Uma Introdução ao CORBA

Eduardo Link, Everton Batista Petró Alexandre, Joe Luiz Wolf, Marcelo da Silva Strzykalski

Resumo. A heterogeneidade dos ambientes de hardware e software empregados em sistemas computacionais distribuídos tem conduzido a uma dificuldade de integrar tais soluções para a geração de aplicações que se utilizem de tais plataformas. A especificação CORBA da OMG tem como objetivo justamente proporcionar o desenvolvimento de aplicações distribuídas baseadas no paradigma orientado a objetos que sejam independentes do sistema operacional bem como da linguagem de programação adotada para a construção do software. Dessa forma, o presente trabalho introduz os conceitos gerais, a arquitetura e os serviços do CORBA. Além disso, apresenta alguns estudos de caso que demonstram a utilização dessa tecnologia no contexto de um modelo cliente-servidor.

1 Introdução

CORBA (Common Object Request Broker Architecture) é uma especificação aberta criada pela OMG (Object Management Group) que define um *framework* para o desenvolvimento de sistemas de software distribuídos baseados na tecnologia de objetos [1] [2].

Tal especificação permite que aplicações distribuídas desenvolvidas em diferentes dispositivos computacionais, linguagens de programação e sistemas operacionais possam interagir, possibilitando que uma coleção de objetos distribuídos heterogêneos possam colaborar de forma transparente.

A OMG desenvolveu uma versão "core" de CORBA, sendo que versões direcionadas para ambientes especializados foram descritas em especificações separadas. O CORBA em tempo real possibilita a elaboração de aplicações que exijam rápidos tempos de resposta entre o envio de uma requisição e a resposta a essa. Por sua vez, o CORBA/e, do qual o CORBA mínimo é um subconjunto, permite o desenvolvimento de aplicativos para plataformas que possuem recursos computacionais escassos, tais como dispositivos móveis, soluções de hardware e software embutidas.

Este artigo tem a finalidade de abordar de forma clara e direta os principais conceitos relacionados com a arquitetura CORBA. Após apresentá-lo conceitualmente, é mostrada uma aplicação simples utilizando uma implementação do CORBA.

A seção 2 apresenta os conceitos gerais do padrão CORBA. A seguir, a arquitetura da tecnologia em questão é detalhada (seção 3). Após, os principais serviços CORBA são analisados. A seção 5 exibe uma aplicação CORBA. As considerações finais do artigo são apresentadas na seção 6.

2 Conceitos Gerais

CORBA é organizado em diversos componentes com o intuito de permitir a integração de sistemas baseados em objetos. Os conceitos centrais de tal arquitetura são abaixo descritos [1].

2.1 Object Request Broker

O Object Request Broker (ORB) é um elemento estrutural que atua como elemento intermediário na transferência de mensagens entre o cliente CORBA e o servidor CORBA. Cabe destacar que o ORB esconde do programador a complexidade inerente das primitivas de comunicação em redes.

Nesse contexto, um cliente é uma entidade que deseja executar uma operação em um objeto que será invocado remotamente, cujo código e dados implementam a funcionalidade requerida. Mais especificamente, uma invocação de objeto remoto envolve a especificação do objeto destino, da operação a ser invocada e dos parâmetros que serão enviados (*in*) e retornados (*out*) desse.

O ORB é responsável por todos os mecanismos necessários para a localização da implementação do objeto invocado, garantindo que a implementação do objeto possa receber a requisição e transferir os dados que foram gerados por essa de volta ao cliente.

Cabe destacar que o cliente pode efetuar uma requisição tanto por meio do emprego da interface de invocação dinâmica quanto por meio de um *stub* gerado em IDL (*Interface Definition Language*). Dessa maneira, uma implementação de objeto recebe uma requisição seja via um *skeleton* gerado em IDL ou através de um *skeleton* dinâmico.

Um cliente CORBA processa requisições por meio de uma referência para o objeto remoto, de modo que esse inicia uma requisição via chamadas a rotinas do *stub* local ou inicia uma requisição pela construção dinâmica da chamada de método. A partir desse passo, o ORB localiza a operação do objeto referenciado, transmite os parâmetros e transfere o controle para a implementação desse objeto por meio do *skeleton* baseado em IDL ou por meio da chamada remota de método que foi construída de forma dinâmica. Quando a requisição está completa, tanto o controle quanto o resultado obtido é retornado para o cliente. Cabe acrescentar que invocações remotas via stub (interface de invocação estática) são geralmente síncronas (a não ser que uma operação seja declarada *oneway* na IDL que a descreve) enquanto que invocações remotas via interface dinâmica podem ser tanto síncronas quanto assíncronas.

O ORB permite a criação de objetos de cujas operações possam ser invocadas por programas clientes localizados em qualquer lugar. Além disso, a interoperabilidade entre ORBs localizados remotamente é garantido pelo uso do protocolo Internet Inter-Orb (IIOP), baseado no TCP/IP (mais especificamente, no TCP).

2.2 Interface Definiton Language

A Interface Definition Language (IDL) define os tipos de objetos CORBA por meio da especificação das suas interfaces, que consistem de um conjunto de métodos nomeados bem como os parâmetros associados a tais operações.

Cada objeto CORBA possui uma interface claramente definida e especificada em IDL, a qual define quais são os serviços que o objeto fornece, isto é, quais são as operações que o objeto suporta, sem fornecer qualquer detalhe de implementação. Cabe destacar que um objeto CORBA pode implementar mais do que uma interface, sendo que essa é neutra com relação às linguagens de programação empregadas para a escrita do programa cliente. Além disso, um cliente não precisa conhecer como o objeto remoto foi implementado, a localização desse bem como em qual sistema operacional esse roda

2.3 Object Management Architecture

Como descrito em [3], a OMA **(**Object Management Architecture), foi criada pelo OMG, com o objetivo de integrar as aplicações baseadas em Objetos distribuídos. Sua estrutura é baseada essencialmente em dois elementos: o *Modelo de Objeto*, que define o Objeto que será distribuído pelo sistema heterogêneo e o *Modelo de Referência*, que define as características da integração destes Objetos.

O RFP (*Request for Proposal*) é utilizado para que os Modelos de Objetos e de Referência, possam ser compatíveis com especificações anteriormente utilizadas e, com isso tornar possível a construção de sistemas de objetos distribuídos interoperáveis em sistemas heterogêneos. CORBA possui toda sua estrutura baseada na arquitetura OMA. Esta define a comunicação entre os Objetos distribuídos através de quatro elementos:

- Objetos de Aplicação: São os Objetos criados pelo usuário, que possuem características definidas de acordo com os seus objetivos. Eles também são tidos como os usuários finais de toda a infra-estrutura CORBA.
- Facilidades Comuns: São componentes definidos em IDL que fornecem serviços para o uso direto das aplicações de Objetos. Estes estão divididos em duas categorias que são,horizontal e vertical. As Facilidades Comuns. definem regras de integração para que os componentes possam colaborar efetivamente.
- Serviços Comuns: São serviços que possibilitam a implementação e utilização de Objetos. Eles definem uma estrutura de Objetos de baixo nível, que ampliam as funcionalidades do ORB, sendo utilizados para criar um componente, nomeá-lo e introduzi-lo no ambiente.
- ORB: É o elemento que permite que Objetos emitam solicitações em um ambiente distribuído e heterogêneo, de forma transparente.

3 Arquitetura

O padrão CORBA passou a ser concebido pela OMG em 1989 com o intuito de proporcionar interoperabilidade entre aplicações heterogêneas [2].

Sua arquitetura é baseada no modelo Cliente/Servidor e no paradigma de orientação a objetos distribuídos. Deste modo, clientes fazem requisições aos objetos CORBA. As solicitações dos clientes e a resposta dos objetos CORBA são realizadas de forma transparente através do ambiente de comunicação ORB [2] [4].

Todos os objetos CORBA que implementam serviços devem ter suas interfaces definidas em IDL, pois, desta forma, estes serviços poderão ser solicitados por clientes desenvolvidos em linguagens de programação distintas da utilizada para implementar o objeto servidor. Após a definição da interface, um compilador IDL gera os *stubs* e os *skeletons*, utilizados para a invocação de métodos remotos. Assim, os clientes desta arquitetura requisitam serviços a um objeto CORBA por meio do *stub*. Depois da chegada da solicitação ao servidor, o *skeleton* invoca o método do objeto que implementa o serviço e envia ao *stub* uma mensagem com os resultados gerados pelo objeto CORBA. Por fim, o *stub* recebe a mensagem, extrai os resultados e os retorna ao cliente solicitante. Este processo é denominado invocação estática.

A invocação dinâmica foi concebida para minimizar o problema da falta de flexibilidade na invocação estática. Deste modo, o padrão CORBA possui duas interfaces responsáveis por este tipo de invocação: a DII(Interface de Invocação Dinâmica) e a DSI(Interface *Skeleton* Dinâmica). A primeira possui a mesma função do *stub*, enquanto que a segunda corresponde ao *skeleton*. Estas interfaces são fornecidas diretamente pelo ORB, portanto, não existe mais a necessidade de cada cliente ou servidor ter uma interface específica. A figura 3.1 ilustra a arquitetura CORBA:

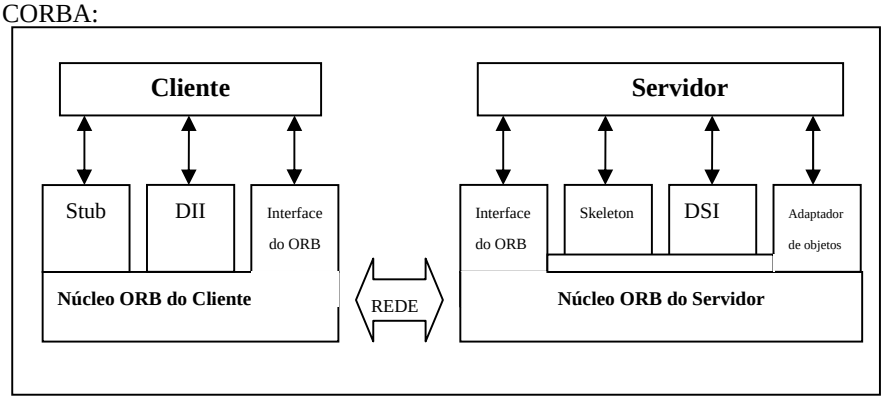

Figura 3.1: Arquitetura CORBA.

A seguir, os principais componentes CORBA serão detalhados [4] [5] [6]:

- Aplicação Cliente: Tem a função de requisitar serviços aos objetos localizados no servidor. É importante ressaltar que o cliente e o servidor podem estar executando sobre ambientes computacionais heterogêneos;
- *Stub*: É a interface de invocação estática gerada a partir da compilação de uma IDL. Esta interface é utilizada pelas aplicações clientes para a realização de chamadas a objetos servidores remotos;
- DII: A interface de invocação dinâmica permite que o cliente chame um método no servidor sem que tenha conhecimento de sua interface em tempo de compilação;
- Repositório de Interfaces: Banco de dados onde residem todas as interfaces dos serviços conhecidos pelo ORB. Este repositório é utilizado por clientes que usam a DII para chamar os métodos de objetos cujas interfaces são desconhecidas em tempo de compilação;
- Aplicação Servidora: Tem por finalidade receber solicitações das aplicações clientes, processá-las e enviar uma mensagem contendo a resposta requisitada;

- *Skeletons*: É a interface estática do servidor. Esta interface recebe requisições do cliente e encaminha ao objeto servidor;
- DSI: É a interface de *skeletons* dinâmica. É uma forma de entregar chamadas de métodos a uma implementação de objeto cujas interfaces não são conhecidas em tempo de execução.
- Repositório de Implementação: O repositório de implementações contém os dados necessários para que a ORB possa localizar e ativar implementações dos objetos;
- Adaptador de Objetos: É um componente da arquitetura CORBA que existe apenas no lado do servidor. É responsável pela criação de referências para objetos CORBA, pela ativação dos objetos e pelo direcionamento das requisições;
- Núcleo ORB: Tem a função de intermediar a comunicação entre o cliente e o servidor.

4 Serviços

Para que seja possível a interoperabilidade entre objetos instanciados nos servidores distribuídos e as aplicações construídas pelos desenvolvedores, somente a estrutura de objetos de **Facilidades Comuns** (IDL - Interface Definition Language) e dos objetos que constituem o **ORB** (Object Request Broker), não são suficientes.

Uma vez que a IDL tem por objetivo resumidamente padronizar as invocações feitas pelos clientes aos métodos de objetos que estão instanciados no servidor, e o ORB de coordenar e interceder entre estas invocações e suas respectivas mensagens de retorno, vê-se a necessidade **serviços** para controlar e executar estas transações entre o cliente/servidor.

Como citado anteriormente as RFP's (*Request for Proposal*) definem um padrão para a criação dos mais diferentes objetos que compõem toda estrutura de um sistema desenvolvido observando a API CORBA. Na figura 4.1 podemos observar os conjuntos de serviços especificados e incorporados **OMA,** e também uma linha do tempo de quando foram criados**.**

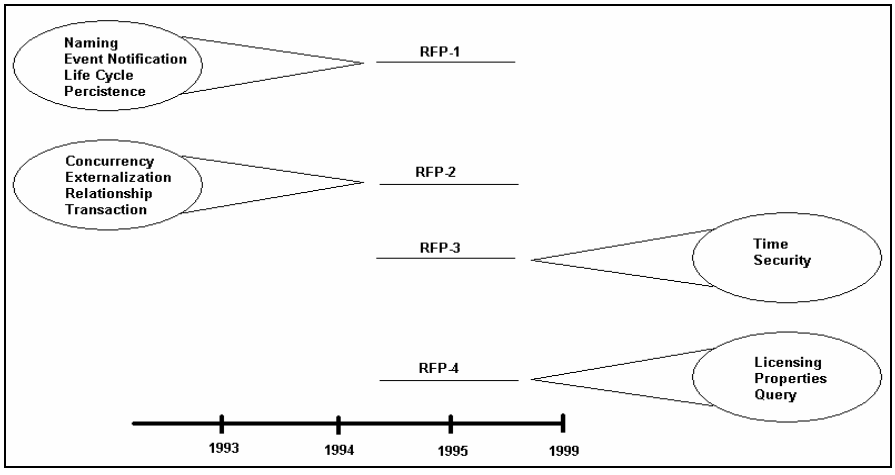

Figura 4.1: Object Services RFP's. Fonte: [7].

A partir de 2000 a **OMG** especifica 16 serviços [6] fundamentais para que aplicações OO (Orientadas a Objeto) e seus componentes possam “rodar” em uma arquitetura CORBA interoperando com objetos distribuídos. Abaixo está uma descrição, resumida, do que cada um dos 16 serviços faz:

- Life Cycle Service (Ciclo de Vida): define serviços e convenções para criar, mover, deletar e copiar objetos. Com isso os clientes podem instanciar objetos sem saber sua localização, pois isso é abstraído pelo sistema.
- Relationship Service (Relacionamento): proporciona um modo para criar associações (links) dinâmicas entre componentes que não sabem nada uns dos outros.
- Naming Service (Nome): este serviço permite que os objetos possam referenciar (e localizar) outros objetos pelo nome.
- Object Transaction Service (Transação): para transações on-line. Fornece às aplicações a confiabilidade, pois em um ambiente distribuído um mesmo objeto pode receber inúmeras solicitações para isso deve-se ter um esquema de “lock” para se manter a integridade das transações.
- Trading Object Service (Negócio): este serviço funciona como páginas amarelas de e para objetos. Permite que você possa localizar qualquer serviço que queira, de qualquer provedor disponível. Da mesma forma, você pode oferecer seus serviços para outros objetos. Chamado com a ajuda do Naming Service.
- Security Service (Segurança): fornece funcionalidades como: autenticação, controle de acesso, auditoria,comunicação segura, criptografia e ferramentas administrativas.
- Concurrency Service (Concorrência): assegura a um banco de dados ou sistema de arquivos que somente um cliente por vez tenha acesso a um registro ou arquivo. É como "chaveador" que atua sobre os objetos no ORB.
- Persistent State Service (Persistência): proporciona uma interface única para os objetos acessarem os vários mecanismos de persistência - bases de dados relacionais, bases de dados OO, e simples arquivos.
- Event Service (Evento): permite que os componentes registrem, ou desregistrem, dinamicamente, os seus interesses por eventos específicos. O sistema poderá emitir-lhes uma notificação assim que o evento estiver disponível.
- Time Service (Tempo): este serviço fornece um mecanismo para sincronizar relógios num ambiente distribuído. Permite ainda definir e gerir eventos acionados pelo tempo.
- Notification Service (Notificação): é uma extensão do Event Service, adiciona funcionalidades como, capacidade de transmitir eventos na forma de dado estruturado; habilidade dos clientes selecionarem somente os eventos que tem interesse em receber; os clientes tem habilidade de descobrir através dos canais de eventos, quais são os tipos que lhes interessam e podem inscrever-se para recebe-los.
- Collection Service (Coleções): com este serviço é possível criar agrupamentos de objetos que possuem comportamento semelhante,

pondendo eles serem acessados via um índice de objetos pertencentes ao grupo.

- Property Service (Propriedades): com este serviço é possível associar qualidades (propriedades) dinamicamente aos objetos. Por exemplo, em um sistema de downloads, a cada vez que um download é feito, um contador de download é incrementado ao objeto, mas este atributo não faz parte da implementação do objeto.
- Query Service (Busca): permite a busca de objetos baseado em predicados, está busca e feita e também retorna coleções de objetos que satisfação as condições. Por exemplo em um BD orientado a objeto poderia ser utilizado a própria linguagem SQL para efetuar estas buscas.
- Externalization Service (Externalizar): é a forma de se gravar em um streem em disco ou em memória ou através da rede um dado estado de um objeto, é uma forma de serializar e depois recarregar um objeto em um outro momento.
- Licensing Service (licenciamento): alguns servidores podem possuir objetos que devam ter certo controle de acesso baseado em um contrato de licença de uso, para isso deve ser possível implementar o serviço de licenciamento que vai tratar quais os tipos de clientes que podem ou não acessar os serviços de determinados objetos.

5 Estudo de caso: aplicações do framework

A seguir será mostrada uma aplicação simples utilizando uma implementação do CORBA, utilizando o modelo cliente-servidor.

A aplicação se trata de um servidor de contas bancárias onde cada cliente pode retirar e depositar dinheiro, e consultar seu saldo. A implementação do servidor foi feita utilizando a linguagem de programação C++. Após a implementação do servidor, são mostradas duas possibilidades para a implementação de um cliente, uma utilizando a linguagem Java e a outra C++. Isto é possível graças à heterogeneidade do CORBA.

Primeiramente temos um arquivo IDL, o skeleton do servidor, que descreve a interface de acesso aos objetos do tipo Bank e Account. Com isto, qualquer cliente saberá como acessar os serviços prestados pelo servidor. A figura 5.1 mostra uma possível implementação do skeleton do servidor.

```
// -*- c++ -*-

interface Account {
    void deposit(in unsigned long amount);
    void withdraw(in unsigned long amount);
    long balance();
};

interface Bank {
    Account create();
};
```

Figura 5.1 Skeleton do servidor (IDL).

A implementação do servidor inclui as declarações das classes. A figura 5.2 mostra uma possível implementação da classe Account, onde cada método descrito no skeleton deve ser implementado. Note que os tipos básicos, tais como inteiros e long, são utilizados com a implementação do CORBA, já que cada arquitetura pode ter uma diferente representação para inteiros, floats, etc.

```
class Account_impl : virtual public POA_Account
{
public:
  Account_impl();

  void deposit(CORBA::ULong);
  void withdraw(CORBA::Ulong);
  CORBA::Long balance();

private:
  CORBA::Long bal;
};

Account_impl::Account_impl ()
{
  bal = 0;
}

void Account_impl::deposit(CORBA::Ulong amount)
{
  bal += amount;
}

void Account_impl::withdraw(CORBA::Ulong amount)
{
  bal -= amount;
}

CORBA::Long Account_impl::balance ()
{
  return bal;
}
```

Figura 5.2 Implementação da Classe Account.

A classe Bank segue os mesmos princípios da classe Account mostrada acima. A figura 5.3 mostra uma possível implementação para esta classe. Note que ele utiliza a classe Account, por isso são necessários alguns comandos específicos para uma correta associação dos objetos.

```
class Bank_impl : virtual public POA_Bank
{
private:
  PortableServer::POA_var theRootPOA;
public:
  Bank_impl(PortableServer::POA_var);
  Account_ptr create();
};

Bank_impl::Bank_impl(PortableServer::POA_var poa)
{
  theRootPOA = poa;
```

```
}

Account_ptr Bank_impl::create ()
{
  Account_impl * servant = new Account_impl;
CORBA::Object_var ref = theRootPOA-
>servant_to_reference(servant);
  Account_ptr acc = Account::_narrow(ref);
  return acc;
}
```

Figura 5.3 Implementação da Classe Bank.

Uma vez definidas as classes podemos ver uma implementação do servidor. A figura 5.4 mostra uma das possibilidades. Neste exemplo é utilizado o serviço de nome (Naming Service) para definir e encontrar os objetos.

```
int main(int argc, char *argv[])
{
  CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);
  CORBA::Object_var poaobj = orb-
>resolve_initial_references("RootPOA");
  PortableServer::POA_var poa =
PortableServer::POA::_narrow(poaobj);
  PortableServer::POAManager_var mgr = poa-
>the_POAManager();

  Bank_impl * servant = new Bank_impl(poa);
  CORBA::Object_var bank = poa-
>servant_to_reference(servant);

  CORBA::Object_var nsobj = orb-
>resolve_initial_references("NameService");
  CosNaming::NamingContext_var nc =
CosNaming::NamingContext::_narrow(nsobj);

  if (CORBA::is_nil(nc)) {
    cerr << "oops, I cannot access the Naming Service!"
<< endl;
    exit (1);
  }

  CosNaming::Name name;
  name.length(1);
  name[0].id = CORBA::string_dup("Bank");
  name[0].kind = CORBA::string_dup("");

  cout << "Binding Bank in the Naming Service ... " <<
flush;
  nc->rebind(name, bank);
  cout << "done." << endl;

  cout << "Running." << endl;
  mgr->activate();
  orb->run();

  /*  Shutdown (never reached) */
```

```
  poa->destroy(TRUE, TRUE);
  delete servant;

  return 0;
}
```

Figura 5.4 Implementação do Servidor.

O cliente também utiliza o Naming Service paraa acessar os objetos no servidor. A figura 5.5 seguida mostra uma possível implementação do cliente, utilizando a linguagem de programação C++.

```
int main (int argc, char *argv[])
{
  CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);
  CORBA::Object_var nsobj = orb-
>resolve_initial_references("NameService");
  CosNaming::NamingContext_var nc =
CosNaming::NamingContext::_narrow(nsobj);

  if (CORBA::is_nil (nc)) {
    cerr << "oops, I cannot access the Naming Service!"
<< endl;
    exit (1);
  }

  CosNaming::Name name;
  name.length (1);
  name[0].id = CORBA::string_dup("Bank");
  name[0].kind = CORBA::string_dup("");

  cout << "Looking up Bank ... " << flush;
  CORBA::Object_var obj = nc->resolve(name);
  cout << "done." << endl;

  Bank_var bank = Bank::_narrow(obj);

  Account_var account = bank->create();
  if (CORBA::is_nil (account)) {
    printf ("oops: account is nil\n");
    exit (1);
  }

  account->deposit(700);
  account->withdraw(450);
  cout << "Balance is " << account->balance() << endl;

  return 0;
}
```

Figura 5.5 Implementação do Cliente em C++.

A figura 5.6 mostra uma possível implementação do cliente utilizando a linguagem Java. O princípio é o mesmo da implementação em C++, utilizando o Name Service para acessar os objetos no servidor. É importante salientar que

qualquer uma destas implementações para o cliente poderia se comunicar com o servidor, graças à heterogeneidade do CORBA.

```
public class Client {
  public static void main(String args[]) {
    try{
      Properties props = new Properties();
      props.put("org.omg.CORBA.ORBInitRef",
"NameService=corbaloc::localhost:9999/NameService");
      ORB orb = ORB.init(args, props);
      org.omg.CORBA.Object objRef =
orb.resolve_initial_references("NameService");
      NamingContextExt ncRef =
NamingContextExtHelper.narrow(objRef);
      String name = "Bank";
      Bank bank =
BankHelper.narrow(ncRef.resolve_str(name));
      Account acc = bank.create();
      acc.deposit(700);
      acc.withdraw(450);
      System.out.println("Balance is: " +
acc.balance());
    }
    catch(Exception e) {
      System.out.println("ERROR : " + e);
      e.printStackTrace(System.out);
    }
  }
}
```

Figura 5.6 Implementação do Cliente em Java.

6 Conclusões

O emprego de CORBA em soluções computacionais do mundo corporativo ainda é pequeno, quando comparado com o uso de outros *middlewares* de comunicação, tais como *Web Services*. Provavelmente, um importante fator explicativo de tal cenário pode ser encontrado na própria complexidade da especificação CORBA (a versão atual é composta de 1152 páginas), que nunca foi implementada na sua totalidade.

Outro problema do CORBA é que ele não tem uma implementação padrão em sua totalidade, e falta suporte para as que existem. Isto dificulta a implementação e o funcionamento de programas que poderiam se beneficiar com as características propostas pelo CORBA. Esta falta de suporte já causa grandes dificuldades para executar os programas, fica mais complicado ainda quando tentamos implementar ou até mesmo compilar os programas. Estes fatores são a grande causa do CORBA não ser utilizado popularmente. Assim, outras opções, mais simples de se realizar este tipo de serviço, como web services, são utilizadas, ainda que percam algumas das características propostas pelo CORBA.

Cabe ressaltar que o referido framework também apresenta atributos positivos. Uma ótima característica do CORBA é que por definição, sua estrutura abstrai os conceitos de sistemas operacionais, protocolos de comunicação, linguagens de programação e hardware. Ou seja, um sistema que implementa esta API é capaz de

dar suporte a inúmeros outros sistemas que tenha interesse em compartilhar objetos. Além do benefício e facilidade do reuso de objetos, ainda se tem a flexibilidade de poder integrar ambientes heterogêneos. Outro grande fator que aprova o uso do CORBA é que os objetos são indiferentes quanto a sua localização, o que proporciona um aumento muito grande da confiabilidade dos sistemas, pois os objetos podem ser transportados entre os servidores via rede, se acontecer de algum servidor ser desconectado o mesmo objeto pode ser transportado para outro servidor e continuar respondendo as solicitações de forma transparente para os clientes.

7 Referências Bibliográficas

[1] Object Management Group. **Common Object Request Broker Architecture: Core Specification**. [on line] Disponível na Internet. URL: http://www.omg.org/technology/documents/corba_spec_catalog.htm. 30.mar.2007.

[2] Siegel, J., **CORBA Fundamentals and Programming.** Nova Iorque: Wiley, 1996.

[3] Souza, Anamélia; Mazzola, Vitório. **Implementando Aplicações Distribuídas Utilizando CORBA: Um Estudo de Caso Voltado à Área de Banco de Dados.** 2.ed. Pará – CESUPA, UFSC – SC.

[4] Harrison, T. H., Levine, D. L., Schmidt, C. D., **The design and performance of a real-time CORBA event service.** In Proceedings of the 12th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, p: 184 – 200, ACM Press, Atlanta, USA, 1997.

[5] Bastide, R., Palanque, P., Ousmane, S. Navarre, D., **Formal specification of CORBA services: experience and lessons learned.** In Proceedings of the 15th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, ACM Press, Minneapolis, USA, 2000

[6] Nardi, A. R., **Componentes CORBA.** Dissertação de Mestrado, USP. São Paulo: SP, 2003.

[7] Object Management Group. **CORBAservices**. [on line] Disponível na Internet. URL: http://www.omg.org/technology/documents/formal/corbaservices.htm. 02.abr.2007.

[8] Ben-Natan, Ron **CORBA: a guide to common object request broker arquiteture.** Nova Iorque: NY, 1995.