG, independente de arquitetura, que permite se especificar as interfaces dos objetos distribuídos de uma forma que todos possam requisitar serviços a eles ([MAI1997]).

O CORBA abre o mundo não-Microsoft para os programadores de linguagens para Windows. Embora os programas construídos com Delphi, por exemplo, possam ser executados apenas em Windows, eles podem se conectar com outros objetos CORBA sendo executados em diferentes sistemas operacionais. Por exemplo, o CORBA oferece uma boa integração com Java ([CAN2000]).

Figura 2 - Funcionamento Básico do CORBA



Fonte: [MAI1997]

A Figura 2 mostra o funcionamento básico de uma chamada CORBA. O solicitador do serviço enxerga apenas uma interface de um objeto que está distribuído pela rede, e após executar a chamada, apenas aguarda pela resposta. No CORBA esta interação entre chamador e objeto distribuído chamado é feita através de uma Referência para Objeto (ou *Object Referenc*e) que o chamador deve obter antes de executar a chamada. Toda a interação entre o chamador e o objeto distribuído chamado deve ser feita pelo *broker* (ORB), fazendo com que os principais participantes dessa interação não precisem se preocupar com detalhes de comunicação, problemas de rede, ordenação/desordenação *(marshalling/unmarshallin*g), procura e ativação de objetos, etc. Para o chamador todos os serviços são atendidos de uma forma transparente, para o objeto distribuído chamado, todas as requisições se comportam da mesma forma, como se fossem requisições locais ([MAI1997]).

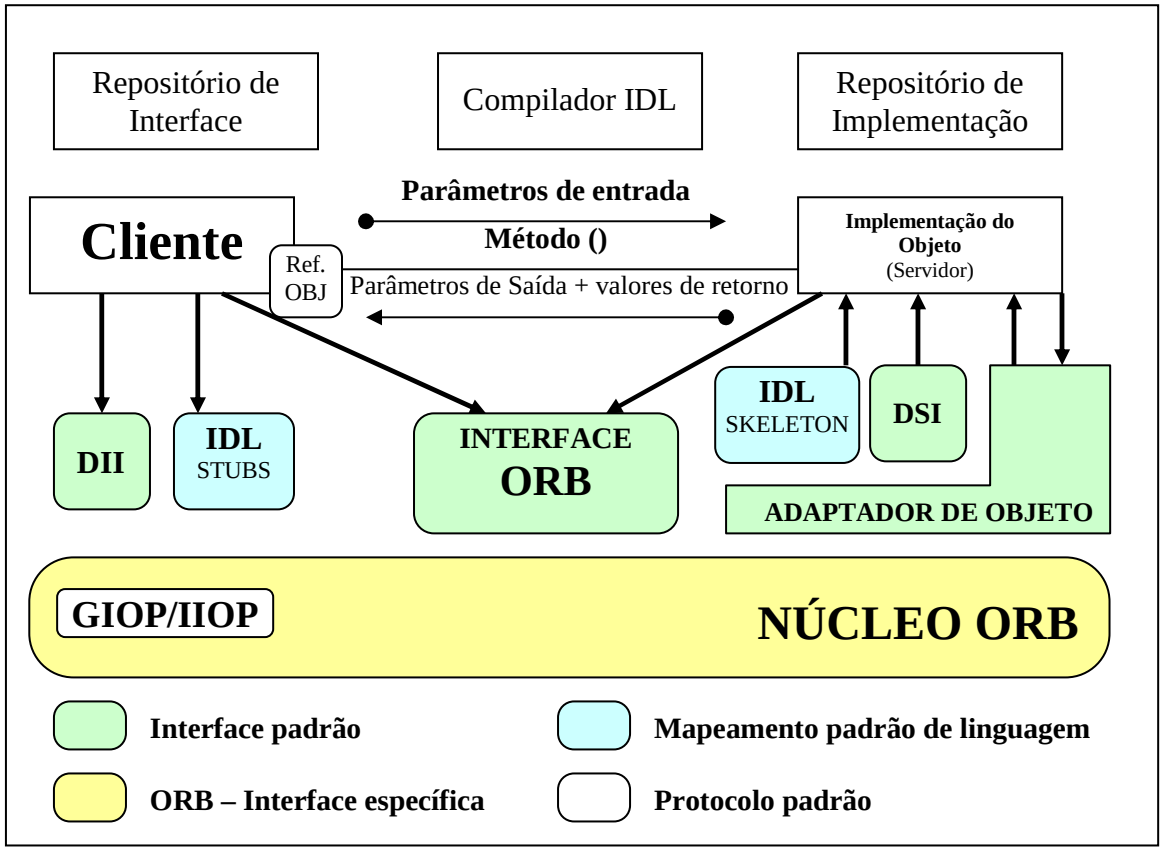
É importante notar que um objeto distribuído não é a mesma coisa que, por exemplo, um objeto C++. Um objeto distribuído tem uma interface exportada, a sua implementação pode ser feita em qualquer linguagem de programação (inclusive linguagens não orientadas a objetos) e pode executar em qualquer processador. Os objetos distribuídos são independentes de linguagem de programação, *hardware* (máquina), sistema operacional, rede, protocolo de comunicação, espaço de endereçamento, compiladores, etc. Podem até mesmo ser chamados de Componentes ([MAI1997]).

As chamadas aos objetos distribuídos são feitas após um mapeamento da IDL para a linguagem de programação sendo usada pelo chamador Este mapeamento é feito automaticamente por pré-processadores IDL, e os módulos gerados são ligados ao código do chamador. Isto permite a possibilidade de que um objeto distribuído possa ser desenvolvido em uma linguagem e o chamador possa ser desenvolvido em outra, realmente provocando a independência entre o chamador e objeto distribuído chamado ([MAI1997]).

3.3 ARQUITETURA CORBA

A Figura 3 mostra em detalhes a estrutura que existe para que a comunicação entre objetos distribuídos seja possível através do CORBA.

Figura 3 –Arquitetura CORBA



Fonte: [SCH1999]

Cada componente da Figura 3 será detalhado na seqüência deste capítulo.

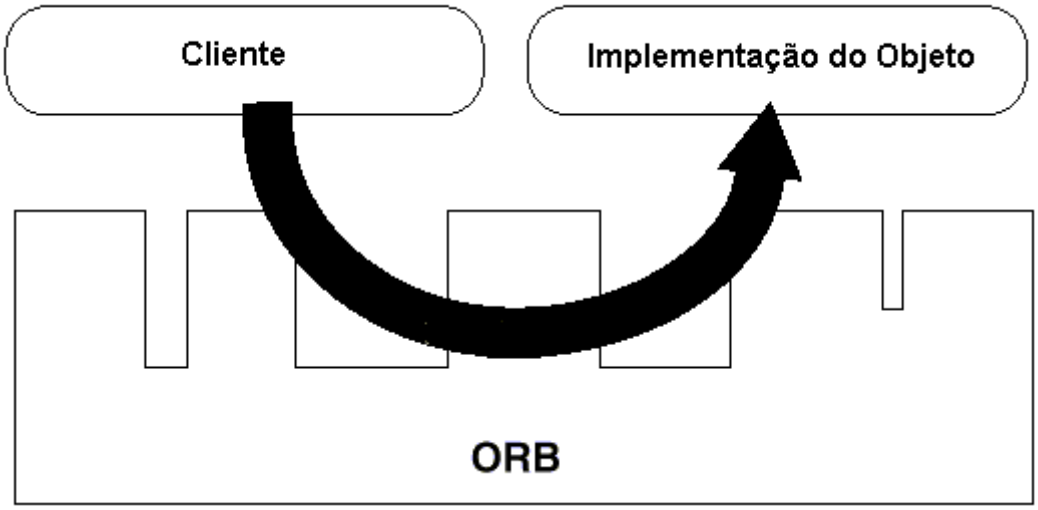
3.3.1 *OBJECT REQUEST BROKER* (ORB)

Objetos clientes precisam fazer requisições aos objetos servidores, que contém a implementação. Esta requisição é passada para o servidor através de um mecanismo chamado ORB.

O ORB fornece um mecanismo para que as requisições dos clientes localizem o objeto servidor de uma forma transparente. O ORB simplifica a programação distribuída porque livra o cliente de certos detalhes na chamada de métodos. Os clientes fazem chamadas de métodos como se fosse uma chamada local. Quando um cliente faz uma requisição, o ORB é responsável por procurar a implementação do objeto, ativando-o se necessário, enviar a requisição para o objeto e retornar qualquer resposta ao cliente ([SCH1999]).

A Figura 4 mostra a forma com que o ORB interage com o objeto cliente e o objeto servidor, mostrado como Implementação do Objeto.

Figura 4 – Requisição do cliente passando pelo ORB



Fonte: [OMG1995]

3.3.2 INTERFACE ORB

Um ORB é uma entidade lógica que pode ser implementada de várias formas (como um ou mais processos, ou um conjunto de bibliotecas). Para isolar as aplicações de detalhes de implementação, a especificação CORBA define uma interface abstrata para um ORB ([SCH1999]).

Como a maioria da funcionalidade do ORB é provida pelos *stub*s, *skeleton*s, invocações dinâmicas ou adaptadores de objetos (explicados mais adiante), apenas poucas operações comuns fazem parte da Interface do ORB ([MAI1997]).

Esta interface fornece várias funções auxiliares como um conversor de referência de objeto para strings e vice-versa, cria uma lista de argumentos para requisições feitas através da "*dynamic invocation interface*" descrita mais adiante ([SCH1999]).

3.4 *INTERFACE DEFINITION LANGUAGE* (IDL)

A IDL é a linguagem da OMG para especificar interfaces. A interface de um objeto é especificada em IDL e compõem conjunto de operações e seus parâmetros ([MAI1997]).

A IDL define os tipos de objetos, especificando as suas interfaces. Uma interface consiste em um conjunto de operações identificadas e de parâmetros para estas operações. Perceba que, embora a IDL forneça um esqueleto conceitual para descrever os objetos manipulados pelo ORB, ela não é indispensável para que se tenha um código fonte disponível para o ORB trabalhar. Enquanto informações equivalentes estejam disponíveis em forma de rotinas *stub* ou em uma *interface repository*, um ORB em particular pode ser habilitado para funcionar corretamente ([OMG1995]).

A IDL é a forma com que uma implementação de um objeto informa a um cliente quais operações estão disponíveis e como elas devem ser chamadas. A partir das definições IDL, é possível mapear objetos CORBA para uma linguagem de programação em particular ([OMG1995]).

3.4.1 *STUBS* E *SKELETONS*

São os *Stubs* e *Skeletons* que disponibilizam a interface da implementação do objeto (IDL), tanto para o objeto cliente, quanto para o próprio ORB, simplificando a chamada efetiva dos métodos.

Um *stub* fornece a interface para o objeto cliente, de forma que a chamada de métodos seja feita de forma simples, sem preocupações com detalhes da comunicação.

Um *skeleton* fornece a interface no lado servidor, para que o ORB possa fazer a chamada de métodos da implementação de uma forma mais simples e padronizada.

A transformação entre as definições IDL e a linguagem de programação destino é automatizada por um compilador IDL. O uso de um compilador reduz as possíveis inconsistências clientes *stubs* e servidores *skeletons*. Esta é uma forma estática de fazer as chamadas aos métodos dos objetos, porém, nem toda a linguagem tem o mapeamento direto para estas estruturas ([SCH1999]).

A existência de um *skeleton* não implica necessariamente na existência de um *stub* correspondente. Os clientes podem chamar objetos servidores via DII ([OMG1995]).

3.4.2 *DYNAMIC INVOCATION INTERFACE* (DII) E *DYNAMIC SKELETON INTERFACE* (DSI)

Uma interface também está disponível para que se permita fazer a construção dinâmica de chamadas de objetos, ou seja, que se chame uma rotina *stub* específica para uma operação em particular em um determinado objeto. Isto é chamado de *Dynamic Invocation Interface* (DII). Assim o cliente pode especificar o objeto a ser chamado, os métodos a serem executados e o conjunto de parâmetros para estes métodos, através de uma chamada ou seqüência de chamadas. O código do cliente precisa fornecer informações sobre os métodos a serem executados e os tipos de parâmetros a serem passados (talvez obtendo isto de um Repositório de Interfaces ou de outra rotina fonte). A forma de se fazer uma chamada dinâmica de interfaces pode variar bastante de uma linguagem de programação para outra ([OMG1995]).

Da mesma forma acontece com o lado servidor, que também permite uma chamada dinâmica de objetos, sem que se tenha a declaração estática de sua interface em um *skeleton*. Este procedimento caracteriza a construção de uma *Dynamic Skeleton Interface* (DSI). O código da implementação do servidor deve fornecer descrições de todos os parâmetros dos métodos que possui ao ORB e o ORB disponibiliza os valores de qualquer parâmetro de entrada para a execução de um método ([OMG1995]).

3.4.3 ADAPTADOR DE OBJETO (*OBJECT ADAPTER*)

O adaptador de objeto auxilia na comunicação entre o objeto servidor e o ORB, tanto para a ativação quanto para o recebimento de requisições. É o adaptador de objeto que faz a associação entre um servidor e o ORB ([SCH1999]).

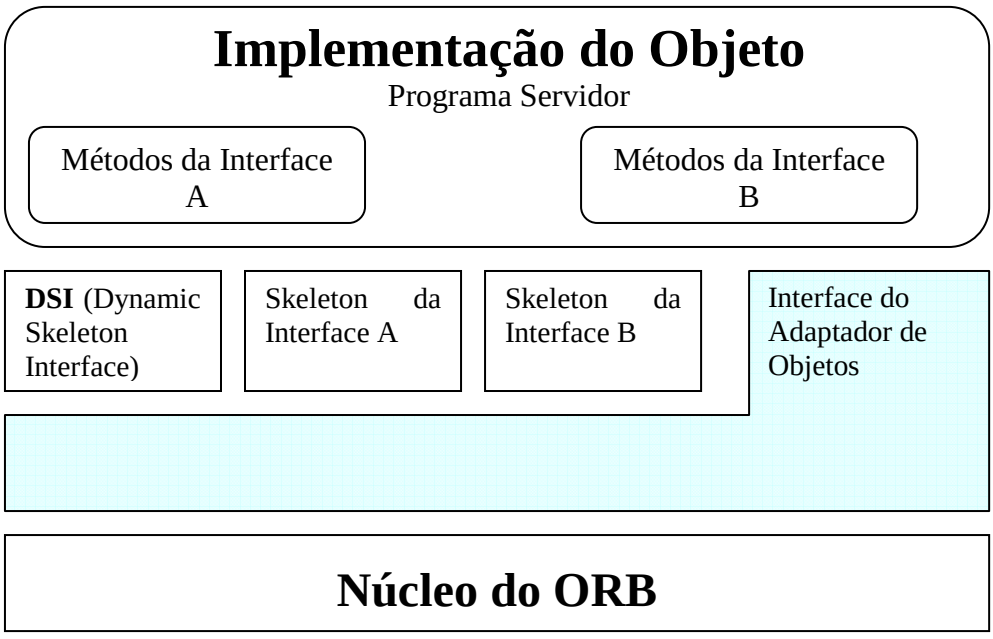
Vários serviços fornecidos pelo ORB são através do adaptador de objeto:

a) geração e interpretação de referências de objeto;
b) chamada de métodos;
c) segurança na comunicação;
d) ativação e desativação de objetos;
e) mapeamento de referencias de objeto para implementações;
f) registro de implementações.

A grande variação de objetos no que diz respeito a tempo de vida, políticas, estilos de implementação e outras propriedades, dificulta o núcleo do ORB para prover uma interface simples que seja comum e eficiente para todos os objetos. Assim através dos adaptadores de objeto é possível para o ORB objetivar grupos particulares de implementações de objeto que têm necessidades semelhantes com interfaces adaptadas para eles ([OMG1995]).

Pode-se verificar na Figura 5 que o ORB precisa do adaptador de objetos para acessar os *skeletons* e o DSI.

Figura 5 - Estrutura de um Adaptador de Objeto



3.4.4 REPOSITÓRIO DE INTERFACE (*INTERFACE REPOSITORY*)

O repositório de interface é um serviço que fornece objetos persistentes que representam a informação da IDL em uma forma disponível em tempo de execução. As informações do repositório de interface podem ser usadas pelo ORB para executar requisições. É usado para as requisições dinâmicas. Além disso, usando as informações do repositório de interface, é possível para um programa encontrar um objeto referente a uma interface que não era conhecido quando o programa foi compilado, contudo, podendo determinar quais operações são válidas no objeto e fazendo uma chamada nelas ([OMG1995]).

Além de seu papel no funcionamento do ORB, o repositório de interface é um lugar comum para armazenar informações adicionais associadas com interfaces a objetos de ORB. Por exemplo, depurar informações, bibliotecas *stub* e *skeletons*, rotinas para formatar ou procurar determinados tipos de objetos, etc., poderiam ser associados com o repositório de interface ([OMG1995]).

3.4.5 REPOSITÓRIO DE IMPLEMENTAÇÃO (*IMPLEMENTATION REPOSITORY*)

O repositório de implementação contém informações que permitem que o ORB localize e ative implementações de objetos. Embora muitas das informações do repositório de implementação sejam específicas para um ORB ou ambiente operacional, o repositório de implementação é o lugar convencional para registrar tais informações ([OMG1995]).

Geralmente, instalações de implementações e controle de políticas relacionadas à ativação e execução de implementações de objetos são feitos através de operações no repositório de implementação ([OMG1995]).

Além de seu papel no funcionamento do ORB, o repositório de implementação é um lugar comum para armazenar informações adicionais associadas com implementações de objetos ORB. Por exemplo, depurar informações, controle administrativo, alocação de recursos, segurança, etc., poderiam ser associados com o repositório de implementação ([OMG1995]).

3.4.6 PROTOCOLO GIOP (*GENERAL INTER-ORB PROTOCOL*) E IIOP (*INTERNET INTER-ORB PROTOCOL)*

O GIOP (*General Inter-ORB Protocol*) contém as especificações padronizadas da troca de mensagens entre objetos CORBA ([CAP1999]).

Para facilitar as requisições e prover a interoperabilidade entre os ORB´s, a especificação do CORBA 2.0 descreve o protocolo chamado *Internet Inter-ORB Protocol* (IIOP), que está sendo rapidamente aceito pelas empresas líderes mundiais na produção de softwares ([SIE1996]).

O IIOP, que é o GIOP sobre TCP/IP, é o único protocolo padronizado pela OMG para troca de mensagens remotamente ([CAP1999]).

O GIOP/IIOP foi desenvolvido para atingir os seguintes objetivos:

a) maior abrangência possível: o IIOP é baseado no TCP/IP, o protocolo mais usado e mais flexível disponível atualmente, definindo somente mínimas adições de camadas para troca de mensagens entre objetos CORBA;
b) simplicidade: o IIOP foi desenhado da maneira mais simples possível para permitir que não seja utilizado somente por pessoas especializadas;
c) escalabilidade: foi desenhado para suportar o tamanho da Internet de hoje, e do futuro;
d) baixo custo: os custos de implementação do protocolo são reduzidos ao máximo para que o desenvolvedor se preocupe com o ORB em si, não com o protocolo de comunicação;
e) generalidade: o GIOP foi desenhado para ser implementado em qualquer protocolo orientado a conexão confiável, não somente o TCP/IP ([CAP1999]).

3.4.7 CLIENTE E IMPLEMENTAÇÃO DO OBJETO (SERVIDOR)

Nesta estrutura, também se tem o papel do cliente propriamente dito e o servidor, que são, na verdade, os programas responsáveis pela execução da parte da aplicação referente ao negócio.

O cliente conhece apenas a estrutura lógica do objeto servidor de acordo com a sua interface e verifica o seu comportamento através das chamadas dos métodos.

Embora seja possível generalizar dizendo que um cliente é um programa ou um processo fazendo requisições a um objeto, é importante saber que é um cliente em relação a um determinado servidor. Por exemplo, um objeto servidor pode ser cliente de outros objetos servidores.

Tanto o cliente quanto o servidor podem ser desenvolvidos em várias linguagens de programação, fazendo a sua comunicação via IDL (com *stubs* e *skeletons*), ou via DII e DSI.

A implementação do objeto (ou servidor) provê a semântica do objeto definindo dados para a instância do objeto e definindo código para os métodos do objeto. Vários meios de se implementar os objetos podem ser suportados, por exemplo, servidores separados, bibliotecas, um programa por método, aplicação encapsulada, banco de dados OO, etc. Geralmente, as implementações de objetos não dependem do ORB, ou de como o cliente faz a chamada ([MAI1997]).

4 *DISTRIBUTED COMPONENT OBJECT MODEL* - DCOM

4.1 HISTÓRICO

DCOM tem suas raízes na tecnologia de objeto da Microsoft, que evoluiu durante a última década de DDE (*Dynamic Data Exchange*, uma forma de troca de mensagens entre programas Windows®), OLE (*Object Linking and Embedding*, pondo vínculos visuais entre programas dentro de uma aplicação), COM (*Component Object Model*, usado como a base para toda a ligação de objetos), e ActiveX (COM habilitado para a Internet).

Com o lançamento do Windows NT 4.0 no início de 1997, a Microsoft introduziu o DCOM *(Distributed Component Object Mode*l) que é uma implementação do COM contemplando características distribuídas de aplicações desenvolvidas em uma rede de computadores com Windows NT ([MAI1997]).

A evolução desta tecnologia tem um tema consistente: cada interação reduz a complexidade de construção de aplicações grandes enquanto fornece uma funcionalidade mais rica para o usuário. Isto pode diminuir os custos de desenvolvimento porque os programadores podem usar componentes pré-construídos e interfaces programadas, reduzindo muito a integração e testes exigidos quando há uma integração entre vários programadores ([MIC2000]).

A maioria dos programadores para Windows reconhece estes benefícios e usam a arquitetura ActiveX. Há mais de três milhões de programadores profissionais treinados em ActiveX e suas tecnologias – OLE, COM e DCOM – e centenas de empresas de software independentes que fornecem componentes prontos. Estes componentes podem ser utilizados por programadores que trabalham com Microsoft Visual Basic®, PowerBuilder, Micro Focus Visual Object COBOL, e outras ferramentas populares ([MIC2000]).

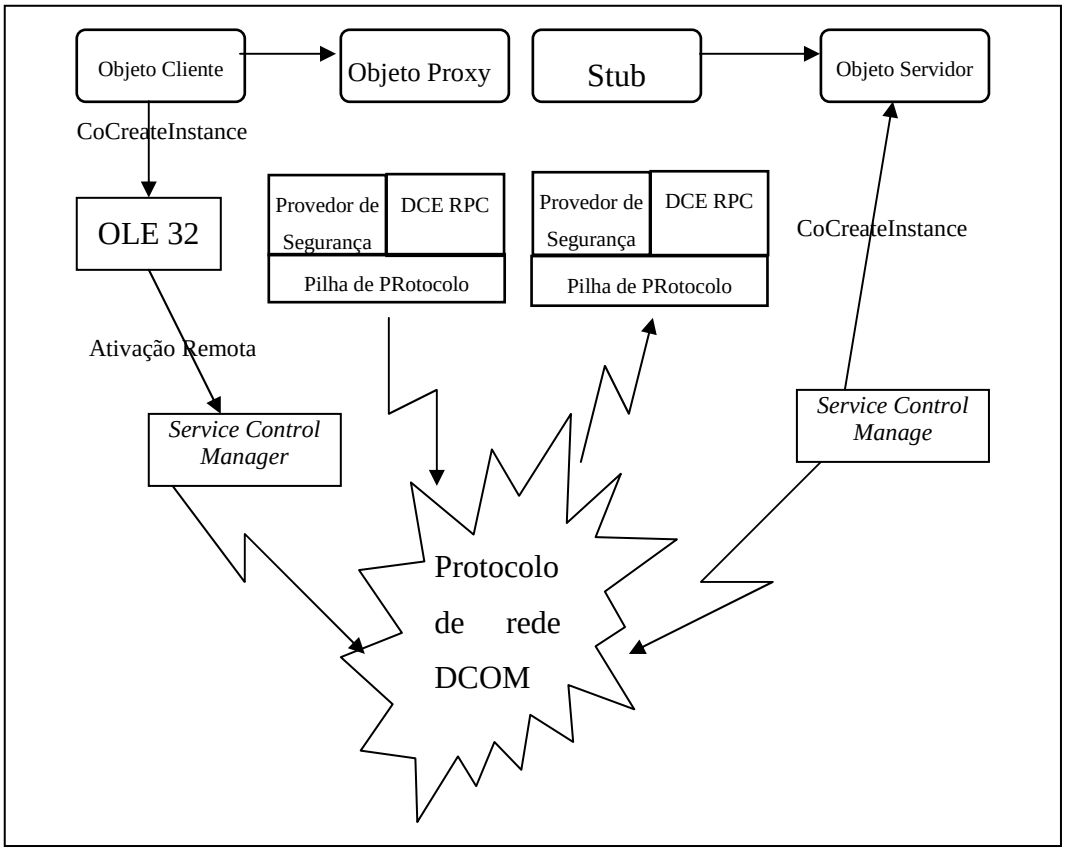
O uso de DCOM ainda se limita ao ambiente Windows (DCOM ainda não está disponível para Unix, Macintosh, OS/2 e outros ambientes) e as instalações que disponham de apenas Windows NT versão 4.0 ou superior ([MAI1997]).

4.2 VISÃO GERAL

Quando objetos cliente e servidor estão em computadores diferentes, o DCOM simplesmente substitui o processo de comunicação local entre processos por um protocolo de rede. Nem o cliente, nem o servidor sabem se a conexão é local ou via rede.

A figura 6 dá uma visão geral da arquitetura DCOM: O *run-time* do COM fornece serviços orientados a objetos para objetos clientes e servidores, usando RPC (*Remote Procedure Call*) e o provedor de segurança para gerar pacotes de rede padrão que obedecem ao protocolo padrão do DCOM.

Figura 6 – Arquitetura DCOM



Uma das principais partes do COM é um mecanismo para estabelecer conexões com componentes e criar novas instâncias de componentes. Este mecanismo é normalmente chamado de Mecanismo de Ativação.

Um dos requisitos mais básicos em um sistema de objetos distribuídos é a possibilidade de criar componentes. No mundo COM, classes de objetos são nomeados com um identificador global único (GUID – *global unique identifiers*). Quando GUID são utilizados para identificar uma determinada classe de objetos, são chamadas de Identificador de Classes (CLSID). Estes GUID não são nada mais que um grande inteiro (128 bits). Este tamanho diminui bastante a chance de duplicidades.

As bibliotecas COM verificam o arquivo binário (DLL ou Executável) no registro do sistema, criam o objeto e retornam um ponteiro para a interface de modo que se possa chamar todos os métodos e atributos disponíveis.

No DCOM, o mecanismo de criação de objetos do COM foi aprimorado para permitir a criação de objetos em outro computador. Para possibilitar a criação remota de objetos, as bibliotecas COM precisam saber o nome do servidor na rede. Uma vez conhecidos o nome do servidor e o identificador da classe, as bibliotecas COM chamam o *service control manager* (SCM) da máquina cliente, que se conecta com o SCM da máquina servidora, requisitando a criação deste objeto.

O DCOM tem dois principais mecanismos para que os clientes indiquem o nome do servidor remoto ao criar um objeto:

a) Com uma configuração fixa no registro do sistema ou no depósito de classes do DCOM;
b) Com um parâmetro explícito na chamada das funções **CoCreateInstanceEx**, **CoGetInstanceFromFile**, **CoGetInstanceFromStorage**, ou **CoGetClassObject**. Estas funções são disponibilizadas pelas bibliotecas COM para que seja possível fazer a criação de objetos.

O primeiro mecanismo é útil para manter a transparência de localização: clientes não precisam saber se um componente está executando local ou remotamente. Pelo fato de o nome do servidor remoto fazer parte das informações do componente servidor na máquina cliente, os componentes clientes não precisam se preocupar com a localização do servidor. Todos os clientes precisam saber apenas o identificador da classe a ser criada. Assim basta apenas chamar a função que efetua a chamada necessária às bibliotecas COM e estas, transparentemente, criam o objeto correto no servidor pré-configurado. Mesmo os clientes

COM desenvolvidos antes da existência do DCOM podem usar servidores remotos com este mecanismo.

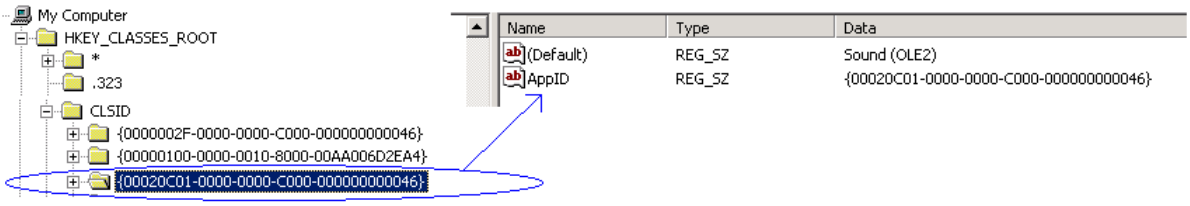
Isto se deve ao fato de que a configuração referente ao nome do servidor está, não no código do cliente, mas sim, no registro do sistema ou no depósito de classes do COM. Assim, quando o nome de uma máquina servidora ou de um componente servidor mudar, o que irá se alterar é o registro do sistema ou o depósito de classes do COM. O que não pode alterar é o identificador da classe, pois é ele que o cliente utiliza para se referenciar ao servidor.

O nome do servidor remoto está armazenado no registro do sistema sob uma nova chave em "HKEY_CLASSES_ROOT (HKCR)": [HKEY_CLASSES_ROOT\APPID\{<appid-guid>}].

O identificador de classe está em outro ponto do registro localizado em: [HKEY_CLASSES_ROOT\CLSID\{<clsid-guid>}].

A Figura 7 mostra a localização do identificador de classe no registro do Windows.

Figura 7 – Localização do identificador de classe no registro do sistema



4.3 EMPACOTANDO PARÂMETROS E OBJETOS - "MARSHALING"

4.3.1 CHAMADA REMOTA DE MÉTODOS

Quando um cliente precisa chamar um objeto em espaço de endereçamento diferente, os parâmetros do método a ser chamado precisam ser passados de alguma forma do cliente para o objeto servidor. O cliente coloca os parâmetros em uma pilha (na realidade, alguns parâmetros são passados diretamente através dos registradores da CPU, porém, para

simplificar esta apresentação, referencia-se apenas ao uso da pilha como forma de passagem de parâmetros). No caso de uma chamada direta a um objeto, este busca os parâmetros diretamente da pilha e retorna os valores de retorno também para ela ([MIC2000]).

Para chamadas remotas (mais especificamente quando o cliente e o servidor não compartilham a mesma pilha), qualquer código (normalmente bibliotecas COM) precisa ler todos os parâmetros da pilha e colocá-los em um buffer de memória podendo assim transmitir através da rede. O processo de leitura de parâmetros da pilha para um buffer de memória é chamado de *marshaling*. Isto não é muito simples, pois os parâmetros podem ser completamente diferentes – eles podem ser um ponteiro para um vetor ou um ponteiro para uma estrutura. Para que isso seja possível, o *marshaling* possui uma completa hierarquia para ponteiros de estruturas para todos os parâmetros, podendo assim restaurar no espaço do processo do objeto servidor ([MIC2000]).

Em contrapartida ao *marshaling*, está o processo de ler os dados dos parâmetros e montar uma pilha igual à pilha montada pelo processo cliente. Este processo é chamado de *unmarshaling*. Assim que a pilha está recriada, o objeto servidor pode ser chamado. Como feito na chamada, todos os valores de retorno e parâmetros de saída do processo servidor devem passar pelo processo de *marshaling*, enviados novamente ao processo cliente, passados pelo processo de *unmarshaling* e montar a pilha no cliente ([MIC2000]).

O COM fornece mecanismos sofisticados para os métodos de *marshaling* e *unmarshaling* de parâmetros, que montam na chamada remota de procedimentos (RPC - *remote procedure call*) uma infra-estrutura definida como parte do ambiente distribuído de comutação (DCE - *distributed computing environment*) padrão. O DCE e RPC definem uma representação padrão de dados, a *Network Data Representation* (NDR) para todos os tipos relevantes de dados. Dessa forma, para que o COM possa efetuar o *marshaling* e o *unmarshaling* dos parâmetros corretamente, ele precisa saber a interface completa do método, incluindo os tipos de dados, tipos de estruturas e tamanhos de vetores na lista de parâmetros. Esta descrição é possível, utilizando uma linguagem de definição de interface (IDL), o qual está embutido no DCE e RPC padrão IDL. Arquivos IDL podem ser compilados por qualquer compilador IDL. O compilador IDL gera um código fonte em C que contém os comandos para executar o *marshaling* e o *unmarshaling* para a interface descrita no arquivo IDL. O código do lado cliente é chamado de *proxy* enquanto o código do lado do objeto servidor é

chamado de *stub*. O *proxy* e o *stub* gerados pelo compilador IDL são objetos COM que serão carregados pelas bibliotecas COM quando necessário. Quando o COM precisa procurar uma combinação *proxy*/*stub* para uma interface em particular, ele simplesmente procura o identificador de interface (IID – Interface ID) abaixo da chave "HKEY_CLASSES_ROOT\Interfaces" no registro do sistema e lê a chave "ProxyStubClsid" ([MIC2000]).

4.3.2 MANIPULANDO PONTEIROS DE INTERFACE

Novos ponteiros para interface podem aparecer como resultado de uma chamada de ativação ***CoCreateInstance*** ou como um parâmetro de uma chamada de método. Para manipular estes dados de uma forma única, o COM tem um tipo de dado especial: o Ponteiro de Interface. O processo de *marshaling* e *unmarshaling* para este novo tipo de dado significa simplesmente criar um par *proxy*/*stub* capaz de manipular todos os métodos existentes na interface. O COM possui as formas padrão de fazer o tratamento do *proxy* e do *stub* na criação de um ponteiro para interface, porém é possível que o cliente faça alguma extensão ou até mesmo sobreponha as rotinas por alguma que seja mais adequada ao seu processo. A forma de sobrepor as rotinas para efetuar o processo de *marshaling e unmarshaling* é implementando as interfaces IMarshal e IStdMarshalInfo.

4.3.2.1 MANIPULAÇÃO PADRÃO

Se um objeto não implementa IMarshal nem IStdMarshalInfo, o COM usa o processo de *marshaling* de interface padrão, baseado no IID, da interface a ser manipulada. No lado servidor, o COM verifica o IID no registro do sistema e identifica o CLSID do processo *stub*. Então o COM cria o objeto *stub* e o inicializa com o ponteiro de interface a ser manipulado. O *stub* liga-se então ao ponteiro de interface (chamado de ***Iunknown***) e o utiliza para executar as chamadas de métodos que lhe serão feitas. Durante o processo de *marshaling*, o COM gera estas informações para que o *proxy* possa conectar-se ao servidor correto, ao objeto correto e à sua interface.

Quando acontece o processo de *unmarshaling* no lado cliente, o COM lê o IID para o servidor e verifica o CLSID para este IID. Então o COM cria o objeto *proxy* com o restante das informações recebidas e manipuladas no processo de *unmarshaling*. Finalmente, o COM

busca do *proxy* recém criado, o seu IID e retorna o ponteiro de interface resultante como o resultado do processo de *unmarshaling*.

4.3.2.2 MANIPULAÇÃO CUSTOMIZADA

Haverá situações em que o desenvolvedor queira ter o controle total sobre a comunicação entre o cliente e o servidor. O processo customizado de *marshaling* de uma interface permite ao desenvolvedor o controle total do fluxo do ponteiro de interface do servidor a ser manipulado. Esta é uma decisão de "tudo ou nada", pois um servidor pode ter seu processo de *marshaling* customizado para todas as interfaces ou para nenhuma delas. Um servidor indica se quer o processo de *marshaling* customizado, pela implementação da interface ***IMarshal***.

Antes de iniciar o processo de *marshaling* de um ponteiro de interface, o COM verifica no servidor a existência de um ***IMarshal***. Através do IMarshal, o servidor indica o CLSID do *proxy* customizado para ser usado para o servidor, como também os dados absolutos para ser usado na inicialização do *proxy*, quando o ponteiro de interface está passando pelo processo de *unmarshaling*.

4.4 GERENCIAMENTO DE CONEXÃO

O primeiro mecanismo para controlar o tempo de vida de um objeto servidor é o contador de referências, usando os métodos *AddRef* e *Release* da interface *IUnknown*. Quando um objeto cliente é terminado de forma anormal, o servidor fica sabendo, pois possui uma estrutura que efetua comandos *ping* numa freqüência pré-determinada, onde, se forem executados mais comandos ping sem resposta do que uma quantia também pré-definida, todas as referências do objeto servidor com aquela estação são eliminadas ([MIC2000]).

Muitas aplicações distribuídas necessitam de uma comunicação bi-direcional entre dois objetos. Um objeto servidor pode precisar notificar um cliente de um certo evento, ou então objetos servidores podem precisar enviar dados aos clientes na medida em que eles vão ficando disponíveis ([MIC2000]).

O modelo de programação simétrica do COM torna isto extremamente fácil tanto para o cliente, quanto para o servidor, por implementar este tipo de infra-estrutura. Um cliente

simplesmente passa um ponteiro de interface para o servidor e o servidor pode usar esse ponteiro para enviar notificações ou dados para este cliente ([MIC2000]).

Pontos de conexão padronizam a passagem de ponteiros de interface para um objeto servidor. Este mecanismo resolve também problemas de referência cíclica entre dois objetos. Com pontos de conexão, o objeto implementa a interface para registrar a comunicação de retorno para o cliente (*callback*), interface esta chamada de *IconectionPoint* ([MIC2000]).

Para aplicações distribuídas, a característica *multicast* dos pontos de conexão pode ser muito útil. Vários clientes podem registrar-se com o mesmo ponto de conexão a um servidor, e o ponto de conexão manda notificações simultâneas para todos os clientes ([MIC2000]).

5 DESENVOLVIMENTO DO PROTÓTIPO

O protótipo implementado neste trabalho é uma ferramenta de consulta a banco de dados desenvolvida em Delphi 5.0. Esta ferramenta é composta por dois objetos: um servidor e um cliente.

O objeto servidor tem as rotinas de acesso ao banco de dados através dos recursos disponibilizados pelo próprio Delphi (BDE). Ele opera recebendo comando SQL do objeto cliente, tratando este comando e executando no banco de dados conectado (a especificação do banco de dados utilizado torna-se pouco relevante para o objetivo deste trabalho) e retornando os dados ao cliente.

5.1 MODELAGEM DE CLASSES

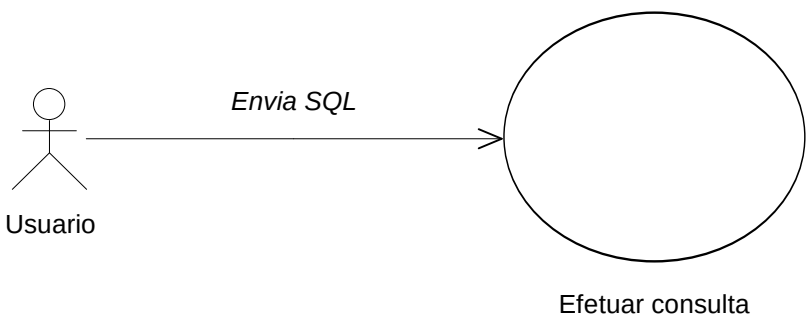
Para especificar este protótipo foi utilizada a linguagem de modelagem *Unified Modeling Language* (UML). A UML é uma linguagem padrão para especificar, visualizar, documentar e construir artefatos de um sistema e pode ser usada com todos os processos ao longo do ciclo de desenvolvimento e através de diferentes tecnologias de implementação.

A ferramenta utilizada para a modelagem UML foi o *Rational Rose* da empresa *Rational Rose Corp*.

5.2 CASO DE USO

Neste protótipo tem-se um caso de uso onde o usuário envia um comando SQL e recebe os dados resultantes. A figura 8 mostra o caso de uso “Efetuar consulta”.

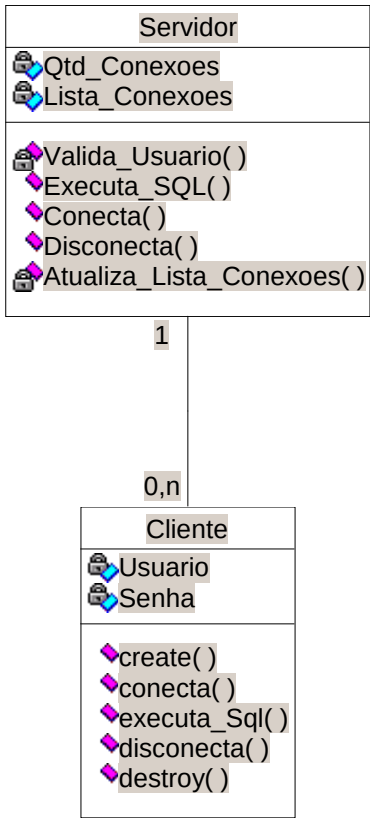
Figura 8 - Caso de uso do protótipo



5.3 DIAGRAMA DE CLASSES

Na figura 9 pode-se verificar o diagrama de classes onde estão descritos os atributos e as operações das classes: Servidor e Cliente.

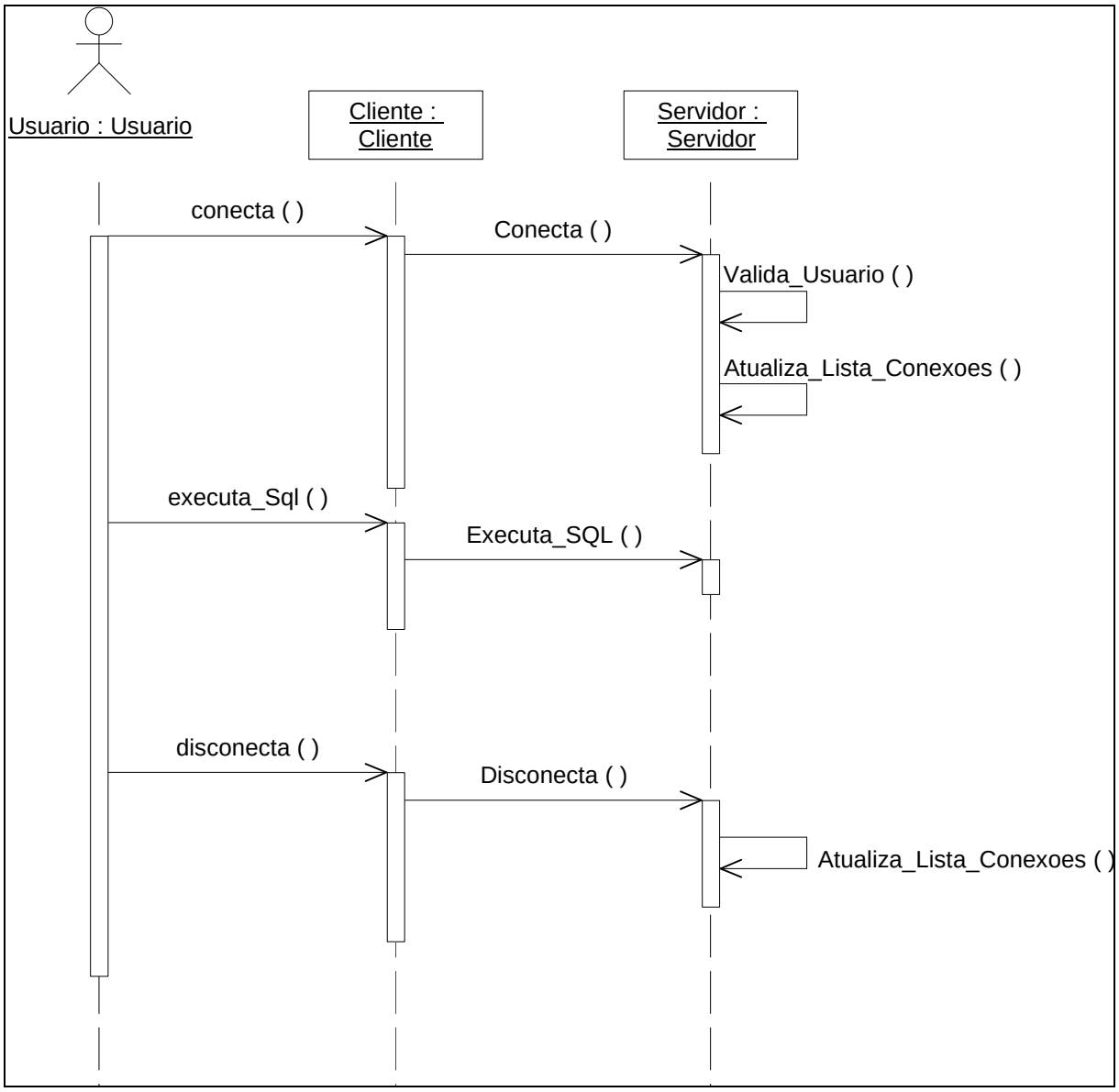
Figura 9 - Diagrama de classes do protótipo



5.4 DIAGRAMA DE SEQUÊNCIA

O diagrama de seqüência, ilustrado na figura 10, mostra a interação que existe entre o usuário e o objeto cliente, e a interação entre o objeto cliente e o servidor.

Figura 10 - Diagrama de seqüência do protótipo

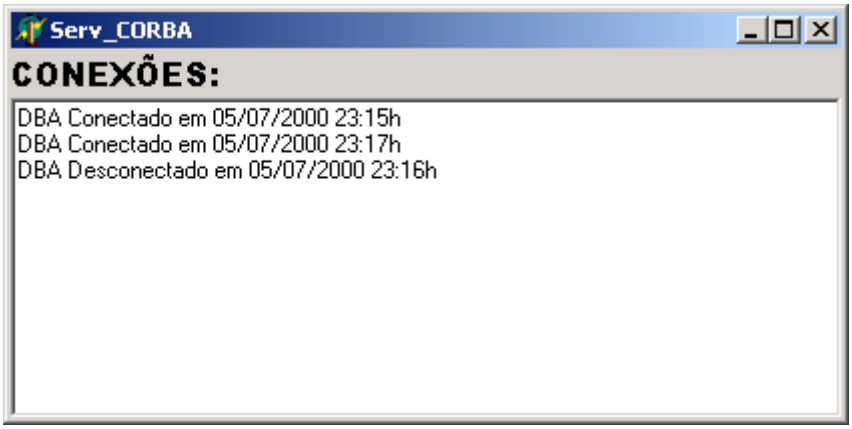


Este protótipo foi desenvolvido em duas versões, onde cada uma utiliza uma das tecnologias estudadas neste trabalho: DCOM e CORBA.

5.5 PROTÓTIPO

Como pode ser visto na figura 11, a aplicação servidora não tem a necessidade de interagir com o usuário, função esta da aplicação cliente. Neste lado, a aplicação apenas está executando para atender as requisições enviadas pelos objetos clientes.

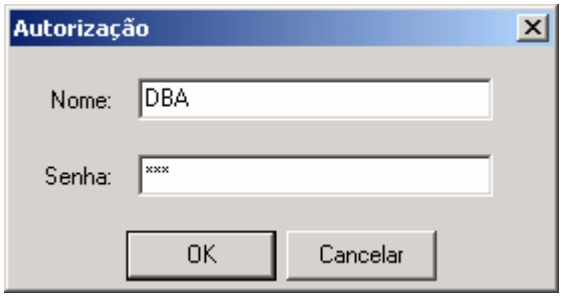
Figura 11 - Tela do programa servidor



O objeto cliente faz o papel de interface com o usuário e recebe o comando SQL que o usuário deseja executar, mandando este comando para o objeto servidor (após a sua conexão com o mesmo) e aguardando o retorno dos dados para serem mostrados ao usuário.

A figura 12 mostra a tela de autorização de usuário que está na aplicação cliente. Antes de habilitar a execução de qualquer comando SQL é necessário efetuar o *login* do usuário.

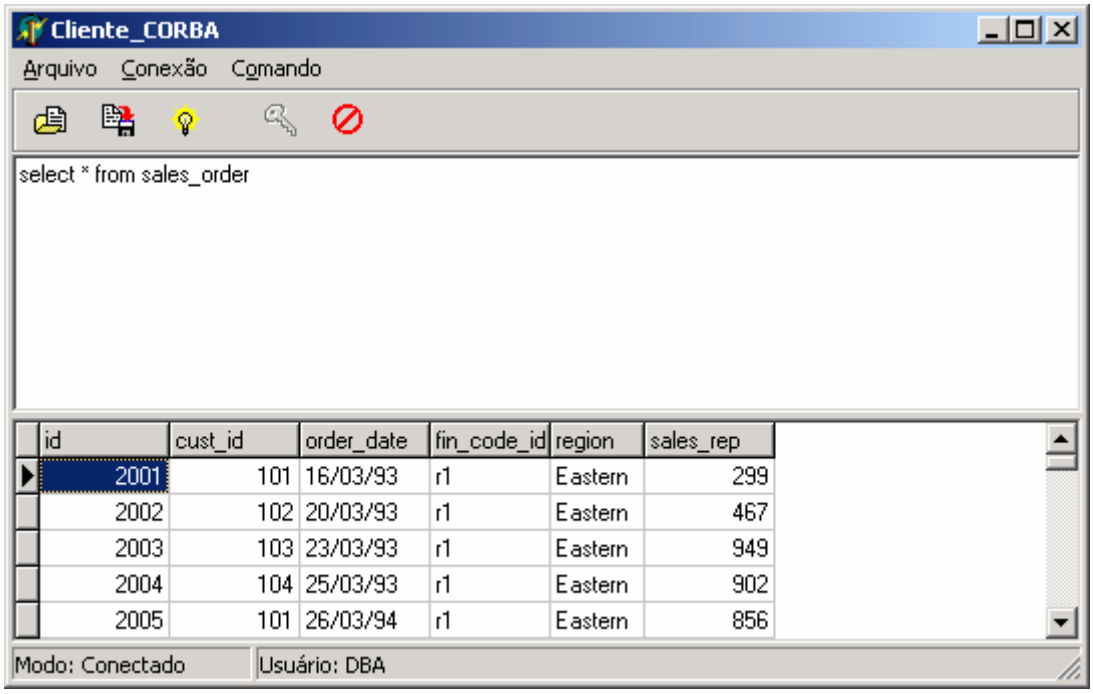
Figura 12 - Tela de autorização de usuário



Feita a autorização do usuário, a aplicação cliente está pronta para receber um comando SQL. A figura 13 mostra o resultado da execução de um comando SQL. O comando

SQL é enviado para o objeto servidor, que executa o comando na base conectada e devolve o resultado para que o cliente possa mostrar.

Figura 13 - Tela principal do programa cliente



5.6 DELPHI 5.0 – CARACTERÍSTICAS COM RELAÇÃO A CORBA E DCOM

Como o desenvolvimento do protótipo foi feito com a ferramenta Delphi 5.0, serão expostas a seguir, algumas características desta ferramenta com relação às duas tecnologias estudadas neste trabalho.

5.6.1 CORBA

No Delphi 5, o suporte a CORBA ainda é limitado. O principal fator limitante do suporte a CORBA do Delphi é que ele permite apenas chamadas de método através da execução de vínculo tardio do CORBA, chamada DII ([CAN2000]). Desta forma, não é possível efetuar as chamadas de métodos de forma direta utilizando a IDL dos objetos servidores.

É por isso que há interesse em um conversor de IDL para Pascal. Segundo anúncio da Inprise está havendo um trabalho para que seja disponibilizado o suporte a CORBA nativo, incluindo um conversor CORBA de IDL para Pascal. Pelo cronograma divulgado pela Inprise, esse suporte já deveria estar disponível ([CAN2000]).

Porém, quando se tem uma situação onde o servidor é escrito em Delphi, é possível utilizar a interface que é gerada em uma unit separada. Esta unit contém as interfaces e os objetos *stub* e *skeleton* do objeto implementado. Desta forma, o cliente pode utilizar esta unit para obter a interface desejada de forma estática, sem a necessidade da interface do servidor estar em um repositório de interface.

O suporte a CORBA do Delphi está baseado no ORB *VisiBroker* para C++ (Versão 3.3.2) com uma implementação especial (orbpas.dll) que expõe um subconjunto da funcionalidade do ORB para aplicações Delphi.

5.6.2 DCOM

O Delphi é totalmente compatível com a tecnologia DCOM. Quando é examinado o código fonte, o *Object Pascal* parece ser mais fácil de usar do que o C++ ou outras linguagens para escrita de objetos COM. Essa simplicidade deriva principalmente da incorporação de tipos de interface à linguagem *Object Pascal*. A propósito, as interfaces também são usadas de modo análogo para integrar a linguagem Java com a tecnologia COM na plataforma Windows ([CAN2000]).

5.6.3 CONSTRUÇÃO DO SERVIDOR

Para a construção de uma aplicação servidora de dados (na verdade, a camada do meio que faz a conexão com o banco de dados), é necessário usar um módulo de dados remoto (*Remote Data Module*) ou um dos módulos de dados remotos especializados para CORBA e MTS (este último não será discutido neste trabalho) ([CAN2000]).

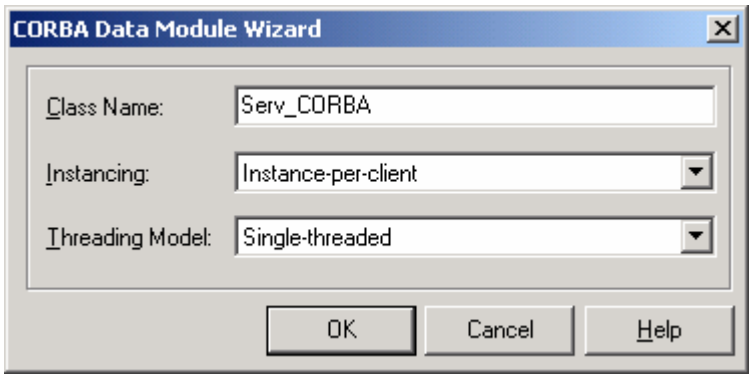
A aplicação servidora contém um ou mais componentes de acesso ao banco de dados (*TQuery, TTable*, etc.) de acordo com o objetivo.

O único componente específico necessário no lado do servidor é o *DataSetProvider*. É necessário um *DataSetProvider* para cada tabela ou consulta do servidor que se queira tornar disponível no lado cliente ([CAN2000]).

5.6.3.1 CORBA

No Delphi é possível construir um servidor CORBA simples, baseado em um módulo de dados. Quando é selecionada a opção *CORBA Data Module* na guia *Multitier* do *Object Repository*, o Delphi apresenta a caixa de diálogo *CORBA Data Module Wizard*, mostrado na figura 14 ([CAN2000]).

Figura 14 - Wizard do Delphi que permite a construção de um módulo de dados remoto com suporte a CORBA



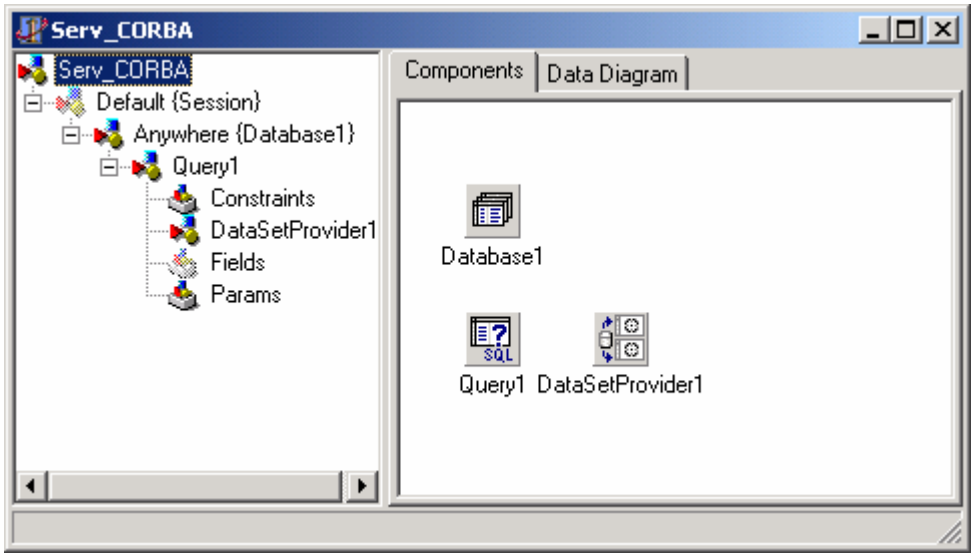
Nesse assistente, é possível especificar os modelos de instanciação e processamentos ([CAN2000]):

a) a instanciação *per-client* indica que uma nova instância do módulo de dados é criada para cada conexão;
b) a instanciação *shared* indica que uma única instância do módulo de dados manipula todas as solicitações de vários clientes.

Em um servidor, cada módulo de dados recebe apenas uma solicitação de cliente por vez, de modo que os dados da instância estão seguros contra possíveis conflitos. Em comparação, em um servidor com múltiplos processamentos, o cliente pode enviar várias solicitações simultâneas e isso impõe algum cuidado extra para o desenvolvedor ([CAN2000]).

Depois de terminado o *Wizard*, pode-se incluir os componentes de acesso ao banco de dados e também o(s) *DataSetProvider*(*s*) correspondente(s). A figura 15 mostra como ficou o módulo de dados CORBA implementado neste protótipo de uma forma totalmente visual.

Figura 15 – Módulo de dados CORBA gerado de forma visual



No quadro 1 é possível observar parte do código gerado pelo processo descrito anteriormente. É neste código que o objeto servidor é implementado. Pode-se observar os métodos *Login* e *Executa_SQL* que fazem a consistência de usuários e a execução do SQL no banco de dados respectivamente.

Quadro 1 - Interface do objeto servidor CORBA

```
unit UServ_CORBA;

interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs, ComObj, VCLCom,
StdVcl, DataBkr, CorbaRdm, CorbaObj, PServ_CORBA_TLB, Provider, Db, DBTables;
type
  TServ_CORBA = class(TCorbaDataModule, IServ_CORBA)
    Query1: TQuery;
    Database1: TDatabase;
    DataSetProvider1: TDataSetProvider;
  private
    { Private declarations }
  public
    { Public declarations }
  protected
    procedure Login(const Nome, Senha: WideString); safecall; //consistência de usuário
    procedure Executa_SQL(const SQL: WideString); safecall;   //executa o comando SQL do
  end;                                                        //cliente
var
  Serv_CORBA: TServ_CORBA;

implementation
uses CorbInit, CorbaVcl;

procedure TServ_CORBA.Login(const Nome, Senha: WideString);
begin
if  (NOME <> 'dba') or (SENHA <> 'sql') then  //implementar cadastro de usuários e senhas
  raise Exception.Create ('Usuário ou senha inválidos');  //exceção mostrada o cliente.
end;

procedure TServ_CORBA.Executa_SQL(const SQL: WideString);
begin
Query1.active := false;  //Desativa o componente Query
Query1.SQL.Text := SQL;  //Atribui o novo comando SQL enviado pelo objeto cliente
Query1.Active:= true;    //Ativa o componente query, executando o comando no banco
end;
```

Além da unidade com o código do objeto, outra unidade é gerada para fornecer as interfaces do objeto para que o Delphi possa usá-las sem a necessidade buscá-las de um repositório de interface.

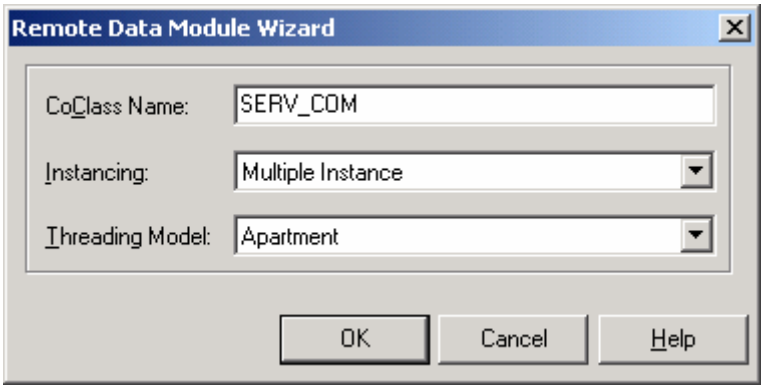
Para que a execução do servidor seja feita com sucesso, é necessário que se tenha no ar o *Visibroker Smart Agent* que serve para registrar a aplicação servidora e todos os seus objetos implementados e estabelecer a comunicação com um objeto cliente que quer fazer uma requisição a um objeto implementado neste servidor.

5.6.3.2 DCOM

A construção de um servidor COM para fornecer dados se dá de forma bastante simples, assim como acontece com o CORBA.

O servidor é baseado em um módulo de dados que pode ser criado selecionando a opção *Remote Data Module* na guia *Multitier* do *Object Repository*. Então o Delphi apresenta a caixa de diálogo *Remote Data Module Wizard*, onde são pedidas algumas informações sobre o objeto a ser criado. A figura 16 mostra como estas informações são pedidas.

Figura 16 - *Wizard* do Delphi que permite a criação de um módulo de dados remoto

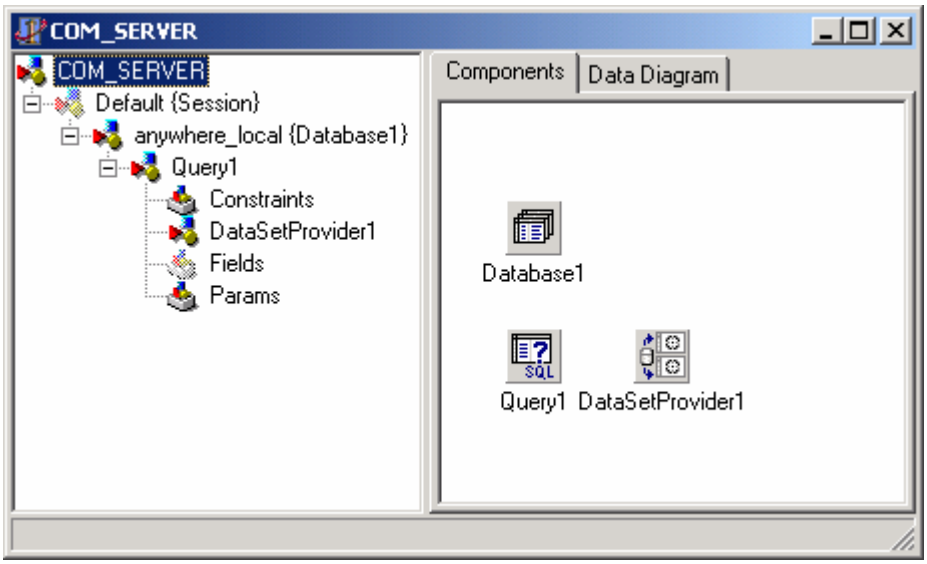


Além do nome do objeto, o *wizard* pede informações quanto à forma de instanciação e o suporte a *thread*.

Ao terminar o *wizard*, tem-se o módulo de dados para que se possa adicionar os componentes de acesso ao banco e também o componente que irá disponibilizar os dados a

um cliente COM. A figura 17 mostra o módulo de dados já com os componentes utilizados para o protótipo.

Figura 17 - Módulo de dados do servidor DCOM



Com este módulo de dados, o Delphi gera automaticamente o objeto em uma unit e define as interfaces necessárias para o funcionamento do suporte COM em outra. O quadro 2 mostra o código do objeto servidor depois de implementados os métodos usados no protótipo.

Quadro 2 - Código do objeto servidor DCOM

```
type
  TCOM_SERVER = class(TRemoteDataModule, ICOM_SERVER)
    Query1: TQuery;
    Database1: TDatabase;
    DataSetProvider1: TDataSetProvider;
  private
    { Private declarations }
  protected
    class procedure UpdateRegistry(Register: Boolean; const ClassID, ProgID: string);
override;
    procedure LOGIN(const NOME, SENHA: WideString); safecall;
    procedure Executa_SQL(const SQL: WideString); safecall;
  public
    { Public declarations }
  end;

Implementation
class procedure TCOM_SERVER.UpdateRegistry(Register: Boolean; const ClassID, ProgID: string);
begin
  if Register then
  begin
    inherited UpdateRegistry(Register, ClassID, ProgID);
    EnableSocketTransport(ClassID);
    EnableWebTransport(ClassID);
  end else
  begin
    DisableSocketTransport(ClassID);
    DisableWebTransport(ClassID);
    inherited UpdateRegistry(Register, ClassID, ProgID);
  end;
end;

procedure TCOM_SERVER.LOGIN(const NOME, SENHA: WideString);
begin
if  (NOME <> 'dba') or (SENHA <> 'sql') then
  raise Exception.Create ('Usuário ou senha inválidos');
end;

procedure TCOM_SERVER.Executa_SQL(const SQL: WideString);
begin
Query1.Close;
Query1.sql.Text := SQL;
Query1.Open;
end;
```

Note-se que, além dos métodos implementados para o protótipo, o objeto contém um método *UpdateRegistry*, que é chamado na execução do servidor. Este método faz com que ele seja registrado no registro do Windows onde ele está rodando. Isto significa que, para que um objeto possa encontrar este servidor, tanto na rede quanto em um ambiente local, o servidor deve ter sido executado pelo menos uma vez. Esta é a forma de “instalar” um servidor DCOM. Após este processo, as bibliotecas COM do windows se encarregam de localizar e instanciar o servidor quando algum cliente necessitar.

Para a execução do servidor, não é necessário fazer nenhuma configuração, desde que se tenha os recursos COM ou COM+ instalados no Windows.

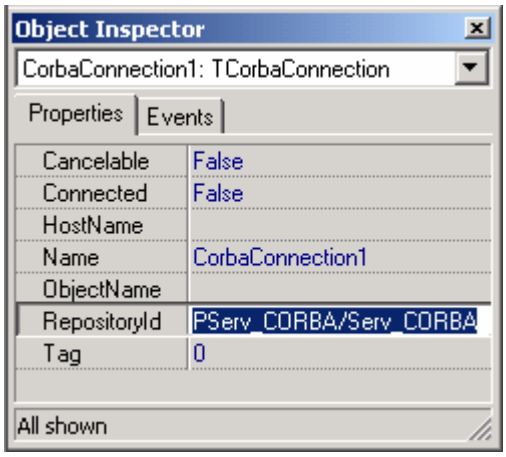
5.6.4 CONSTRUÇÃO DO CLIENTE

Para que a conexão entre a aplicação cliente e o módulo de dados remoto do servidor seja possível, foram utilizados componentes específicos para cada tipo de conexão, conforme será descrito nos capítulos 5.5.4.1 e 5.5.4.2. Para tornar os dados disponíveis para o usuário, foi utilizado um componente *ClientDataSet* que, juntamente com os componentes de conexão se comunica com o *DataSetProvider* da aplicação servidora. Este *ClientDataSet* fornecerá os dados para um componente *DBGrid* para que se possa visualizar o retorno do comando SQL enviado.

5.6.4.1 CORBA

No cliente CORBA, o componente Delphi utilizado foi o *CorbaConnection*. Este componente tem a função de localizar o objeto servidor, comunicando-se com o ORB. Para que se possa estabelecer a conexão, a aplicação servidora deve estar sendo executada. Assim com a propriedade *RepositoryId* preenchida com o nome do executável e do objeto servidor, a comunicação pode ser estabelecida alterando a propriedade *connected* para *true*. A figura 18 mostra como é informado o servidor a ser conectado.

Figura 18 - Propriedades do componente CorbaConnection



É através deste objeto que se busca a referência para a interface do objeto servidor, possibilitando assim a chamada dos métodos implementados. Se a interface do objeto servidor encontra-se em um repositório de interfaces, é possível utilizá-la diretamente através da propriedade *AppServer* pois a validação dos métodos se dará durante as chamadas. Caso contrário, faz-se um *Cast* para o tipo de interface que se tem no código fonte da unit que contém as interfaces do servidor e que foi utilizada no cliente (forma esta utilizada no protótipo).

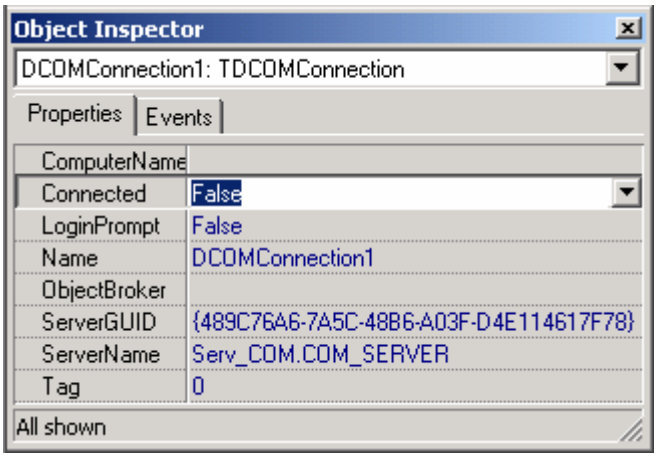
5.6.4.2 DCOM

A criação de um cliente DCOM no Delphi se torna uma tarefa simples para uma aplicação como a deste protótipo. Além do componente *client dataset* necessário para a comunicação com o módulo de dados, é necessária a utilização de um componente *DCOMConnection*, que irá estabelecer a conexão com o *DataSetProvider* do lado Servidor. É através do *DCOMConnection* que se pode obter a interface do servidor DCOM para a chamada dos métodos implementados. Esta interface está disponível através da propriedade *AppServer* do componente DCOMConnection. Por trás deste processo, estão as bibliotecas COM fazendo todo o trabalho de localização de busca da interface, seja em um mesmo computador ou em um servidor.

Na figura 19 é possível observar as propriedades do componente DCOMConnection onde é possível especificar o servidor que se quer conectar de forma bastante simples. Se o servidor já estiver registrado no registro do windows, ele irá aparecer em uma lista, já em

tempo de desenvolvimento na propriedade *ServerName*. Caso não se tenha o servidor registrado, basta incluir o seu identificador na propriedade *ServerGUID*. Já em tempo de desenvolvimento é possível até ativar o servidor alterando a propriedade *Connected* para *True*.

Figura 19 - Propriedades do componente DCOMConnection



6 COMPARAÇÃO

A Tabela 1 abaixo mostra as principais características das tecnologias DCOM e CORBA, de forma a identificar as diferenças entre elas.

Tabela 1 - Comparação das principais características de CORBA e DCOM

	DCOM	CORBA
Fornecedor	Microsoft	*Object Management Group* (OMG)
Plataforma	Windows NT 4.0 ou superior Windows 95 c/ I.E. 5 Windows 98	Independente
Nome para mapeamento de objetos	Registro do sistema	*Implementation repository*
Localização de objetos	*Service Control Manager* (SCM)	*Object Request Broker* (ORB)
Ativação de objetos	SCM	*Object Adapter* (Adaptador de Objeto)
Informações sobre os métodos	*Type Lybrary*	*Interface Repository*
Chamada de métodos	Via *Remote Procedure Call* (RPC)	Via ORB
Nome do lado do cliente	Proxy	Stub/Proxy
Nome do lado servidor	Stub	Skeleton
Protocolo de comunicação	DCOM	GIOP/IIOP

Com a tabela 1, fica bastante visível a diferença entre as duas tecnologias, no que diz respeito ao fator **multi-plataforma**:

a) A tecnologia DCOM utiliza-se de serviços específicos dos sistemas operacionais da Microsoft para que possa localizar (SCM), ativar (SCM) e estabelecer a comunicação (RPC) com o objeto servidor. Não se pode ignorar a influência da Microsoft em praticamente todas as áreas da computação, e não se pode ignorar o fato de que a presença de um modelo de objetos distribuídos embutido no Windows NT promove ainda mais esta área ([MAI1997]).

b) O CORBA, por outro lado já foi projetado com um dos principais objetivos de ser independente de plataforma. Assim, todos os recursos necessários para que haja a comunicação entre um cliente e uma implementação de objeto foram desenvolvidos de forma a não dependerem de nenhum recurso específico de um sistema operacional específico ou de algum hardware.

Outra característica bastante clara na comparação, é a de que os resultados finais são bastante parecidos, onde se obtém um **alto nível de abstração** com relação à forma de comunicação entre os objetos cliente e servidor:

a) o cliente DCOM obtém um ponteiro para a interface do objeto servidor, podendo assim efetuar as chamadas de métodos de forma direta, deixando toda o tratamento de comunicação para as bibliotecas COM e para os serviços do sistema operacional. Essas interfaces são disponibilizadas no delphi através da *type lybrary* gerada em uma unidade separada.
b) o cliente CORBA também consegue um alto nível de abstração na chamada de métodos, por obter uma referência do objeto servidor, onde estão disponíveis todos métodos fornecidos por ele, deixando todo o tratamento da comunicação para ORB. Esta obtenção de interface é possível através do *interface repository*, que é uma aplicação que contém as informações dos objetos CORBA rodando no momento e atende as chamadas feitas por objetos cliente.

As estruturas DCOM e CORBA provêem um tipo de comunicação cliente-servidor. Para requisitar um serviço, um cliente invoca um método implementado por um objeto remoto que atua como o servidor no modelo de cliente-servidor. O serviço fornecido pelo servidor é encapsulado como um objeto e a interface é descrita em uma linguagem de definição de interface (IDL). As interfaces definidas em um arquivo IDL funcionam como um “contato” entre um servidor e seus clientes. Clientes interagem com um servidor invocando métodos descritos na IDL. A real implementação do objeto é escondida do cliente. Algumas características de programação orientada a objetos estão presentes ao nível de IDL, como encapsulamento de dados, polimorfismo e herança simples. CORBA também suporta herança múltipla ao nível de IDL, o que não acontece no DCOM. Ao invés disso, o DCOM utiliza-se de interfaces múltiplas de um objeto para alcançar um propósito semelhante. IDL CORBA também pode especificar exceções.

Segundo [CAP1999], CORBA possui implementações que estão longe no ciclo de maturidade, enquanto DCOM está começando. Isso não quer dizer que DCOM não está sendo aprovado, pois muitas aplicações estão sendo construídas com sucesso utilizando essa tecnologia. Existem inclusive diversas *bridges* entre DCOM e CORBA, disponibilizando

objetos Windows para implementações CORBA, deixando claro que as tecnologias podem trabalhar juntas.

6.1 PERFORMANCE

Outro experimento efetuado neste trabalho foi o de performance. Será demonstrado a seguir de que forma os testes foram efetuados.

Para os testes de performance, foi utilizada a base de demonstração do banco de dados Sybase Sql Anywhere 5.0.

Executou-se um comando SQL com *join* entre as tabelas *sales_order* (pedidos) e *sales_order_items* (itens de pedido), resultando um total de 69.111 registros. A descrição das tabelas pode ser conferida nas tabelas 2 e 3.

Tabela 2 - Tabela de Pedidos usada nos testes

SALES_ORDER	
CAMPO	TIPO
ID	Integer
Cust_ID	Integer
Order_date	Date
Fin_code_id	Char (2)
Region	Char (7)
Sales_rep	Integer

Tabela 3 - Tabela de Itens de pedidos usada nos testes

SALES_ORDER_ITEMS	
CAMPO	TIPO
ID	Integer
Line_ID	Smallint
Prod_ID	Integer
Quantity	Integer
Ship_date	Date

O comando SQL, que segue no quadro3, foi executado por cinco vezes para cada tecnologia e para cada situação, com tempos uniformes entre cada uma, sem que outro comando SQL fosse executado nesse período.

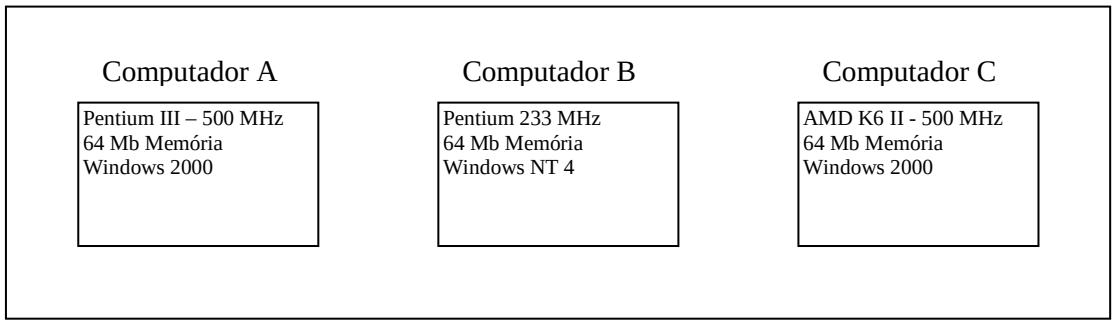
Quadro 3 - Comando SQL executado no teste de performance

```
SELECT A.*, B.* FROM SALES_ORDER A, SALES_ORDER_ITEMS B
WHERE A.ID = B.ID
```

Vale observar que a implementação CORBA deste trabalho utilizou-se do mapeamento de interface de forma estática, o que somente é possível quando o servidor é desenvolvido também em Delphi.

Para estes testes utilizou-se um ambiente com três computadores que serão referenciados como Computador A, Computador B e Computador C. Na figura 20 pode-se ter uma breve descrição de cada computador.

Figura 20 - Computadores envolvidos nos testes



Com estes três computadores foi possível efetuar os testes nas quatro situações possíveis em um ambiente de três camadas descritas a seguir.

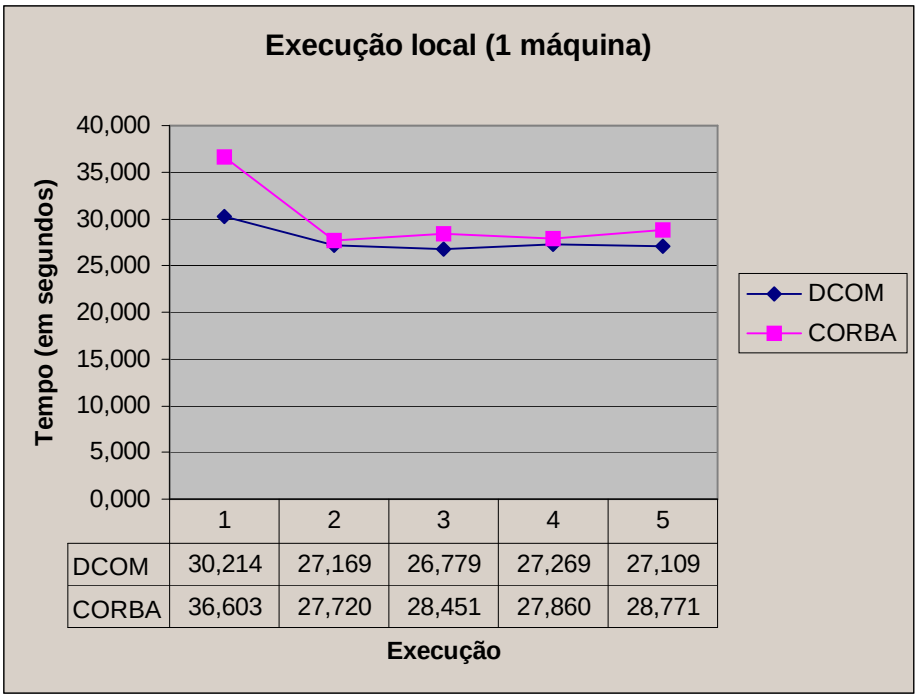
É importante observar que, os tempos da primeira execução de cada testes são consideravelmente superiores que o restante. Isto se deve ao fato de que a conexão com o banco de dados acontece nesse momento.

6.1.1 EXECUÇÃO LOCAL

Esta situação consiste na existência do banco de dados, da aplicação servidora e da aplicação cliente em um único computador.

Neste caso foi utilizado o Computador A. Os resultados desta execução podem ser vistos no Quadro 4 que demonstra uma pequena vantagem para o DCOM.

Quadro 4 - Demonstrativo da execução local

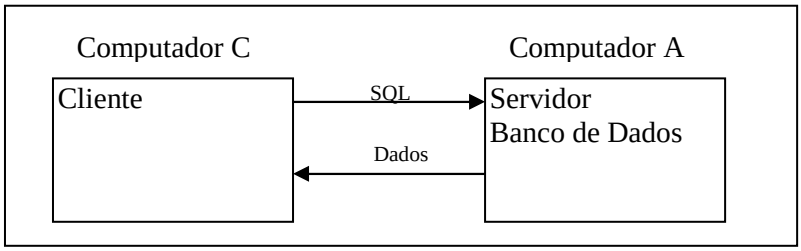


	1	2	3	4	5
DCOM	30,214	27,169	26,779	27,269	27,109
CORBA	36,603	27,720	28,451	27,860	28,771

6.1.2 CLIENTE REMOTO

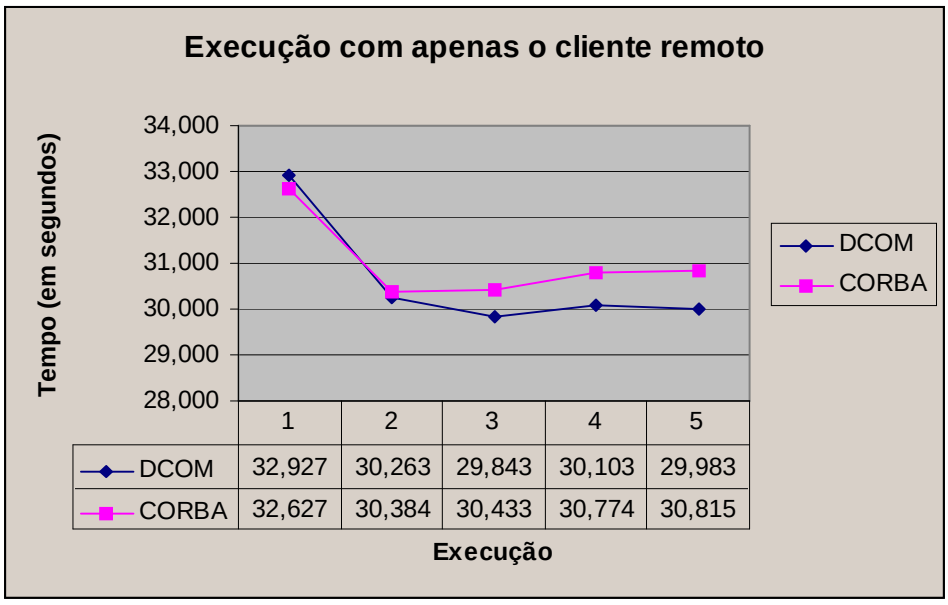
Esta situação consiste na presença do banco de dados e aplicação servidora em um computador, e a aplicação cliente em outro fazendo acesso remoto. Esta situação foi testada com os Computadores A e C. A figura 21 mostra a distribuição dos aplicativos neste teste.

Figura 21 - Estrutura com apenas o cliente remoto



Pode – se ver no quadro 5 o desempenho de cada tecnologia numa situação onde a diferença está na comunicação remota dos objetos clientes e servidores.

Quadro 5 - Demonstrativo da execução apenas com o cliente remoto



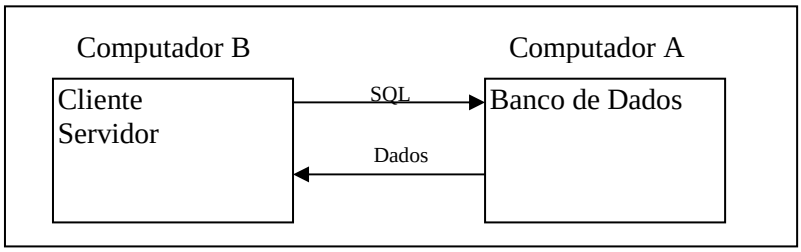
	1	2	3	4	5
DCOM	32,927	30,263	29,843	30,103	29,983
CORBA	32,627	30,384	30,433	30,774	30,815

É possível verificar no quadro 5 que, com exceção da primeira execução, o DCOM foi um pouco mais rápido que o CORBA.

6.1.3 BANCO DE DADOS REMOTO

Esta situação caracteriza-se em ter-se apenas o banco de dados sendo acessado remotamente e as aplicações cliente e servidora em um mesmo computador. Este teste foi feito utilizando os computadores A e B, conforme mostrado na figura 22.

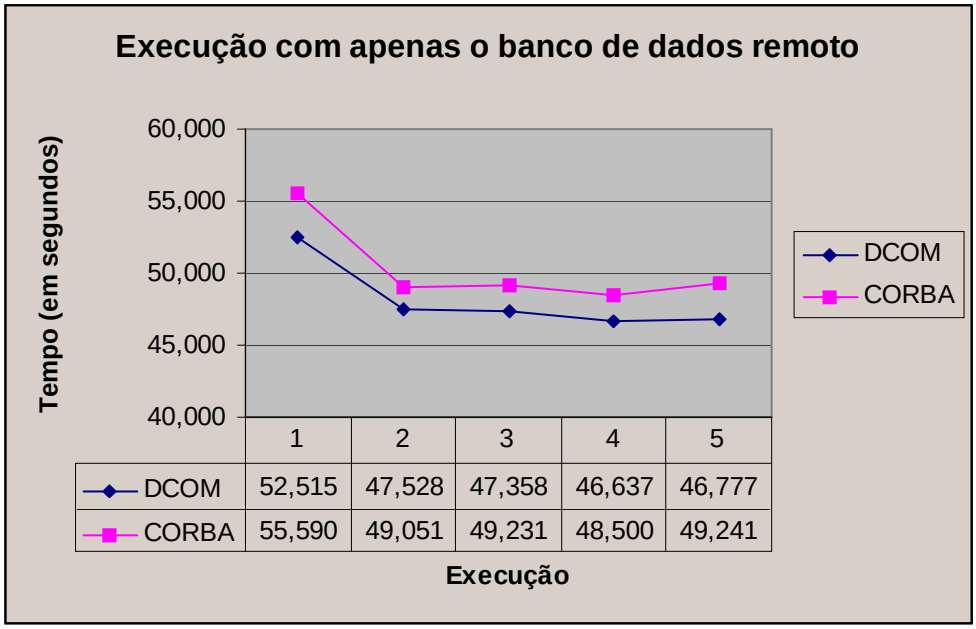
Figura 22 - Estrutura com apenas o banco de dados remoto



Esta estrutura faz com que toda a comunicação entre os objetos cliente e servidor sejam feitas localmente (no computador B).

O quadro 6 mostra o comportamento de cada tecnologia com esta estrutura.

Quadro 6 – Demonstrativo da execução com apenas o banco de dados remoto



	1	2	3	4	5
DCOM	52,515	47,528	47,358	46,637	46,777
CORBA	55,590	49,051	49,231	48,500	49,241

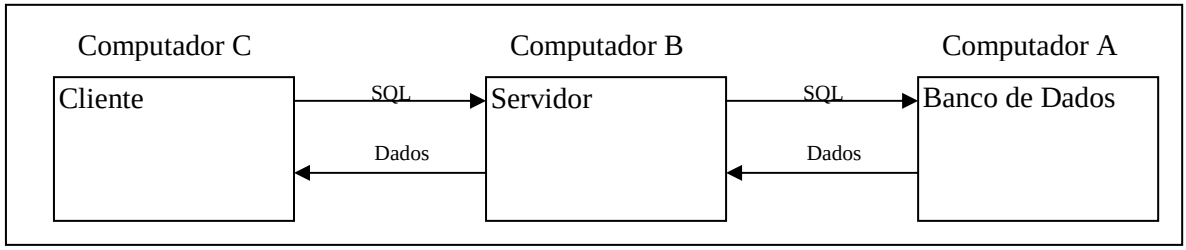
Observa-se no quadro 6 uma superioridade do DCOM com relação ao CORBA.

O fato da diferença de tempos ser mais perceptíveis se deve a situação onde o computador B tem uma configuração mais antiga (por ser um Pentium 233) que os outros, o que pode estar realçando a pequena diferença entre as tecnologias. Observa-se também que os tempos de execução com o servidor rodando no computador B são relativamente mais altos, devido à capacidade de processamento deste computador ser menor. Este é um indicador de que é importante o servidor estar sendo executado em um computador mais “potente” para se conseguir uma melhor performance.

6.1.4 TRÊS CAMADAS REMOTAS

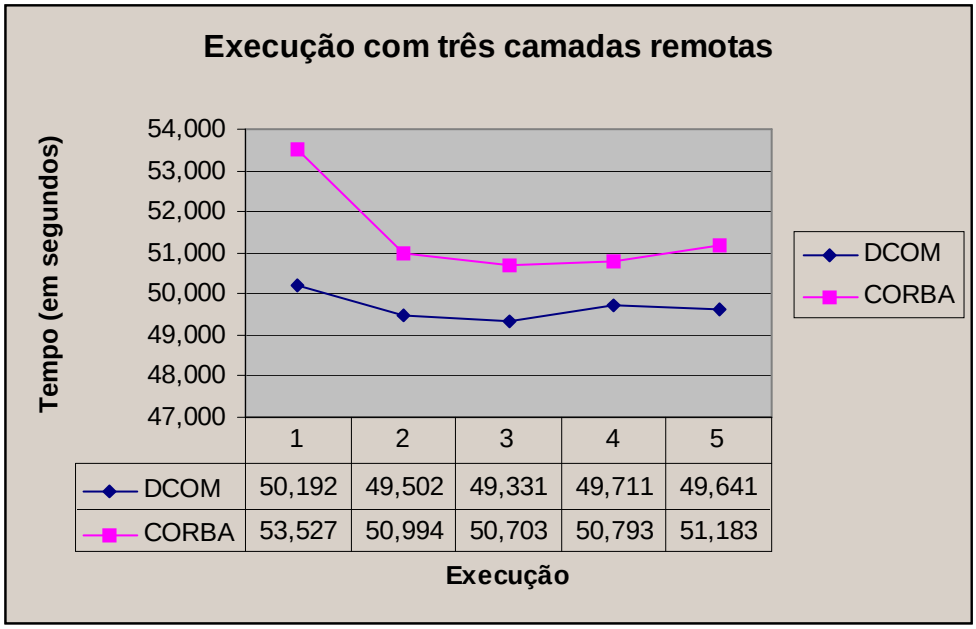
Nesta execução foram utilizados os três computadores da forma demonstrada na figura 23, onde se tem o computador C com a aplicação cliente acessando a aplicação servidora no computador B, que acessa o banco de dados no computador A.

Figura 23 - Estrutura de três camadas remotas



A situação demonstrada na figura 23 mostra que toda a comunicação, tanto entre os objetos cliente e servidor ou entre o servidor e o banco de dados, é feita remotamente. Pode-se verificar no quadro 7 que o DCOM ainda é mais rápido que o CORBA.

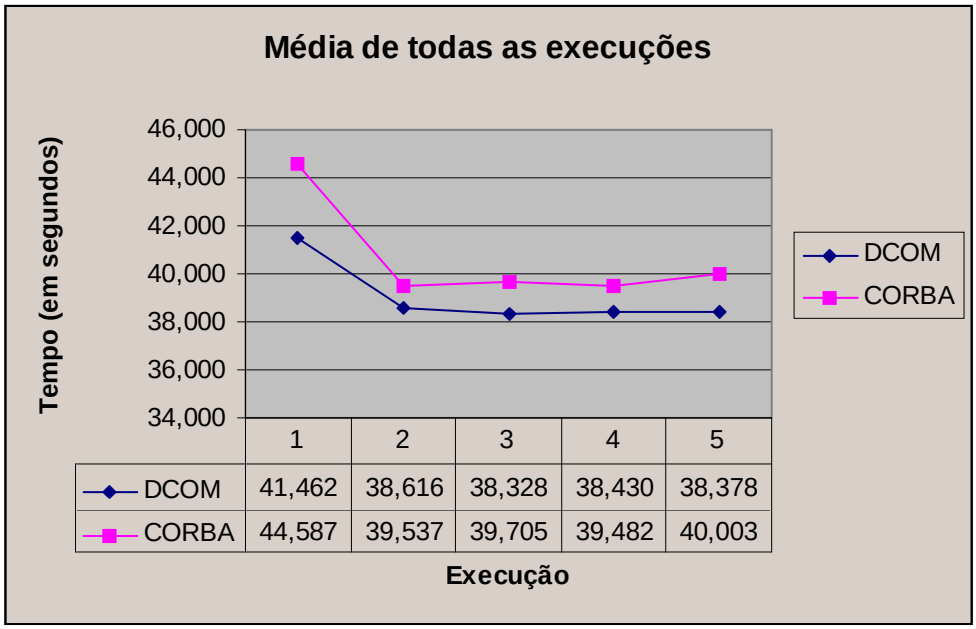
Quadro 7 - Execução com três camadas remotas



6.1.5 MÉDIA GERAL

Como forma de conclusão dos testes de performance foi feita uma análise geral das situações demonstradas anteriormente tirando uma média para cada tecnologia. O quadro 8 mostra o resultado geral, onde se confirma a superioridade do DCOM na questão performance.

Quadro 8 - Média geral da execução nas quatro situações



	1	2	3	4	5
DCOM	41,462	38,616	38,328	38,430	38,378
CORBA	44,587	39,537	39,705	39,482	40,003

7 CONSIDERAÇÕES FINAIS

Este capítulo irá apresentar as conclusões com relação ao trabalho desenvolvido como também as dificuldades encontradas e sugestões para trabalhos futuros nesta área.

7.1 CONCLUSÕES

A tecnologia de objetos distribuídos chegou a um ponto em que parece apenas uma questão de tempo para que se torne algo comum, assim como aconteceu com a programação estruturada e a orientada a objetos.

Os recursos disponibilizados pelas tecnologias estudadas neste trabalho parecem que realmente vêem ao encontro de muitas necessidades em nível de programação e em nível mercadológico também.

O trabalho alcançou o seu objetivo, colocando lado a lado as duas principais frentes de tecnologia para o desenvolvimento de objetos distribuídos que parecem estar prontas para o mercado, servindo como poderosa ferramenta para o desenvolvimento de diversos tipos de aplicação.

Embora o CORBA tenha a grande vantagem de ser independente de plataforma, abrindo assim o mundo “não Microsoft” para os desenvolvedores para a plataforma Windows, não se pode deixar de observar que a Microsoft com seus sistemas operacionais têm uma grande representatividade no mercado mundial, colocando o DCOM também como uma boa opção.

Os experimentos mostraram a ferramenta Delphi como ótima opção para o uso da tecnologia DCOM, porém para o CORBA ainda estão faltando facilitadores, principalmente o mapeamento de interfaces IDL para o Object Pascal, o que permitiria a comunicação de forma mais rápida com objetos não desenvolvidos em Delphi.

7.2 DIFICULDADES ENCONTRADAS

Por serem tecnologias novas e ainda pouco difundidas no mercado, a comparação entre CORBA e DCOM não pode ser baseada em nenhum testemunho, ficando assim com as conclusões tiradas pelos estudos e pela prototipação.

7.3 SUGESTÕES

Como sugestão para trabalhos futuros, fica a do estudo de outra ferramenta que suporte CORBA e DCOM para desenvolvimento de objetos distribuídos.

Pode-se fazer um estudo do uso destas ferramentas para o desenvolvimento de aplicações para Internet.

REFERÊNCIAS BIBLIOGRÁFICAS

[CAN2000] CANTÙ, Marco. **Dominando o Delphi 5.0 - a bíblia.** São Paulo: Makron Books, 2000.

[CAP1999] CAPELETTO, Johni Jeferson. **Comunicação entre objetos distribuídos utilizando a tecnologia CORBA (*Common Object Request Broker Architeture*)**. Blumenau: Universidade Regional de Blumenau. 1999.

[FUR1998] FURLAN, José Davi. **Modelagem de objetos através da UML – the unified modeling language.** São Paulo: Makron Books, 1998.

[MAI1997] MAINETTI, Sérgio. **Objetos distribuídos**. Curitiba: Visionnaire, junho 1997.

[MIC2000] MICROSOFT. MSDN – **Microsoft developer network**: 2000. Endereço eletrônico: http://msdn.microsoft.com. Data da consulta 13/03/2000.

[OMG1995] OMG, Object Management Group. **The object model:** 1995. Endereço eletrônico: http://www.infosys.tuwien.ac.at/Research/Corba/OMG/obmod2.htm. Data da consulta: 10/04/2000.

[OMG1998] OMG, Object Management Group. **Common object request broker architecture and specification v2.2. Object Management Group**: 1998. Endereço eletrônico: http://www.omg.com. Data da consulta: 03/04/2000.

[SCH1999] SCHMIDT, Douglas. **Overview of CORBA**: 1998. Endereço eletrônico: http://www.cs.wvstl.edu/~schmidt/corba-overview.html. Data da consulta: 08/05/2000.

[SIE1996] SIEGEL, Jon; FRANTZ, Dan; et all. **CORBA fundamentals and programming.** New York, EUA : John Wiley & Sons, 1996.