Estudo das Plataformas de Computação Distribuída COM/DCOM e Aplicações em Banco de Dados Distribuídos

RESUMO

Este trabalho tem como objetivo o estudo de técnicas e padrões que consolidam a computação distribuída, focando os padrões COM e DCOM da Microsoft, nos conceitos e na arquitetura, verificando a dificuldade para se trabalhar com estas tecnologias e suas vantagens e desvantagens perante as grandes plataformas de desenvolvimento.

INTRODUÇÃO

De forma simplificada, um sistema distribuído é aquele constituído de programas que executam simultaneamente em vários computadores interligados em rede, tentando dar ao usuário a impressão de que este é um sistema único e isolado.

A utilização de sistemas distribuídos tem aumentado cada vez mais em empresas e corporações, devido a diversos fatores como:

- Desenvolvimento das redes de computadores com grande desempenho e altas capacidades de banda;
- Disseminação do uso de computadores pessoais e de estações de trabalho em substituição aos mainframes;
- Heterogeneidade entre plataformas de hardware e software;
- Aumento de capacidade de processamento;
- Necessidade de uma transparência mais efetiva na utilização dos processos computacionais.

O uso da tecnologia de objetos distribuídos está contribuindo no desenvolvimento de sistemas distribuídos, uma vez que os mesmos são implementados, na maioria das vezes, em plataformas heterogêneas. Estes objetos distribuídos permitem, além da distribuição das aplicações, a reutilização de software, evitando assim custos extras com o desenvolvimento de novos programas ou a adaptação dos mesmos na utilização no sistema distribuído.

Objetos COM - component object model

OLE (*Object Linkind and Embedding*) é uma tecnologia que define um procedimento padronizado em que um módulo cliente e um módulo servidor podem se comunicar através de uma determinada interface. Aqui, "*módulo*" indica um aplicativo ou uma biblioteca DLL (*Dynamic Link Labraries*). Os dois módulos podem ser executados no mesmo computador ou em computadores diferentes. Há várias interfaces possíveis dependendo do papel do cliente e do servidor e pode-se acrescentar interfaces novas

com objetos específicos. Essas interfaces são completadas por objetos servidores. Um objeto servidor normalmente implementa mais de uma interface e todos os objetos servidores têm serviços em comum.

Por definição, COM é a implementação do OLE. Na prática, os termos OLE e COM são freqüentemente utilizados de modo intercambiável. COM especifica os detalhes técnicos de OLE e qualquer linguagem de concordância COM pode ser utilizada para escrever objetos COM/OLE. Inicialmente, os códigos dos objetos COM eram escritos através das linguagens de programação C e C++. Mas também é possível desenvolver novos objetos COM utilizando outras linguagens de programação, como por exemplo, o *ObjectPascal* (compilador do Ambiente de desenvolvimento Delphi). No código-fonte, o *ObjectPascal* é mais fácil de utilizar do que C++ para escrever objetos COM. Outra linguagem muito utilizada é o Visual Basic, da Microsoft, que é totalmente compatível com esta tecnologia.

Se OLE é um conjunto de interfaces, é importante observar que essas interfaces servem para a comunicação entre dois módulos de software, dois arquivos executáveis, ou um arquivo executável e uma DLL. Implementar objetos DLL geralmente é simples, porque no Win32 um programa e as suas DLLs residem no mesmo espaço de endereço de memória. Isso significa que, se o programa passa um endereço de memória para a DLL, o endereço permanece válido. Quando são utilizados dois arquivos executáveis, torna-se muito trabalhoso para o OLE, devendo ser utilizado o conceito denominado ***marshaling*** para permitir que os dois aplicativos se comuniquem. ***Marshaling*** é o mecanismo que permite ao cliente fazer chamadas às funções da interface para objetos remotos em outro processo ou em uma máquina diferente.

Com objetos COM, a aplicação cliente não precisa conhecer onde o objeto reside, simplesmente gera chamadas para interfaces de objetos. Um objeto COM executa os passos necessários para gerar a chamada para localizar o objeto desejado. Estes passos diferem, dependendo se o objeto está no mesmo processo no cliente (*Servidores in-process*), em diferentes processos no cliente (*Servidores ouf-of-process*) ou em outras máquinas (*Servidores remotos*), como são descritos abaixo:

- **Servidores in-process**: Uma DLL (*Dynamic Link Labraries*) executando no mesmo espaço da aplicação cliente. Por exemplo: um controle *ActiveX* embutido em uma página da WEB visualizado no Internet Explorer ou Netscape. Aqui, o controle *ActiveX* é "baixado" para a máquina do cliente e invocado com alguns processos do WEB Browser. O Cliente comunica in-process com servidor usando chamadas da interface COM.
- **Servidores out-of-process**: Outra aplicação (*EXE*) executando em um espaço diferente de processo, mas na mesma máquina do cliente. Por exemplo: uma planilha eletrônica do Excel embutida em um Documento do Word; duas aplicações separadas executando na mesma máquina. O servidor local usa a interface COM para comunicar com o cliente.
- **Servidores remotos:** Uma DLL (*Dynamic Link Labraries*) ou outra aplicação executando em uma máquina diferente da máquina do cliente, por exemplo: um Banco de Dados em Delphi está conectado com uma aplicação servidora em outra máquina da rede. O Servidor remoto usa a Interface DCOM (*Distributed COM*) para comunicar com a aplicação servidora.

Na **figura 2** quando um processo qualquer ou um processo em uma máquina diferente são chamados, a interface COM usa o **proxy** para iniciar as chamadas remotas. O **proxy** é um objeto que prove o parâmetro **marshaling** e a comunicação requerida ao cliente para chamar uma aplicação que está executando em um ambiente diferente, tal como uma parte de um outro processo. O **proxy** reside nos mesmos processos no

cliente e é utilizado para verificar todas as chamadas a interfaces no cliente. O **proxy** detecta as chamadas no cliente e redireciona-as para um objeto "*real*" em execução. Este mecanismo torna-se ativo no cliente quando acessamos objetos em diferentes espaços de processamento ou em máquinas diferentes, se os processos estiverem no mesmo espaço de endereçamento, este procedimento é chamado de **marshaling**. A diferença entre servidores out-of-process e servidores remotos é o tipo de comunicação usada entre os processos. ([2], [3])

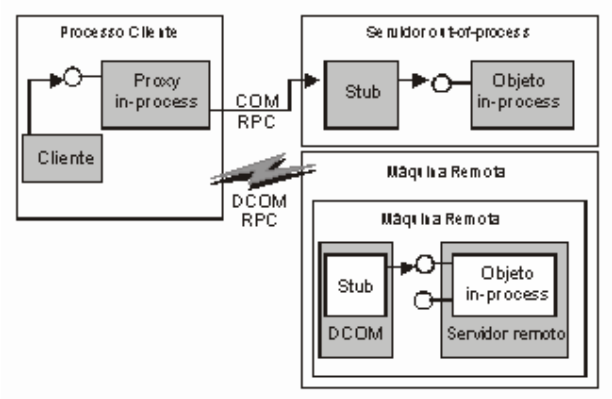


figura 2: exemplo de servidor out-of-process e servidor remoto

FUNCIONAMENTO DA TECNOLOGIA COM

Entender como COM trabalha pode parecer complicado a princípio. Uma razão para isto é o fato que COM utiliza uma notação específica. Uma segunda razão é que COM contém vários conceitos novos. Um dos modos mais fáceis de compreender os conceitos é comparar objetos COM com objetos C++ normais, identificando as semelhanças e diferenças. Pode-se traçar conceitos pouco conhecidos de COM no modelo padrão de C++, que é mais conhecido pelos programadores facilitando o entendimento.

Considere a classe denominada **Teste** escrita em C++, contendo os métodos **MetodoA**, **MetodoB** e **MetodoC**. Cada método possui parâmetros de entrada e possui e retorna um dado valor de retorno.

Declaração da classe Teste:

```
class Teste {

public:

int MetodoA(int a);

int MetodoB(float b);

float MetodoC(float c);
```

};

A própria declaração da classe descreve a classe. Para utilizar esta classe, deve-se instanciá-la, ou seja, definir um objeto. Cada objeto ou é criado como uma variável (local ou global) ou é criado dinamicamente usando a declaração **new**. A declaração **new** cria uma variável dinâmica na memória retornando um ponteiro para ela. Como exemplo:

Teste *px; // ponteiro para classe Teste

px = new Teste; // cria o objeto na memória

px->MetodoA(1); // chama o método

delete px; // libera o objeto, desalocando da memória

É importante entender e reconhecer que COM segue o modelo orientado a objetos. COM tem classes, métodos e instâncias como C++. Embora não seja necessário utilizar um novo objeto de COM, será necessário criá-lo na memória. O acesso aos objetos COM é feito através de ponteiros, tendo que liberar o espaço de memória previamente alocado quando terminar a execução do aplicativo.

No código COM não são usadas as primitivas***new*** e ***delete***. Embora o objeto COM do exemplo seja escrito em C++, a sintaxe deste objeto é inteiramente nova. Objetos COM são implementados através de chamadas para o COM API, que provê funções que criam e destroem objetos de COM. Exemplo de um programa COM:

iTeste *pi // ponteiro para Interface COM

CoCreateInstance(,,,,&pi) // Cria a Interface

pi->MetodoA(1); // chama o método

pi->Release(); // libera a interface

Neste exemplo, a classe **iTeste** é chamada de "*interface*". A variável **pi** é um ponteiro para interface. O método ***CoCreateInstance*** cria uma instância do tipo **iTeste.** Este ponteiro para interface é usado para chamar o método. ***Release*** libera a interface.

Entendendo os argumentos do método ***CoCreateInstance***:

STDAPI CoCreateInstance(

REFCLSID *rclsid***,** // Identificador da Classe (CLSID) do objeto

LPUNKNOWN *pUnkOuter***,** // Ponteiro para interface IUnknown do objeto "*externo*"

DWORD *dwClsContext***,** // Contexto de execução do objeto

REFIID *riid***,** // Referencia para o identificador da interface

LPVOID * *ppv* // Ponteiro que recebe o ponteiro para o objeto criado

);

Um ponto importante a ser notado é que os passos básicos para chamar um objeto de COM são idênticos aos passos usados em C++. A sintaxe simplesmente é um pouco diferente.

Há quatro fatores básicos que ditam o projeto de objetos COM:

1) objetos C++ sempre rodam no mesmo espaço de processo. Objetos COM podem rodar entre processos ou computadores.

2) os métodos de um Objeto COM podem ser chamados através de uma rede de comunicação.

3) os nomes dos métodos C++ devem ser únicos em um determinado processo. Nomes de objetos COM devem ser únicos.

4) servidores COM podem ser escritos em uma variedade de linguagens diferentes e em sistemas operacionais completamente diferentes, enquanto que objetos C++ são sempre escritos em C++.

Em COM, pode-se criar objetos em outros processos, ou em qualquer máquina da rede. Não é necessário criar um objeto COM que use a declaração normal de C++ (*new*). Para criar um objeto COM, alguma entidade (*um EXE ou um Serviço*) deve ser responsável pela alocação de memória remota e a criação de objetos. Esta é uma tarefa muito complexa. Este problema é resolvido com um conceito denominado *COM Server*. Este servidor deve que manter comunicação com o cliente.

MÉTODOS COM PODEM SER CHAMADOS ATRAVÉS DE UMA REDE

Tendo acesso a uma máquina na rede e se está instalado o *COM Server* para o objeto a ser utilizado nesta máquina, pode-se criar um objeto COM neste computador.

Considerando que um objeto COM não está necessariamente na máquina local, é necessário um método para apontar para este objeto, embora sua memória esteja em outro lugar. Tecnicamente, não há nenhum modo para fazer isto. Mas na prática, pode ser simulado introduzindo um novo nível inteiro de objetos. Um dos meios de como COM faz isto é através de um conceito chamado **proxy/stub**.

Outro assunto importante é a passagem de dados entre o cliente COM e o servidor COM. Quando dados são passados entre processos, *threads* ou rede, é chamado "***marshaling***". Novamente, o **proxy/stub** cuida do ***marshaling***.

OBJETOS COM DEVEM SER ÚNICOS

Objetos COM devem ser únicos no mundo. Isto pode parecer em princípio, um exagero. Mas considere a Internet como a rede mundial. Até mesmo se for utilizado uma única máquina, COM tem que manipular a possibilidade. Singularidade é o assunto. Em C++ todas as classes são manipuladas inequivocamente pelo compilador.

O compilador pode ver as definições da classe utilizadas em um programa e pode corresponder a todas as referências. O compilador também garante que há apenas uma classe com um determinado nome. COM tem que garantir que há somente um objeto com um determinado nome, embora o número total de objetos disponível em uma rede pode ser enorme. Este problema é resolvido criando um conceito chamado GUID, que será visto posteriormente.

OBJETOS COM são INDEPENDENTES DA LINGUAGEM

Podem ser escritos servidores COM em linguagens diferentes utilizando um sistema operacional completamente diferente. Objetos COM têm a capacidade de ser remotamente acessados. Isto significa que eles podem estar em um *thread*, em um processo diferente ou até mesmo em um computador diferente. O outro computador pode estar rodando debaixo de um sistema operacional diferente. Para transmissão de parâmetros pela rede é especificada uma interface entre o cliente e servidor. Também há um compilador novo chamado MIDL (*Microsoft Interface Definition Language*). Este compilador especifica genericamente a interface entre o servidor e cliente. MIDL define objetos COM, interfaces, métodos e parâmetros.

A INTERFACE

Em COM, interface tem um significado muito específico e interfaces COM são um conceito completamente novo, não disponível em C++. O conceito de uma interface é inicialmente difícil de entender porque uma interface é uma entidade que não tem uma existência concreta.

Uma interface nada mais é que uma coleção de funções. Os membros daquela interface são todas as funções públicas da classe. Em outras palavras, a interface é a parte visível da classe (*parte pública*). Em C++ não há quase nenhuma distinção entre uma interface e uma classe. Exemplo de uma classe:

class Teste2

```
{

public:

int  DoThis();

private:

void  Helper1();

int  count;

int  x,y,z;

};
```

Nesta classe, apenas é possível ter acesso aos membros públicos. Este subconjunto público da classe é a '*interface*' para o mundo externo. Essencialmente a *interface* esconde o conteúdo da classe do "*usuário*". Uma interface COM não é uma classe C++. Interfaces e classes COM têm seu próprio conjunto especial de regras e convenções.

COM permite uma ***coclass*** (*classe COM*) ter múltiplas interfaces, onde cada interface tem seu próprio nome e sua própria coleção de funções. A razão para esta característica é permitir objetos mais complexos e funcionais.

Uma das regras principais de COM é que é possível acessar um objeto COM através de uma interface. O programa cliente é completamente isolado da implementação do servidor por interfaces. Este é um ponto extremamente importante. Considere o seguinte exemplo: quando uma pessoa entra em um carro, esta se depara com uma variedade de interfaces de usuário. Há uma interface que permite dirigir o carro, outra que permite trabalhar os faróis, outra para controlar o rádio. E assim por diante...

Um objeto COM pode apoiar em uma coleção de interfaces, onde cada uma tem um nome. Para objetos COM criados, freqüentemente é definida e usada uma única interface COM. Mas muitos objetos COM existentes suportam múltiplas interfaces COM dependendo das características que eles apóiam. A interface o isola dos detalhes de implementação.

Quando se constrói um objeto COM, é necessário entender como as interfaces trabalham. O usuário da interface, porém, não precisa conhecer os detalhes de implementação. Como os freios em um carro, o usuário só se preocupa como a interface trabalha, não sobre os detalhes atrás da interface.

Este isolamento de interface e implementação é crucial para objetos COM. Isolando a interface de sua implementação, pode-se construir componentes que podem ser substituídos e reutilizados. Isto simplifica e multiplica a utilidade do objeto.

DEMONSTRANDO UM OBJETO COM TÍPICO

Neste ponto, serão apresentados conjuntamente, todos estes novos conceitos vistos inicialmente e que são necessários para descrever um objeto COM típico e um programa que o acessa. O objeto COM simples contém uma única interface e esta interface possui uma única função. O propósito da função também é extremamente simples – é um *beep*. Quando um programador cria este objeto COM e chama a função, a máquina na qual o objeto COM reside vai dar um *beep*.

Passos necessários para criar este objeto COM:

- Criar o objeto COM e dar um nome a ele. Este objeto deve ser implementado dentro de um servidor COM;
- Definir a interface e dar um nome a ela;
- Definir a função na interface e dar um nome a ela;
- Instalar o servidor COM.

Neste exemplo, o objeto COM chama-se *Beeper*;a interface de *iBeep* e a função *Beep*. Um problema encontrado é a identificação (*nome*) dos objetos, pois é permitido que todas as máquinas no universo COM apóiem servidores de COM múltiplo, cada um contém um ou mais objetos COM, cada objeto implementa uma ou mais interfaces. Estes servidores são criados por uma variedade de programadores, e não há nada para impedir os programadores de escolher nomes idênticos. Algo deve ser feito para prevenir colisão de nome, o conceito de ***GUID (Globally Unique Identifier),*** resolve este problema.

Há realmente apenas dois modos definitivos para assegurar que um nome é único:

- Registrar os nomes em alguma organização governamental;
- Um algoritmo especial que gera números únicos garantindo ser único pelo mundo inteiro.

O primeiro modo é como os nomes de domínio são administrados em uma rede. Tem o problema que é necessário pagar (*~$50,00*) para registrar um nome novo e leva vários dias para registra-lo e entrar em vigor.

O segundo modo consiste em inventar um algoritmo onde é garantido se criar um nome único. De fato, este problema foi especificado originalmente pelo Open Software Foundation (*OSF*). O OSF propôs um algoritmo que combina um endereço de rede (*IP*), o tempo (em 100 nano-segundos) e um contador. O resultado é um número de 128 bits que é único.

O número 2 elevado a 128 (2^{128}) é um número extremamente grande. Poderia ser identificado cada nano-segundo desde o início do universo. OSF chamou este de UUID (*Universally Unique Identifier*). A Microsoft usa este mesmo algoritmo para o COM. Em COM a Microsoft decidiu chamar de **GUID** (*Globally Unique Identifier*).

A convenção para escrever GUID's é em hexadecimal. Exemplo: **"50709330-F93A-11D0-BCE4-204C4F4F5020".** Embora a estrutura de um GUID consiste em quatro campos diferentes, provavelmente será preciso manipular seus membros. A estrutura sempre é usada em sua totalidade.

typedef struct _GUID {

unsigned long Data1;

unsigned short Data2;

unsigned short Data3;

unsigned char Data4[8];

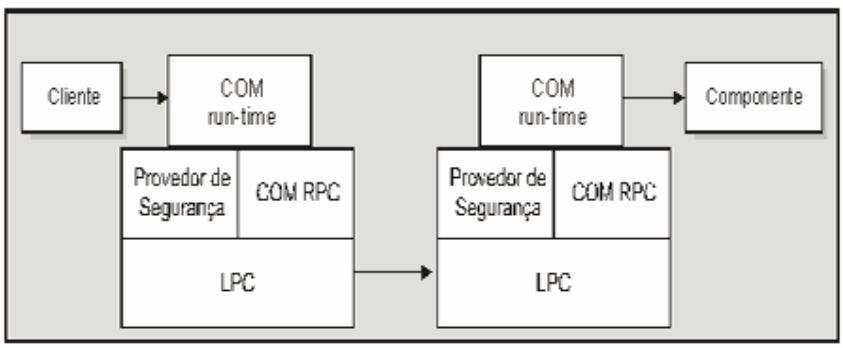
} GUID;

GUIDs são gerados por um programa chamado GUIDGEN. Em um GUIDGEN pressiona-se um botão para gerar um novo GUID. É garantido que cada GUID gerado é único. Pode trabalhar com a seguinte suposição: todas as máquinas na Internet têm um endereço IP único. Então, sua máquina deve estar na rede para que GUIDGEN trabalhe o seu maior potencial. De fato, se não tem um endereço de rede, GUIDGEN vai falsificar um, mas isto reduz a probabilidade de singularidade.

Objetos COM e interfaces de COM têm GUIDs para identificar ambos. Assim o nome "*Beeper*" escolhido para o objeto é na verdade irrelevante. O objeto é nomeado por seu GUID. Pode-se usar um **#define** ou uma constante para relacionar o nome "*Beeper*" com o GUID de forma que não há necessidade em usar um valor de 128 bits ao longo do código. Note que aqueles muitos objetos de COM diferentes criados por muitos programadores podem apoiar a mesma interface *iBeep*. [4]

FUNCIONAMENTO LOCAL DO COM

Componentes COM no mesmo processo



DCOM - DISTRIBUTED COMPONENT OBJECT MODEL

Com o COM já era possível a comunicação de aplicativos numa mesma máquina e o próximo passo era conseguir a comunicação de aplicativos em uma rede. Esse papel foi atribuído ao **Distribuited Component Object Model** (*DCOM - Modelo de Objeto Comum Distribuído*). Embora o COM tenha sido lançado em 1993 o DCOM foi lançado somente em 95 com o lançamento do Windows NT 4.0. Certamente um importante membro da família *ActiveX*, o DCOM implementa um mecanismo de segurança distribuído oferecendo autenticação e encriptação de dados.

O DCOM (*Distributed COM*) permite que objetos de diferentes computadores possam se comunicar através de protocolos comuns, incluindo protocolos baseados na Internet e na Web. Mas até aqui, a comunicação de objetos estava limitada a um único sistema ou exigia programação dispendiosa ou software de utilitários para poder funcionar em rede.

Devido o DCOM ser nativo do ambiente Windows, ele possui um total domínio desta plataforma. Quando se utiliza o DCOM se fala em *ActiveX*, então daí vem o domínio das plataformas Windows, mas o DCOM já não é mais uma exclusividade do ambiente Windows. A Microsoft está trabalhando em conjunto com a Bristol Technology e a Mainsoft para dar suporte ao *ActiveX* na plataforma Unix e também se uniu a Metrowerks e a Macromedia para fazer o mesmo na plataforma Macintosh, sendo que já está disponível no mercado uma versão do ActiveX SDK para Mac.

O Open Group também está em consórcio com a Microsoft para a divulgação e a implementação do DCOM em outras plataformas, mas com uma parceira do porte da Microsoft, que é detentora de grande parte do mercado mundial de sistemas operacionais para microcomputadores, o Open Group não terá grande poder de decisão. Na verdade, ele disponibilizará para a Microsoft apenas seus serviços e sua estrutura, quando for relevante ou necessário para esta.

A comunicação de objetos utilizando DCOM é tão segura quanto a comunicação de objetos utilizando o COM. É totalmente transparente ao programador ou usuário.

A **figura 3** mostra a comunicação entre dois objetos em rede utilizando o modelo DCOM. Um cliente faz a requisição a um servidor de objetos em outro ponto da rede. Essa requisição é padronizada pelas camadas do COM, passando por mecanismos de segurança até o **DCE RPC**. Este se comunica ao servidor através do **Protocolo Stark**. O servidor efetua a operação inversa até o componente requisitado.

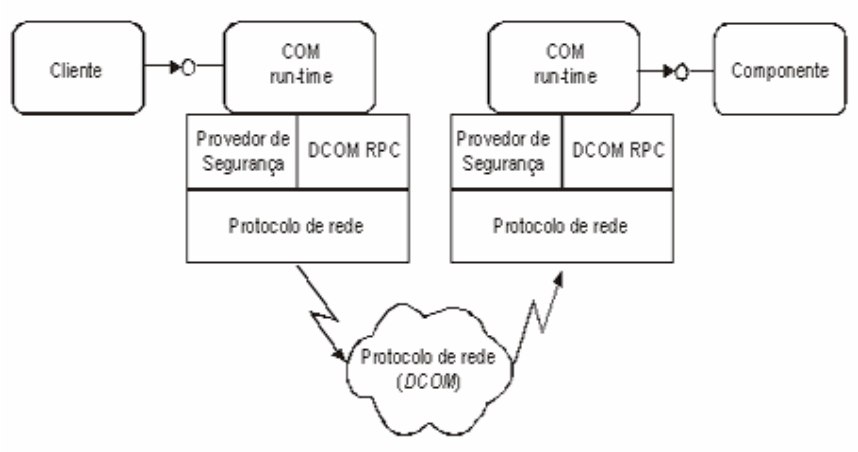


figura 3: DCOM – Componentes COM em máquinas diferentes

O DCOM oferece uma organização com muitos benefícios que vão além da criação de aplicações em rede. A seguir, mostraremos alguns desses benefícios.

NEUTRALIDADE DAS LINGUAGENS DE PROGRAMAÇÃO

Se uma aplicação for escrita em DELPHI, ela poderá acessar outra aplicação escrita em Visual Basic. Isso libera os desenvolvedores de ficarem presos a uma única ferramenta, possibilitando escolher a ferramenta mais adequada para um determinado trabalho.

GERÊNCIA DE CONEXÕES

O DCOM utiliza um contador de referência para determinar quantos clientes estão conectados a um determinado servidor. Assim, quando um novo cliente se conecta ao servidor, o contador se incrementa e, caso contrário, quando um cliente se desconecta do servidor de aplicações, o contador é decrementado. Quando o contador de referência chega a zero, a aplicação se encerra. Quando um servidor de aplicações não mais se encontra em funcionamento, o DCOM abandona a aplicação e estabelece um contador de referência da maneira mais adequada.

Se um cliente se desconecta sem que o contador de referência seja decrementado, deve existir um modo para que o servidor decremente o contador de referência. Isto é feito enviando um **ping** (*sinal enviado ao destinatário verificando se o mesmo está ativo*) do servidor para o cliente. Se o servidor não conseguir enviar um **ping** três vezes seguidas, a conexão será considerada encerrada e o contador de referência será ajustado.

INTERFACES MÚLTIPLAS

O DCOM é implementado fundamentalmente com a utilização de interfaces. As interfaces são os métodos expostos e as propriedades de um objeto em particular. Pode haver muitas interfaces para um mesmo objeto. Isto oferece a oportunidade de personalizar um objeto de acordo com a necessidade do programador, sem ter a necessidade de reconstruí-lo. Isso é muito importante quando um objeto requer uma funcionalidade adicional que somente necessita de um subconjunto dos clientes. Nessa situação, o programador pode utilizar uma mesma implementação com diferentes interfaces. Uma interface pode proporcionar uma funcionalidade básica necessária e outra pode proporcionar uma funcionalidade melhorada. Isto poderá levar, em ambos os casos, a uma mesma implementação real.

INTEGRAÇÃO COM OUTROS PROTOCOLOS DE REDE

Como o DCOM está baseado nos protocolos Transmission Control Protocol/Internet Protocolo (TCP/IP) e Remote Procedure Call (RPC), os servidores de aplicação podem ser invocados de qualquer ponto da INTERNET utilizando o TCP/IP básico ou outros protocolos baseados em TCP (como Point to Point Tunneling Protocol - PPTP) para usuários que conectam o HTTP da INTERNET. Suportando o nível mais baixo de interoperabilidade de rede, a comunicação em rede do DCOM é transparente para as aplicações. Isto oferece muitas opções de rede distintas para o desenvolvedor, o qual poderá escolher a melhor conectividade para a aplicação.

INDEPENDÊNCIA NA LOCALIZAÇÃO E EQUILÍBRIO DE PROCESSOS

Este é o conceito onde uma aplicação cliente não precisa se preocupar em averiguar onde está localizado o servidor de aplicações para poder se conectar ao mesmo. Como o DCOM está dirigido para as aplicações conectadas na rede, o único critério que se deve verificar é se a aplicação realmente exista na rede. O modo como uma aplicação cliente localiza um servidor de aplicações é encontrando as entradas de registro adequadas no arquivo de registro da aplicação cliente. Simplesmente para trocar informação do servidor A com o servidor B o cliente terá que processar a aplicação no servidor B. Quando uma aplicação tem vários usuários, estes podem se dividir em grupos de usuários. Por exemplo, o grupo 1 pode utilizar uma aplicação em um Servidor A, enquanto que o grupo 2 pode utilizar a mesma aplicação em um Servidor B. Havendo isso, a carga pode equilibrar-se ao longo das diferentes máquinas.

Este tipo de equilíbrio de processo se chama estático, porque os usuários sempre conectam com o mesmo servidor, seja qual for o número de usuários que estejam conectados. Isto ocorre em situações simples, mais na maioria das vezes os processos precisam se equilibrar dinamicamente. Tomando o exemplo acima não tão completo, mas realista, em um certo momento o Grupo1 poderá requerer muito mais do servidor de aplicações e nesse mesmo momento as requisições do Grupo2 serão baixadas. Se o equilíbrio de processo for estático, o Servidor1 estará bastante subtilizado enquanto que o Servidor2 estará sobrecarregado. Se de alguma forma alguns usuários do Grupo1 passarem a utilizar o Servidor2 ao invés do Servidor1, se basear no tráfego de

rede existente para cada servidor haverá um equilíbrio na utilização global dos servidores, melhorando a performance da rede.

O DCOM por si mesmo não resolve o equilíbrio dinâmico de carga. Entretanto, recomenda-se que os programadores criem um objeto de referência para ocupar-se deste requisito de aplicação.

SEGURANÇA FRENTE ÀS FALHAS E TOLERÂNCIA AOS ERROS

O DCOM proporciona suporte básico frente aos erros com as utilidades do **pinging**. Essencialmente, isso significa que se um servidor de aplicações falhar por qualquer razão, o DCOM tentará reinicializar a aplicação. Ainda que esta seja uma função importante, não resolve os momentos em que o servidor de aplicações não pode se reinicializar por diversos motivos, incluindo falhas no fornecimento de energia elétrica, falhas no hardware, erros na aplicação etc.

Segundo a Microsoft, a solução para um sistema de aplicações realmente tolerante à falhas se descreve no texto "*DCOM Technical Overview*", da seguinte maneira:

"DCOM facilita a implementação de tolerância aos erros. Uma técnica é o componente de referência introduzido na seção anterior. Quando os clientes detectam a falha de um componente, voltam a se conectar com o mesmo componente de referência que fez a primeira conexão. O componente de referência tem informação sobre os servidores que já não estão disponíveis e proporciona automaticamente ao cliente uma nova amostra do componente funcionando em outra máquina".

INTEGRAÇÃO DE BASE DE DADOS

Para compreender as questões sobre integração de base de dados, podemos estudar os fundamentos do DCOM. O DCOM se baseia em chamadas a procedimentos remotos (RPC). Entretanto, há uma limitação na RPC no que se refere ao conjunto limitado de tipos de dados suportados. Esses tipos de dados são tipos simples, como exemplo: *binários, inteiros, vetores e array*. A transferência de tipo de dados complexos, como objetos, não é suportada. Ainda que os dados de uma base de dados entrem na categoria dos tipos de dados simples, os conjuntos de dados não podem entrar. Alguns fabricantes estão aproveitando essa questão para entregar conjuntos de tipos de dados simples, um vetor para cada coluna. Ainda que essa técnica transporte os dados, a idoneidade para os dados se perde na conversão. É necessário que exista um mecanismo que distribua um conjunto de dados de bases de dados (*DataSets*), que permita ao usuário examinar e modificar esse conjunto de dados e permita ao usuário enviar essas modificações novamente para o objeto finalmente à base de dados.

Além de devolver os dados da base de dados, há também uma questão a respeito das regras comerciais. Uma função básica de um servidor de base de dados é a integridade (*a validação dos dados*). Se não for realizada uma codificação adicional no cliente para evitar óbvias infrações na integridade dos dados, o usuário final só será informado

dessa infração das regras uma vez que tenha enviado os dados de volta ao servidor de aplicações ou ao SGBD. Isso dará como resultado uma interface de usuário nada fácil de se manejar e tráfego desnecessário na rede. A alternativa é reprogramar as regras no módulo cliente da aplicação. Entretanto, isso tem outros efeitos secundários. Se a regra for modificada, o módulo cliente deve ser atualizado. Ainda que possa parecer uma simples operação, onde se tem muitas aplicações clientes distintas que necessitem ser reconstruídas e muitas máquinas clientes para reinstalar o software, isto é na verdade um processo complexo e que requer bastante tempo.[4]

IMPLEMENTAÇÃO DE UMA BASE DE DADOS DISTRIBUÍDA

Após o estudo das tecnologias COM e DCOM é explicado o processo de desenvolvimento de uma aplicação Cliente/Servidor simples, utilizando uma base de dados distribuída, com a tecnologia DCOM. Os exemplos são descritos em Delphi, pelo fato de oferecer uma série de recursos para tais procedimentos.

Nossa aplicação foi desenvolvida sobre o Sistema Operacional Windows 98, utilizando a linguagem Delphi 4 para implementação da aplicação.

Foi desenvolvida uma aplicação Servidor (*DataCOMServer*) que fornece a base de dados para o cliente. O servidor envia as informações para o cliente, tornando possível a edição dos dados e não é utilizada ferramentas locais para edição da base de dados, ou seja, esta aplicação cria uma aplicação de base de dados distribuída, com um cliente totalmente "leve". Para a troca de informações entre cliente e servidor, foi utilizada uma variável do tipo *OleVariant*, que aceita qualquer tipo de informação suportada pela tecnologia COM.

ENTENDENDO O SERVIDOR DE DADOS

A aplicação DataCOMServer é bastante complexa e será explicado seu funcionamento, focando somente nos pontos mais importantes de sua implementação.

Declaração da Classe de Automatização:

```
IDataCOM = interface(IDispatch)

['{41E564F4-2A18-11D2-9CE8-006008928EEF}']

function GetName: WideString; safecall;

function UpdateRecord(Data: OleVariant): WordBool; safecall;

function GetData: OleVariant; safecall;

end;
```

- A função *GetName* é utilizada para testar a comunicação com o Servidor;
- A função *GetData* é utilizada para reparar o conjunto de dados na variável V (OleVariant), que contém os registro selecionados de um banco de dados. O código repete por todos os registros do banco de dados para pegar a informação.
- A função *UpdateRecord* é usada pelo cliente quando ele quer atualizar dados no servidor. Para instanciar, o usuário pode editar um registro em particular e então enviar todas as edições para o servidor com este função.

A declaração abaixo mostra como o Servidor "empacota" os dados antes de enviar ao Cliente:

```
Procedure TForm1.GetData(out V: OleVariant);

Var Customers: TCustomerRecords;

P: Pointer;

begin

V := VarArrayCreate([0, SizeOf(TCustomerRecords)], varByte);

DMod.GetCustAry(Customers);

P := VarArrayLock(V);

Move(Customers, P^, SizeOf(TCustomerRecords));

VarArrayUnlock(V);

end;

end;
```

A declaração acima chama a rotina *GetCustAry* que retorna uma estrutura contendo os dados da tabela *Customer*. O método *GetData* converte a estrutura de uma tabela para o tipo *Variant Array*, usando as funções *VarArrayLock* e *VarArrayUnlock* e este *Variant Array* é passado para o aplicativo Cliente como um pacote de informações.

A automação é comum em COM e serão incorporados na funcionalidade de seu servidor. A meta do Servidor Automatizado é simplesmente exportar a sua funcionalidade para outros programas. Como resultado, faz sentido que o objeto de Automação simplesmente utilizasse métodos existentes no programa.

ENTENDENDO O CLIENTE

A aplicação *DataCOMClient* somente busca a base de dados no Servidor, exibindo os dados e permitindo ao usuário editá-los.

A declaração abaixo provê a comunicação com o Servidor:

```
Procedure TForm1. CreateConnection(IsRemote:Boolean);

Var S: String;

begin

try

if InputQuery('Server Name', Enter Server Name', S) then

FDataCom := CoDataCOM.CreateRemote(S)

end else

FDataCom := CoDataCOM.Create;

Finally

end;

end;
```

Todo o código utilizado para comunicação está descrito acima. Após a busca dos dados do servidor, pede-se para que ele mostre os dados com a função abaixo:

```
Procedure TForm1.GetandDisplayData;

Var V: Variant;

begin

if FDataCom <> nil then

begin
```

```
V :=  FDataCom.GetData;

FCustomers := VariantToData(V);

FillGrid;

end;

end;
```

O código acima chama a função *GetData* da aplicação *DataCOMServer*. Ele também translada os dados do servidor para um bloco de controle de dados chamado pela rotina *VariantToData*. A função *VariantToData* trava o Variant Array obtido do servidor e extrai os dados dele. Esta operação é reversa quando utilizada pelo servidor, quando verifica os dados no Variant Array. Depois, somente será necessário exibir os dados para o usuário. [1]

CONCLUSÕES FINAIS

A utilização de Sistemas Distribuídos cresceu muito nestes últimos anos devido a diversos fatores como: desenvolvimento das redes de computadores, disseminação do uso de computadores pessoais, aumento da capacidade de processamento etc. Há diversas plataformas para desenvolvimento de aplicações distribuídas no mercado, entre as quais pode-se destacar o CORBA, DCOM, MIDAS etc. Este trabalho focou nas tecnologias COM/DCOM da Microsoft, devido à facilidade de acesso e utilização, sendo uma plataforma nativa do Sistema Operacional a partir da Versão NT 4.0; não sendo necessários gastos com aplicativos adicionais

Uma das grandes questões levantadas sobre a tecnologia DCOM diz respeito à confiabilidade se comparada a outras tecnologias citadas acima. Após muitos estudos sobre a tecnologia DCOM, foi observado que esta não possui todos os recursos disponíveis no CORBA. Isto deve-se aos fatos de ser uma tecnologia recente e ainda em desenvolvimento, mas poderá atender muito bem as necessidades básicas de uma empresa. Além disso, não é necessário ter muitos aplicativos para colocar um Sistema Distribuído desenvolvido na tecnologia DCOM em funcionamento, pois como comentado inicialmente, o Sistema Operacional Windows (*presente em 90% das máquinas)*, já provê tais recursos, ao contrário de outras plataformas que exigem muitos aplicativos para monitoração, gerenciamento etc. É claro que todos estes fatores dependem, principalmente, de onde será empregada esta tecnologia, pois ela é muito útil, mas não foi verificada a sua adaptação para grandes Sistemas Distribuídos.

Foi implementada uma arquitetura Cliente/Servidor simples que atendia as necessidades da empresa, mantendo as principais funcionalidades no Servidor, tornando o Cliente operável com o mínimo de código possível.

Em breve esta tecnologia estudada será capaz de competir com grandes tecnologias como o CORBA e o MIDAS, podendo ser utilizada com muito mais freqüência e facilidade pelos desenvolvedores em geral.

BIBLIOGRAFIA

[01] CALVERT'S, Charlie. **Delphi 4 Unleashed**, United States of America: Sams Publishing, 1999.

[02] CANTÙ, Marco. **Dominando o Delphi 4 "A Bíblia"**, São Paulo: Makron Books, 1999.

[03] CANTÙ, Marco. **Dominando o Delphi 5 "A Bíblia"**, São Paulo: Makron Books, 2000.

[04] MICROSOFT CORPORATION. **COM/DCOM.** On-line. Disponível na Internet via WWW: URL: http://msdn.microsoft.com/library/default.asp?URL=/library/ backgrnd/ html/msdn_dcombiz.htm.