CORBA

Slides baseados no material do professor FELIPE CUNHA

Motivação

- Problema nos anos 90:
  - Cada empresa tinha sua própria tecnologia.
  - Dificuldade de interoperabilidade entre sistemas heterogêneos.

- Objetivo do CORBA:
  - Permitir que objetos escritos em diferentes linguagens ou plataformas se comuniquem.
  - Fornecer um middleware padronizado e aberto.

CORBA

CORBA: Common Object Request Broker Architecture

- Padrão para desenvolvimento de aplicações distribuídas para sistemas heterogêneos usando orientação por objetos

OMG: consórcio de empresas responsável pela proposição e manutenção do padrão

- Criado em 1989
- Atualmente, com mais de 800 empresas

OMA: Object Management Architecture

- Visão da OMG de como deve ser a arquitetura básica de um sistema distribuído
  - Primeira especificação importante produzida pela OMG

Hoje: usado principalmente em sistemas legados críticos (defesa, aeroespacial, telecom).

ORB

ORB (Object Request Broker): “barramento lógico de software” que controla toda a comunicação entre os componentes da arquitetura

- Objetivo: prover transparência de localização, de implementação e de acesso
- Ajudar um cliente a invocar um método em um objeto (seguindo o estilo RMI)

CORBA:

- Especificação de um ORB

Implementações do padrão CORBA:

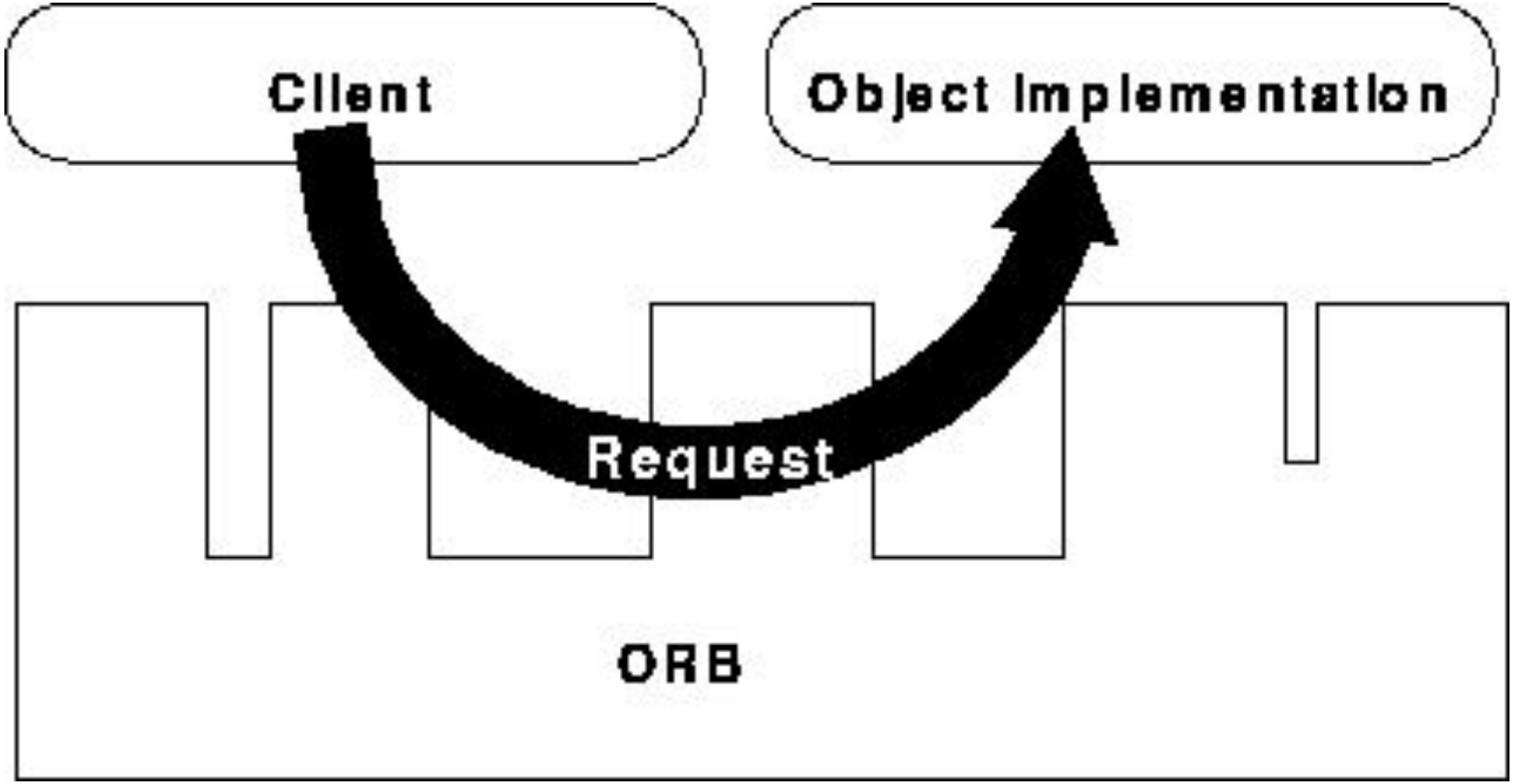
- Java 2 ORB
- Visibroker for Java (da empresa Inprise)
- OrbixWeb (da empresa IONA Technologies)
- Netscape Communicator
- Voyager (da empresa ObjectSpace)

CORBA

Java RMI vs CORBA

- O programador define interfaces remotas e depois usa um compilador para produzir os proxies (stubs) e esqueletos.
  - No CORBA, os proxies são gerados na linguagem do cliente e os esqueletos, na linguagem do servidor.
- Os clientes não são necessariamente objetos, podendo ser qualquer programa que envie mensagens de requisição para objetos remotos e receba respostas.
- Um objeto CORBA pode ser implementado por uma linguagem não orientada a objeto.
  - Instâncias de classes não podem ser passadas como argumentos: usa-se estruturas de dados de vários tipos e complexidade arbitrária.

CORBA: Visão Geral



IDL: Interface Definition Language

Linguagem para definir a interface de um objeto

- Assinatura das operações que são implementadas por um objeto

Linguagem declarativa (sem estruturas de controle)

- Tipos pré-definidos: long, short, float, double, char, boolean, octet (8 bits), etc.
- Tipos estruturados: struct, string, sequence, array, etc.
- A IDL CORBA tem as mesmas regras léxicas da linguagem C++, mas possui palavras-chave adicionais para suportar distribuição, por exemplo, *interface*, *any*, *attribute*, *in*, *out*, *inout*, *readonly* e *raises*.

IDL: Interface Definition Language

Compiladores de IDL geram automaticamente

- stubs (clientes)
- skeletons (servidor)

Objetivo: garantir independência de linguagem de programação

- Compiladores IDL implementam geração de código para Java, C, C++, Python, Smalltalk, Ada, COBOL…

IDL: Interface Definition Language

```
struct Rectangle{
        long width;
        long height;
        long x;
        long y;
};
struct GraphicalObject {
        string type;
        Rectangle enclosing;
        boolean isFilled;
};
interface Shape {
        long getVersion( );
        GraphicalObject getAllState( );     // retorna o estado do objeto GraphicalObject
};
typedef sequence <Shape, 100> All;
interface ShapeList {
        exception FullException{ };
        Shape newShape(in GraphicalObject g) raises (FullException);
        All allShapes( );               // retorna a sequência de referências de objeto remoto
        long getVersion( );
};
```

Figura 8.2 Interfaces IDL *Shape* e *ShapeList*.

CORBA

Semântica de invocação: tem como padrão a semântica de chamada no máximo uma vez.

- a IDL pode especificar que a invocação de um método em particular tenha semântica *talvez*, usando a palavra-chave oneway.

Suporta passagem por valor de objetos que não são CORBA

- São parecidos com objetos, pois possuem atributos e métodos. Entretanto, eles são puramente objetos locais.
- Isso é obtido pela adição, na IDL, de um tipo chamado *valuetype* para representar objetos que não são CORBA.
  - *valuetype* é uma struct com assinaturas de método adicionais (como as de uma interface) e os argumentos e resultados valuetype são passados por valor.

Arquitetura CORBA

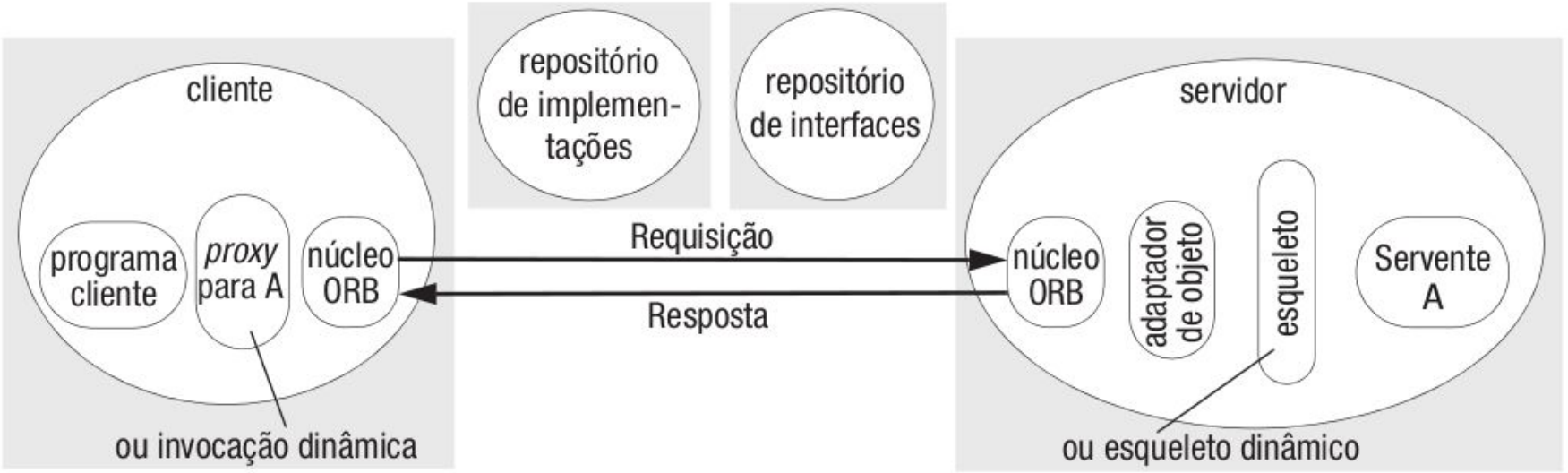


Figura 8.5 Os principais componentes da arquitetura CORBA.

Arquitetura CORBA

Núcleo ORB - inclui toda a funcionalidade do módulo de comunicação

Adaptador de objeto portável (POA) - permite que aplicações e serventes sejam executados em ORBs produzidos por diferentes desenvolvedores.

- separa a criação de objetos CORBA da criação dos serventes que implementam esses objetos

Arquitetura CORBA

Esqueletos - enviam invocações de método remotas para um servente em particular, desempacotando os argumentos nas mensagens de requisição e empacotando exceções e resultados nas mensagens de resposta.

Proxies - são os *stubs* dos clientes, que funcionam de forma similar ao esqueleto.

Repositório de implementações - mantém um mapeamento entre nomes de objetos/adaptadores e a implementação real do servidor.

Repositório de interfaces - fornece informações sobre as interfaces IDL registradas para clientes e servidores que as exigirem.

Aplicações Cliente/Servidor em CORBA

Passo 1: Definir a interface dos objetos remotos em IDL

Passo 2: Compilar a definição em IDL

- Geração automática de stubs (clientes) e skeletons (servidores)

Passo 3: Implementação do servidor

- Definição de classes que implementem as interfaces especificadas em IDL
- Implementação da função main do servidor

Passo 4: Implementação do clientes

- Chamadas a métodos dos objetos remotos (localizados no servidor)

Passo 5: Compilação, linkedição e ativação do servidor e do cliente

CORBA em Java

Java IDL é a tecnologia para criação de objetos distribuídos no padrão CORBA.

Cliente

- Aplicação inclui uma referência ao objeto remoto: possui um método stub que mapeia a chamada remota.
- Stub está conectado a uma ORB, que provê a conexão e invocação do servidor.

Servidor

- ORB usa o código do skeleton e traduz chamada remota à chamada local.

ORB

- Se comunicam pela internet através do Internet Inter-ORB Protocol, IIOP, baseado em TCP/IP.

Desenvolvendo Java IDL

Definir a Interface Remota

- Deve ser definida na linguagem IDL padrão.
- idlj gera o stub e o skeleton.

Compilar a Interface Remota

- O compilador idlj também produz a nova interface que será usada como interface remota.

Implementar o servidor

- Deve incluir mecanismos para iniciar a ORB e se registrar a ela onde ficará esperando requisições do cliente

Implementar o cliente

- Clientes devem possuir uma ORB onde se conectarão para buscar uma referência ao Servidor.

Escrevendo Interface IDL

OMG IDL é uma linguagem puramente declarativa

Passo 1: Declarar o módulo CORBA IDL

- É o módulo container das interfaces e declarações.

Exemplo: arquivo Hello.idl.

```
module HelloApp {
    interface Hello
    {
        string Sayhello();
    };
};
```

Compile com o idlj: idlj -fall Hello.idl

Escrevendo Interface IDL

Serão gerados 6 arquivos.

Hello.java e HelloOperations.java
- Compõem a interface remota.

HelloPOA.java
- Classe Abstrata que implementa o Skeleton no Servidor.

HelloStub.java
- Classe que implementa o stub do Cliente.

HelloHelper.java
- Classe Final que provê type casting em CORBA através do método narrow().

HelloHolder.java
- Classe Final que mantém operações de entrada e saída para os manipulação de parâmetros java.

Pacotes CORBA

O cliente e o servidor CORBA devem importar os seguintes pacotes:

- import HelloApp.*;
  - Pacote que contém a interface remota gerada pelo idlj
- import org.omg.CosNaming.*;
  - Pacote contendo o serviço de nomes que será usado
- import org.omg.CORBA.*;
  - Pacote contendo as classes CORBA
- import org.omg.CosNaming.NamingContextPackage.*
  - Pacote contendo exceções especiais para serviço de nomes.

Cliente CORBA

Criando uma ORB no cliente:

- ORB orb = ORB.init(args, null);
  parâmetros do init: argumentos (pode ser null) e propriedades específicas da aplicação

Usando o serviço de nomes:

- Busca uma referência remota genérica para o servidor de nomes.

```
org.omg.CORBA.Object objRef =
  orb.resolve_initial_references("NameService");
```

- Faz type casting no objeto genérico e converte-o para um objeto de serviço de nomes.
- `NamingContext ncRef = NamingContextHelper.narrow(objRef);`

Cliente CORBA

Buscando a referência para o Servidor:

- O método resolve faz a resolução de nomes no servidor de nomes e retorna uma referência para um objeto genérico. É necessário então se fazer o Type Casting para convertê-lo ao servidor real.

```
Hello helloRef = HelloHelper.narrow(ncRef.resolve(path));
```

Já podemos usar nosso servidor!

Servidor CORBA

Um Servant é um serviçal, ou seja, a classe que abriga o objeto remoto.

A classe Servant é o objeto remoto a ser registrado na ORB.

Ela deve Herdar de HelloPOA

```
class HelloServant extends HelloPOA
{
    public String sayHello()
    {
        return "\nAlo Mundo !!\n";
    }
}
```

Servidor CORBA

O servidor CORBA (HelloServer) possui código análogo ao cliente para registro da ORB e do serviço de nomes.

Após montar a referência para o objeto, deve-se inseri-lo no serviço de nomes:

```
ncRef.rebind(path, helloRef);
```

Agora basta esperarmos requisição do cliente:

```
orb.run();
```

Usando a String IOR

Normalmente, para interoperabilidade entre ORBs, podemos não utilizar o serviço de nomes.

É possível se utilizar a referência de um objeto CORBA (IOR) em forma de String para encontrá-lo na rede.

Método

```
String orb.object_to_string(helloRef);
```

- Produz IOR de um servidor registrado na ORB.

Método

```
org.omg.CORBA.Object =
  orb.string_to_object(IOR);
```

- Recupera referência de um servidor registrado na ORB a partir de sua IOR.
- Deve-se lembrar de usar o narrow para converter o servidor.

IOR: Interoperable Object Reference

Referências para objetos possuem três partes:

- Repository ID: repositório contém uma descrição da interface do objeto. Usado principalmente em chamadas dinâmicas
- Endpoint Info: contém informações necessárias para estabelecer uma conexão com o servidor. No caso de IIOP, o endereço IP e a porta TCP do servidor
- Object Key (ou Object ID): identifica unicamente um objeto no servidor

Formato IOR

ID de tipo de interface IDL	Protocolo e detalhes do endereço			Chave de objeto	
identificador de repositório de interfaces	IIOP	nome de domínio do *host*	número da porta	nome do adaptador	nome do objeto

Conceitos e Terminologia

Objetos CORBA:

- Objeto "virtual" capaz de ser localizado pelo ORB e de receber requisições de clientes

Objetos servidores (Servant):

- Objeto que implementa ("encarna") um ou mais objetos CORBA

Servidor:

- Aplicação que hospeda objetos CORBA

Conceitos e Terminologia (cont.)

Referência para Objetos

- Usadas para identificar, localizar e executar operações sobre objetos CORBA
- Padrão: Interoperable Object Reference (IOR)

Objetos Adaptadores (Object Adapters):

- Funcionam com uma "cola" entre o ORB e objetos servidores (servant).
- Redirecionam requisições recebidas do ORB para o objeto servidor adequado.
- Padrão: Portable Object Adapter (POA)

Interoperabilidade entre ORBs

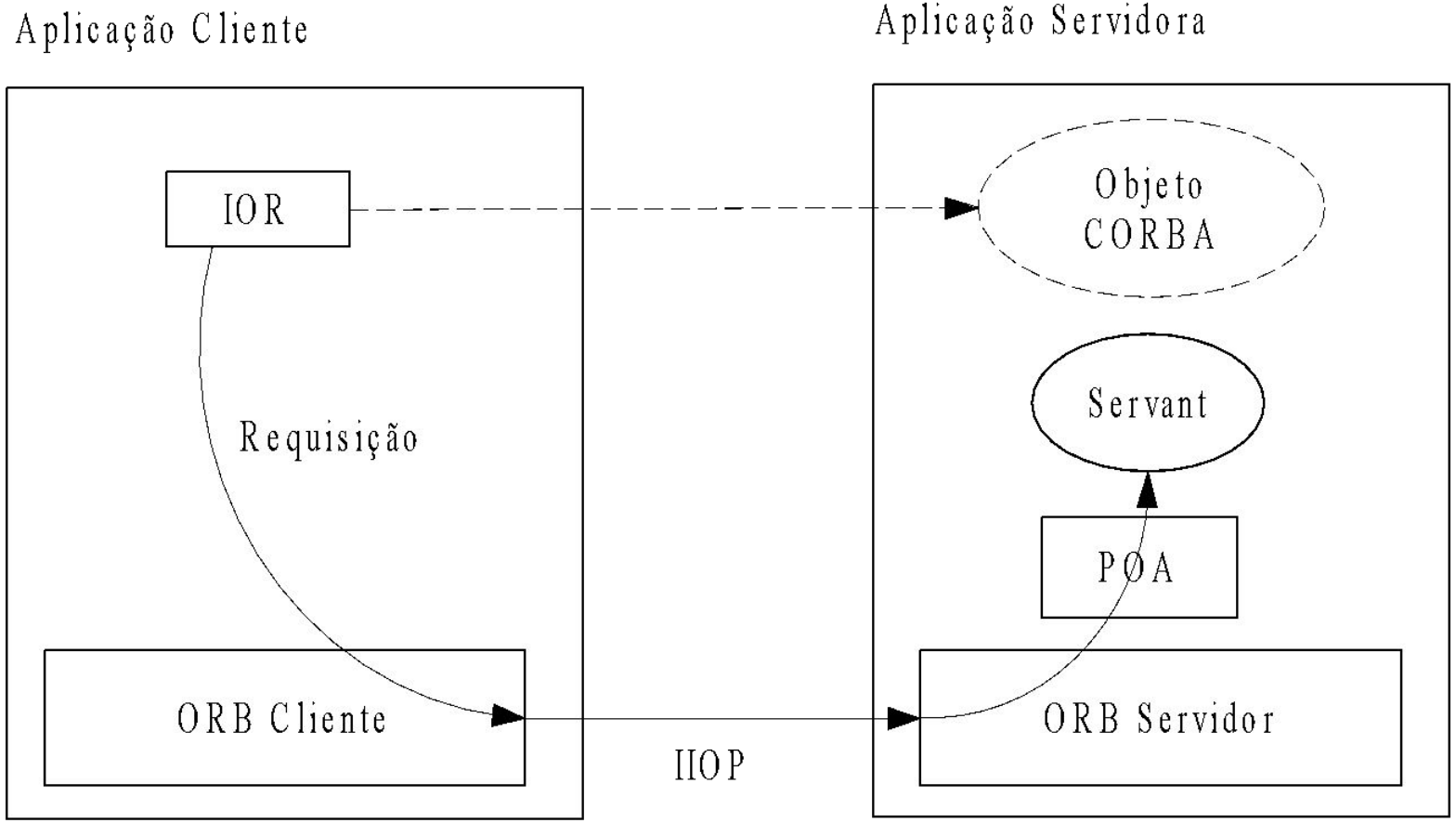
Antes de CORBA 2.0: não era possível

- ORBs de fabricantes diferentes não "conversavam" entre si

CORBA 2.0: definição de um protocolo para troca de mensagens entre ORBs

- GIOP: General Inter-ORB Protocol
  - Definição abstrata para qualquer protocolo de transporte
- IIOP: Internet Inter-ORB Protocol
  - Implementação do GIOP sobre TCP/IP

Envio de Mensagens



Interface de Invocação Dinâmica

DII: Dynamic Invocation Interface

Permite a um cliente escolher dinamicamente a operação que deseja executar

- Para isso, cliente consulta um Repositório de Interfaces (objeto que armazena dados sobre as interfaces de todos os objetos registrados no sistema)
- Marshalling dos parâmetros fica a cargo do programador

CORBA Atualmente

- Ainda usado em:
  - Aviação, defesa, telecom, bancos, sistemas embarcados.
- Substituído por:
  - REST/gRPC em novos sistemas.
- Importância didática:
  - Mostra a evolução dos middlewares distribuídos.

CORBA: Considerações Finais

Vantagens:

- Viabiliza a construção de sistemas distribuídos usando conceitos de OO
  - Maior nível de abstração
  - Independente da infra-estrutura de rede
- Ênfase em interoperabilidade
- Aberto (padrão)

Desvantagens:

- Curva de aprendizado
- Atualmente: Substituído em muitos cenários por Web Services e gRPC.

Conclusão

CORBA foi um marco na padronização de sistemas distribuídos.

Trouxe avanços em interoperabilidade e abstração.

Apesar de não ser mais dominante, é essencial para entender a evolução histórica das arquiteturas distribuídas.