COMMON OBJECT REQUEST BROKER ARCHITECTURE - CORBA

Francisca Joamila Brito do Nascimento
joamila.brito@gmail.com

Introdução

A interação entre sistemas distribuídos é um dos assuntos mais pertinentes para os que buscam compartilhar dados e recursos de forma eficaz em um sistema computacional. Acrescido a isso, existe a questão do compartilhamento feito entre sistemas heterogêneos, ou seja, sistemas desenvolvidos em linguagens de programação diferentes e hospedados, por sua vez, em máquinas físicas também diferentes entre si. Nesse contexto surgiu o *Common Object Request Broker Architecture*, conhecido pelo acrônimo CORBA. CORBA é um padrão arquitetural que norteia a troca de dados entre sistemas heterogêneos tornando o processo mais simples.

O objetivo deste trabalho é apresentar o CORBA em suas características teóricas e práticas, incluindo contexto histórico, conceitos associados, arcabouço técnico necessário para a sua utilização, bem como exemplos de seu uso na prática. Para isso, usaremos como metodologia a pesquisa bibliográfica convencional, da qual extrairemos o conhecimento adequado para tornar este trabalho suficientemente claro.

1 O que é CORBA

"O *Common Object Request Broker Architecture* (CORBA) é um padrão unificador para escrever sistemas de objetos distribuídos." (ROMAN; AMBLER; JEWELL, 2002). De acordo com Geib e Merle (1997), CORBA é um modelo unificado orientado a objeto que permite integrar os *middlewares* heterogêneos e provenientes de fornecedores diferentes. Em outras palavras, CORBA é um padrão feito com o intuito de tornar o desenvolvimento de sistemas de objetos distribuídos tão simples ao ponto de passar a ser intercambiável, assim a barreira da heterogeneidade de *software* e *hardware* pode ser transposta.

O CORBA surgiu do consórcio *Object Management Group* (OMG), no qual estavam envolvidas inicialmente mais de seis mil instituições internacionais que trabalhavam com *software*, segundo Tari e Bukhres (2001). O objetivo do consórcio era conceber um padrão que proporcionasse interoperabilidade entre os sistemas distribuídos que começavam a ser bastante utilizados pelas empresas em seus projetos. O uso das redes principiava para o grande público, logo a necessidade de partilhar dados se mostrava urgente.

A primeira versão do CORBA (1.0) foi apresentada a comunidade em outubro de 1991 e incluía o modelo de objetos CORBA, a *Interface Definition Language* (IDL) e as APIs para a requisição dinâmica, além disso, havia um mapeamento para a linguagem C. Nos dois anos seguintes, mais duas versões foram apresentadas – 1.1 e 1.2 – ambas traziam algumas melhorias e resoluções de ambiguidades da versão 1.0. Apenas em 1996, três anos depois da última atualização, foi publicada a versão 2.0 do CORBA que trazia bastantes novidades, entre elas: DSI, GIOP, IIOP, suporte para camadas de segurança e mapeamento para as linguagens Smalltalk e C++. Até dezembro de 2001 foram mais cinco versões que compuseram as versões 2.x, nesse tempo foram feitas atualizações importantes e mais linguagens foram mapeadas, incluindo Java, Ada e Cobol. Foram incluídos ao padrão o serviço de portabilidade POA, *core* de tempo real e implementadas medidas de segurança adicionais.

As últimas versões do CORBA (3.x) foram anunciadas no ano de 2002. A versão 3.0 trouxe confirmação para a integração com Java, as especificações para tempo real e tolerância a falha e o uso do protocolo de interoperabilidade IIOP. As versões 3.1 e 3.2, últimas a serem

divulgadas pelo OMG trazem apenas algumas melhorias nas especificações apresentadas anteriormente.

2 Principais conceitos

A estrutura do padrão CORBA é formada por um conjunto de conceitos que devem ser explanados antes de abordarmos as características arquiteturais.

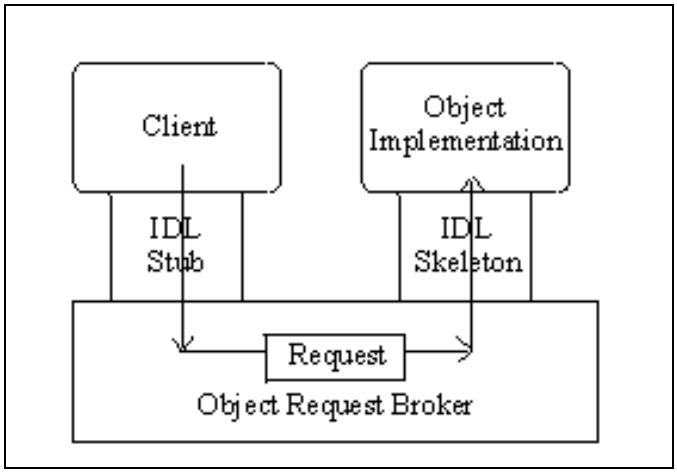

Figura 1 – Exemplo de requisição.
Fonte: www.omg.org

2.1 Object Request Broker (ORB)

O ORB é o componente responsável por todas as nuances da requisição feita da parte do cliente para o objeto e também por encaminhar a resposta devida. "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." (LINK et al., 2007)

O ORB é um elemento estrutural do CORBA que exerce funções diferentes quando invocado pelo cliente ou pelo servidor. Para o cliente ele fornece uma série de serviços, incluindo armazenar as definições da IDL. Já no lado servidor ele executa ações, tais como ativar e reativar objetos de acordo com a demanda de solicitação, preservando assim o uso dos recursos.

Outra característica importante do ORB é que ele fornece todos os serviços de forma transparente, como também garante a interoperabilidade entre as aplicações que o implementa.

2.2 Interface Definition Language (IDL)

De acordo com Killijian (2006), IDL é uma linguagem de definição de interface orientada a objeto. A IDL age diretamente na separação entre a implementação e a interface, proporcionando assim o encapsulamento presente no paradigma orientado a objeto. Como na definição de interface também usada nesse paradigma, a IDL conduz o lado cliente ao reconhecimento das operações disponíveis e de como elas devem ser invocadas.

IDL não é uma linguagem de programação, no entanto, fornece mapeamento para que linguagens de programação sejam usadas pela aplicação CORBA. As linguagens de programação que possuem mapeamento padronizado são: C, C ++, Java, Smalltalk, COBOL, Ada, Lisp, PL, Python, Ruby, e IDLscript. O mapeamento das linguagens é uma funcionalidade fundamental para garantir a interação entre aplicações heterogêneas.

3 Arquitetura

A arquitetura CORBA possui diversos componentes que possibilitam, efetivamente, a comunicação entre os sistemas que a utilizam. Os componentes básicos serão tratados nesta seção.

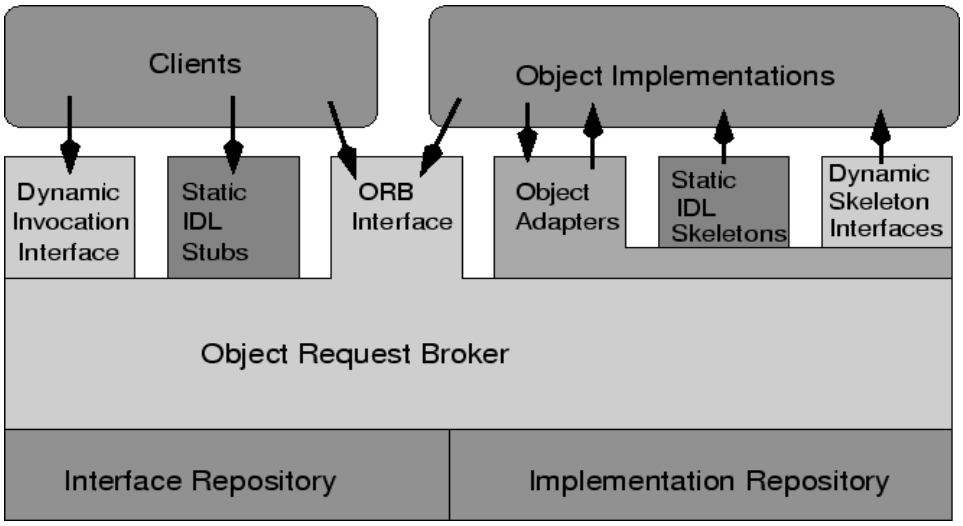

Figura 2 – Componentes da arquitetura CORBA.
Fonte: www.oser.org

3.1 Aplicação cliente

CORBA segue o modelo cliente/servidor, por isso a função de uma aplicação cliente é similar a que conhecemos nesse modelo. A aplicação cliente é que faz as requisições para o servidor, iniciando assim a comunicação. Uma aplicação cliente pode ser escrita em qualquer linguagem mapeada que tenha suporte ORB.

3.2 Aplicação servidora

Assim como citado na seção 3.1, o modelo cliente/servidor também influencia a aplicação servidora que espera uma requisição do cliente para operar. Em suma, o servidor recebe as requisições enviadas pelos clientes, processa o seu conteúdo e devolve uma resposta com os dados solicitados.

3.3 Stub

Para Link et al. (2007), *stub* é a interface de invocação estática gerada a partir da compilação de uma IDL. Por servir para fazer a invocação, o *stub* é um componente que age do lado cliente da aplicação. Em Geib e Merle (1997), vemos que o cliente invoca o *stub* localmente para acessar os objetos, então o *stub* constrói requisições que serão transportadas pelo ORB, e depois entregues ao lado servidor.

3.4 Skeleton

Assim como o cliente tem a sua interface estática (o *stub*), o servidor também conta com a sua, o *skeleton*. Ainda de acordo com Geib e Merle (1997), quando as requisições são encaminhadas pelo ORB para o servidor elas passam primeiro pelo *skeleton*, que fará o encaminhamento em seguida para o objeto servidor.

Skeleton e *stub* são gerados pela compilação da aplicação IDL com a finalidade de intermediarem a comunicação entre servidor e cliente.

3.5 Dynamic Invocation Interface (DII)

A DII é mais uma interface do ORB que tem por finalidade expandir as capacidades da invocação do objeto feita pelo cliente usando a IDL. Isso porque invocar um objeto do sistema a partir de uma IDL é fruto de uma conexão estática, logo qualquer objeto que seja adicionado ao sistema após o estabelecimento da comunicação não poderá ser requisitado. Como a DII é uma interface dinâmica esse potencial problema é superado.

Segundo Killijian (2006), usar a DII permite a aplicação cliente, dentre outras coisas: descobrir novos objetos e suas interfaces, encontrar as definições das interfaces, construir e distribuir as chamadas e receber as respostas. Tornando assim, a execução mais flexível.

3.6 Dynamic Skeleton Interface (DSI)

A DSI é um componente do lado servidor análogo a DII do lado cliente, pois ambas são interfaces que exercem funções similares. Isto é, usando DSI torna-se possível entregar chamadas de métodos para objetos cujas interfaces não são conhecidas em tempo de execução. Tari e Bukhres (2001) ressaltam ainda que o cliente que faz a requisição não toma conhecimento se a implementação está usando o *skeleton* estático gerado pela IDL ou *skeletons* dinâmicos.

3.7 Repositórios

Em CORBA, os repositórios são os componentes que permitem a estocagem dos metadados das interfaces, provendo assim, acessibilidade às coleções de objetos que tenham sido descritos pela IDL. O *Interface Repository* (IFR) e o *Implementation Repository* (IPR) são os dois repositórios presentes na arquitetura.

Tari e Bukhres (2001) destacam que cada repositório se comporta como um "dicionário" provendo meta-descrições sobre informações específicas. Enquanto o IFR age no lado do cliente, o IPR exerce suas funcionalidades do lado do servidor.

3.8 Object Adapter (OA)

Um OA é uma interface responsável por acessar os serviços do ORB. Ele auxilia entregando as requisições do objeto e ativando objetos no servidor. OAs também tem a incumbência de cuidar da política de segurança implementada pelo objeto.

Existem duas categorias de adaptadores mais conhecidas, são elas: *Basic Object Adapter* (BOA) e *Portable Object Adpater* (POA). De acordo com Killijian (2006), algumas das funções do BOA são ativação e desativação de objetos individuais e invocação de métodos. O POA atua auxiliando o ORB na entrega das requisições para os servidores, mas seu principal objetivo é construir aplicações servidoras portáveis entre ORBs diversos. Vale destacar que adaptadores só existem na parte servidora da aplicação.

4 Serviços implementados

Em CORBA, são os serviços que permitem que componentes de aplicações orientadas a objetos interoperem com objetos distribuídos. Killijian (2006) diz que um serviço é caracterizado por interfaces e pelos objetos fornecidos por essas mesmas interfaces. São 16 serviços definidos ainda no conjunto de versões 2.x.

Serviço	Função
Naming	Objetos referenciam e localizam outros objetos usando um nome.
Event	Componentes podem registrar dinamicamente eventos do seu interesse.
Life Cycle	Define convenções para criar/mover/copiar/deletar objetos.
Persistence	Oferece uma interface única para que objetos possam acessar mecanismos de persistência.
Relationship	Cria associações dinâmicas entre componentes que não se conhecem.
Externalization	Grava o estado do objeto externamente de forma que ele possa ser recarregado posteriormente.
Transaction	Conserva a integridade de transações online lhe concedendo confiabilidade.
Concurrency	Concede acesso a apenas um cliente por vez em um sistema de arquivos.
Property	Possibilita a adição de propriedades ao objeto dinamicamente.
Licensing	Implementa serviço de licenciamento, assim apenas alguns clientes podem ter acesso a determinados serviços de um objeto.
Time	Sincroniza relógios no ambiente distribuído.
Trader	Permiter localizar e oferecer serviços a partir de qualquer provedor disponível.
Security	Funcionalidade de segurança, como autenticação e criptografia.
Notification	Extensão do *Event* que adiciona notificações aos eventos desejados.
Collection	Cria conjuntos de objetos semelhantes que serão acessados via índice.
Query	Busca e retorna objetos baseado em condições declaradas.

Tabela 1 – Descrição dos serviços implementados em CORBA.

5 Como usar CORBA

O desenvolvimento de aplicações CORBA não é regido por um padrão único, mas existe determinado consenso, com algumas variações, entre os desenvolvedores em relação à ordem das atividades. Geib e Merle (1997), por sua vez, apresentam uma ordem para os passos do desenvolvimento que segue esse consenso. São eles:

1. Definição dos parâmetros IDL: neste passo, são definidos alguns parâmetros, como os tipos de dados e as interfaces dos objetos.
2. Pré-compilação IDL.
3. Projeção para a linguagem de programação: são gerados os *stubs* (clientes) e *skeletons* (servidores), de acordo com a linguagem de programação escolhida.
4. Implementação das interfaces IDL: nesta etapa os *skeletons* são reutilizados para a implementação dos objetos.
5. Implementação dos servidores: as aplicações servidores, incluindo os objetos e o código para conexão com o barramento, são escritas nesta fase.
6. Implementação das aplicações clientes: são escritas as aplicações clientes, incluindo os *stubs* e os códigos específicos dos clientes que invocarão os objetos.
7. Instalação e configuração dos servidores: neste passo, os servidores são configurados com o intuito de automatizar as respostas às requisições.
8. Difusão e configuração dos clientes: os clientes são configurados, afim de que saibam onde encontrar os servidores.
9. Execução da aplicação distribuída.

6 Quem usa CORBA

CORBA é utilizado em diversos tipos de sistema, de *mainframes* a pequenas aplicações de sistemas embarcados. Costuma-se optar por CORBA quando há a necessidade de gerenciamento de muitos clientes com alto grau de confiabilidade.

CORBA é usado tanto em sistemas de empresas que preservam o seu sigilo quanto em aplicações conhecidas do grande público. “CORBA é usado no GNOME para exportar os mecanismos internos de uma aplicação para o resto do mundo.” (SOUZA; LAGES, 2001). O projeto GNOME usa CORBA, principalmente pela sua capacidade de prover o reuso de componentes e objetos. Como exemplo de exportação de recursos no projeto GNOME, via interface CORBA, temos GMC, Guppi e Gnumeric. O desenvolvimento de aplicações de

tempo real é outra vertente que vem ganhando destaque no âmbito do desenvolvimento com CORBA.

Souza e Mazzola (1999) apresentam em seu artigo a implementação de um banco de dados para uma empresa, o *PitAnita's Hotel*. Nesse trabalho, é abordado o desenvolvimento de cliente e servidor usando CORBA e a linguagem de programação Delphi. Depreendemos daí que a gama de oportunidades de utilização do CORBA é tão grande quanto são as categorias de sistemas distribuídos.

7 Tecnologias similares

No desenvolvimento de aplicações distribuídas, além de CORBA, existem outras tecnologias que lhe oferecem certa similaridade. Em relação a elas, CORBA equilibra algumas vantagens e desvantagens que são mais bem ponderadas apenas quando uma aplicação deve ser projetada.

7.1 CORBA *versus* DCE

O DCE (*Distributed Computing Evironment*) conserva algumas semelhanças em termos de funcionalidades com o CORBA, no entanto, são tecnologias diferentes em sua filosofia, pois enquanto o CORBA foi criado para o paradigma de programação orientado a objeto, o DCE foi criado para o paradigma procedural. Dentre as semelhanças, ambos promovem interação entre ambientes heterogêneos e são baseados no modelo clássico do cliente/servidor. Alguns serviços conhecidos do CORBA, como *Naming*, *Time* e *Security* também existem em DCE. Porém o número de serviços é bastante limitado em relação ao CORBA, outra desvantagem que podemos apresentar é a invocação do objeto feita apenas estaticamente.

7.2 CORBA *versus* DCOM

Assim como o CORBA, o DCOM (*Distributed Component Object Model*) é um *middlware* usado para distribuir componentes de software entre computadores em rede. A primeira diferença entre as duas soluções é que enquanto CORBA é fruto de um consórcio, DCOM é uma evolução de uma solução proprietária da Microsoft, o COM, e, portanto, também pertencente à mesma companhia de *software*. Isso faz com que CORBA seja um padrão já estabelecido na comunidade de desenvolvedores.

Dentre as vantagens que podemos citar do CORBA em relação ao DCOM é a utilização em ambientes heterogêneos, pois o DCOM só está presente em ambiente Windows. O fato do CORBA estar disponível desde o começo da década 1990 também é uma vantagem assinalável. Além disso, CORBA é compatível com diversas linguagens de programação, dentre elas destacamos o Java. CORBA implementa segurança e outros serviços descritos na seção 4 deste trabalho e é bem mais leve que DCOM, sendo possível executá-lo em sistemas com capacidades mais restritas.

7.3 CORBA *versus* RMI

CORBA e RMI (*Remote Method Invocation*) em sua essência são tecnologias feitas para fins diferentes, mas como ambas estão inseridas no mesmo contexto, o desenvolvimento de software distribuído, podemos comparar algumas de suas funcionalidades. Por exemplo, uma das vantagens de CORBA é que os seus serviços são escritos em várias linguagens diferentes e podem ser executados em ambientes diversos, acrescenta-se a isso o mapeamento de linguagens proporcionado pela IDL de CORBA, enquanto RMI só funciona com a linguagem Java. Outra vantagem é que devido à separação entre implementação e interface, a mesma interface pode ser reutilizada.

8 Código demonstração

Licht (2002) apresenta em seu tutorial a demonstração de uma aplicação CORBA que abordaremos nesta seção. O exemplo traz uma aplicação cliente requisitando ao servidor a execução de um método que exibe a mensagem *"Hello world!!"*.

A primeira parte da aplicação é a implementação do módulo IDL, que define a interface.

```
1.  module HelloApp
2.  {
3.     interface Hello
4.     {
5.        string sayHello();
6.        oneway void shutdown();
7.     };
8.  };
```

Figura 3 – Módulo IDL.

A segunda parte do exemplo é a implementação do servidor, onde os métodos descritos no módulo IDL serão definidos. Além disso, a classe inicia o ORB, obtém as referências para o objeto e aguarda ser invocado pela aplicação cliente.

```
1     class HelloImpl extends HelloPOA {
2       private ORB orb;
3
4       public void setORB(ORB orb_val) {
5         orb = orb_val;
6       }
7
8       // implementação do método sayHello()
9       public String sayHello() {
10        return "\nHello world !!\n";
11      }
12
13      // implementação do método shutdown()
14      public void shutdown() {
15        orb.shutdown(false);
16      }
17    }
```

Figura 4 – Implementação do método descrito na IDL.

```java
public class HelloServer {
  public static void main(String args[]) {
    try{
      // Criação e inicialização do ORB
      ORB orb = ORB.init(args, null);

      // obtém a referência do rootpoa e ativa o POAManager
      POA rootpoa = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
      rootpoa.the_POAManager().activate();

      // cria o servant e registra com o ORB
      HelloImpl helloImpl = new HelloImpl();
      helloImpl.setORB(orb);

      // obtém referência do objeto servant
      org.omg.CORBA.Object ref = rootpoa.servant_to_reference(helloImpl);
      Hello href = HelloHelper.narrow(ref);

      org.omg.CORBA.Object objRef =
          orb.resolve_initial_references("NameService");
      // Especifica o servidor de nomes(INS)
      NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);

      // Especifica uma referência de objeto
      String name = "Hello";
      NameComponent path[] = ncRef.to_name( name );
      ncRef.rebind(path, href);

      System.out.println("HelloServer ready and waiting ...");

      // Aguarda a invocação de clientes
      orb.run();
    }
```

Figura 4 – Código da aplicação servidora.

Por fim o código do lado cliente é apresentado, ele obtém a referência para o objeto usando o serviço de *naming*. Então a aplicação invoca o método descrito na IDL que foi implementado pelo servidor.

```java
public class HelloClient
{
  static Hello helloImpl;

  public static void main(String args[])
    {
      try{
        // Cria e inicializa o ORB
        ORB orb = ORB.init(args, null);

        org.omg.CORBA.Object objRef =
        orb.resolve_initial_references("NameService");
        NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);

        String name = "Hello";
        helloImpl = HelloHelper.narrow(ncRef.resolve_str(name));

        System.out.println("Obtained a handle on server object: " + helloImpl);
        System.out.println(helloImpl.sayHello());
        helloImpl.shutdown();

      } catch (Exception e) {
          System.out.println("ERROR : " + e) ;
        e.printStackTrace(System.out);
        }
      }
}
```

Figura 5 – Código da aplicação cliente.

Considerações Finais

A solução para o compartilhamento de objetos em sistemas heterogêneos apresentada pelo CORBA se mostra eficaz. A maneira como CORBA utiliza a abstração de software e hardware facilita bastante o reuso dos componentes. No entanto, CORBA não é largamente

utilizado devido a sua complexidade e diversidade nas versões conhecidas. Há algumas versões que não foram totalmente finalizadas antes da sua sucessora surgir, o que é um obstáculo para quem busca padronizar as suas aplicações. Além disso, a última versão que foi publicada data do ano 2002, o que para alguns indica que a tecnologia está ultrapassada.

Referências

GEIB, J.M.; MERLE, P. **CORBA: des concepts à la pratique.** Disponível em: <http://197.14.51.10:81/pmb/collections/Techniques%20de%20lingenieur/ticd3/h/h2/h2758.pdf> Acesso em: 04 abr 2015.

KILLIJIAN, M.O. **Présentation de CORBA.** Disponível em: <http://corba.developpez.com/presentation/> Acesso em: 04 abr 2015.

LICHT, F.L. **Tutorial CORBA.** Disponível em: <http://virtual01.lncc.br/~licht/tutoriais/> Acesso em: 12 abr 2015.

LINK, E. et al. **Uma introdução ao CORBA.** Disponível em: <https://www.inf.pucrs.br/~gustavo/disciplinas/sd/material/> Acesso em: 04 abr 2015.

ROMAN, E., AMBLER, S.W., JEWELL, T. **Dominando Enterprise JavaBeans.** 2. Ed. Bookman. São Paulo, 2002.

SOUZA, A.D.; LAGES, D.D. **CORBA : Reutilização de Componentes de Software.** Disponível em: <http://www.dcc.ufrj.br/~schneide/es/2001/1/g01/corba.html> Acesso em: 11 abr 2015.

SOUZA, A.C.; MAZZOLA, V.B. **Implementando Aplicações Distribuídas Utilizando CORBA: Um Estudo de Caso Voltado à Área de Banco de Dados.** Disponível em: <http://www.dcc.ufla.br/infocomp/artigos/v1.1/art06.pdf> Acesso em: 11 abr 2015.

TARI, Z.; BUKHRES, O. **Fundamentals of Distributed Object Systems: The CORBA Perspective.** Wiley Series On Parallel and Distributed Computing Albert Y. Zomaya, Series Editor. New York, 2001.

Bibliografia Consultada

BRANDO, T.J. **Comparing DCE and CORBA.** Disponível em: <https://www.mitre.org/sites/default/files/pdf/corba.pdf> Acesso em: 11 abr 2015.

CORBA WEBSITE. **History of CORBA.** Disponível em: <www.corba.org> Acesso em: 02 abr. 2015.

JAVA COFEE BREAK. **Java RMI e CORBA: A comparison of two competing technologies.** Disponível em: <http://www.javacoffeebreak.com/articles/rmi_corba/> Acesso em: 11 abr 2015.

OBJECTIVE INTERFACE SYSTEMS. **What is CORBA?** Disponível em: <http://www.ois.com/Products/what-is-corba.html> Acesso em: 27 mar 2015.

OBJECT MANAGEMENT GROUP. Disponível em: <www.omg.org> Acesso em: 27 mar 2015.

WIKILIVROS. **Sistemas de Informação Distribuídos/Interoperação/Common Object Request Broker Architecture (CORBA).** Disponível em: <http://pt.wikibooks.org/wiki/Sistemas_de_Informa%C3%A7%C3%A3o_Distribu%C3%ADdos/Interopera%C3%A7%C3%A3o/Common_Object_Request_Broker_Architecture_%28CORBA%29> Acesso em: 29 mar 2015.

Questões sobre CORBA

1. Quais as principais características que tornam vantajoso o uso de CORBA?
R.: CORBA é um modelo baseado no paradigma orientado a objeto que proporciona a integração entre ambientes heterogêneos. Além disso, é fruto de um consórcio formado por diversas instituições, o que faz dele um padrão.

2. Qual o conceito do paradigma orientado a objeto implementado pela IDL? Como ela realiza esse conceito?
R.: A IDL realiza o encapsulamento separando a implementação da interface.

3. Defina *stub* e *skeleton* na arquitetura CORBA.
R.: *Stub* e *skeleton* são as interfaces de invocação estática do cliente e do servidor, respectivamente, presentes em CORBA. O *stub* é responsável por encaminhar as requisições do cliente para o servidor, enquanto o *skeleton* recebe as requisições do cliente e as entrega para o objeto servidor. No caminho inverso, quando a resposta é enviada do servidor para o cliente, *stub* e *skeleton* também estão presentes.

4. Os serviços de CORBA formam o mecanismo que permite aos componentes de orientação a objeto interoperarem com objetos distribuídos. Cite dois serviços implementados em CORBA e suas respectivas funções.
R.: *Os serviços estão listados na Tabela 1, seção 4.*