
# CORBA

## *Common Object Request Broker Architecture*

Orientador Prof. Marcos José Santana

Aluno Iran Calixto Abrão

# OMG

- Object Management Group
  - consórcio de empresas
  - reúne cerca de 800 empresas, contando com muitas das maiores empresas de software do mundo
- OMG definiu um conjunto de
  - especificações
  - arquiteturas e
  - ferramentas

  que são coletivamente chamadas de CORBA (*Common Object Request Broker Architecture*)

# Empresas que adotaram CORBA

***2AB***
**ABN Amro**
**Adaptive**
**Alcatel**
**BEA**
***Borland***
**Boeing**
**CA**
**Compuware**
**DaimlerChrysler**
**Ericsson**
**Fujitsu**
**Glaxo SmithKline**
**Hewlett Packard**
**Hitachi**
**Hyperion**
**IBM**
***IONA***
**io Software**
**Kennedy Carter**
**MITRE**
**NASA**
**NEC**
**NIST**
**NTT DoCoMo**
**Northrop Grumman**
**OASIS**
**Oracle**
***PRISM***
**SAP**
**SAS Institute**
**Siemens**
**Sony**
**Softeam**
**Sun**
**Unisys**
**Visa**
**W3C**
**LION Bioscience**
**John Deere**

Fonte: *Middleware that Works*. R. Soley. (www.omg.org)

# Implementações de CORBA

- 2AB orb2
- AT&T OmniORB
- BEA WebLogic Enterprise
- Borland Visibroker
- Deutsche Telekom MICO
- Fujitsu ObjectDirector
- Gerald Brose JacORB
- Hitachi TPBroker
- Harvard Arachne
- IBM WebSphere
- IONA Orbix
- Lockheed Martin HardPack
- Lotus Notes & Domino
- NEC ObjectSpinner
- Netscape Navigator
- OIS ORBExpress
- Oracle 8i & 11i
- PRISM Technologies e*ORB
- Promia SmalltalkBroker
- Red Hat ORBit
- Sun Java, EJB & J2EE
- Washington University TAO
- JacOrb Etc.

Fonte: *Middleware that Works*. R. Soley. (www.omg.org)

# OMA - *Object Management Architecture*

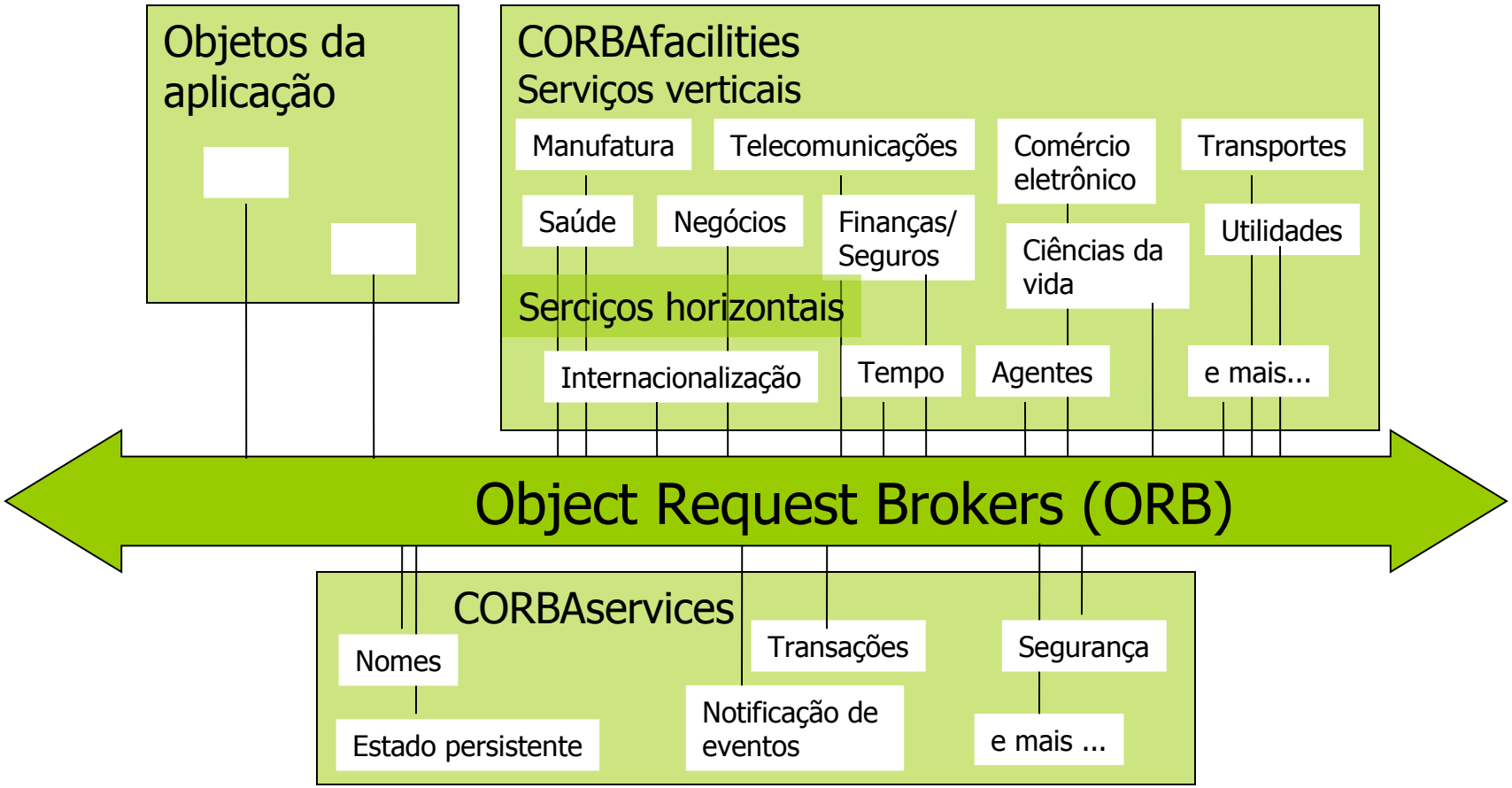
- CORBA é utilizado para conectar objetos entre si
- A integração de aplicações é tratada por uma arquitetura mais abrangente da OMG

  Arquitetura de Gerenciamento de Objetos – OMA (*Object Management Architecture*)
- Através da OMA, a OMG especifica sua visão dos componentes de software de um ambiente distribuído orientado a objetos

# OMA - *Object Management Architecture*

- A OMA é um modelo de referência que padroniza interfaces de infra-estruturas e serviços para uso em aplicações

# OMA - *Object Management Architecture*



# O que o CORBA oferece ?

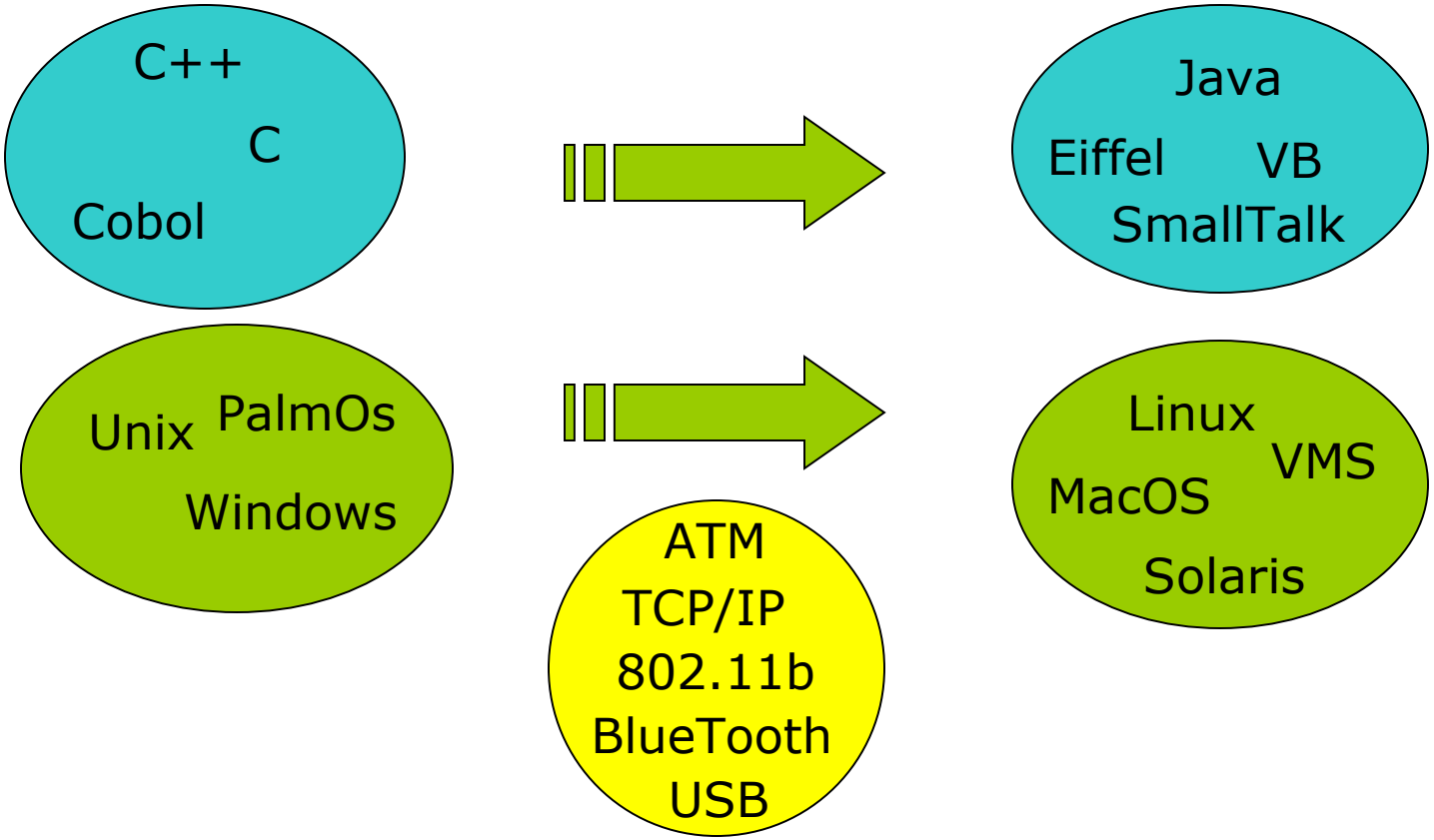
## ou: *Por que usar CORBA?*

- Suporte à heterogeneidade
- Interfaces de objeto padronizadas
- Independência de linguagem de programação
- Transparência de localização
- Ativação de servidores
- Geração automática de código
- Serviços e facilidades padronizados

# Suporte à heterogeneidade

- Facilita a programação de aplicações em ambientes distribuídos heterogêneos
  - Linguagens de programação diferentes
  - Sistemas operacionais diferentes
  - Protocolos de comunicação diferentes

# Suporte à heterogeneidade



# Suporte à heterogeneidade

- Não existe consenso em relação a
  - Linguagens de programação
  - Sistema operacional
  - Protocolos de comunicação
  - Arquiteturas de hardware
- Mas deve haver consenso em relação a
  - Modelos
  - Interfaces e
  - Interoperabilidade

# Suporte à heterogeneidade

- Heterogeneidade é natural em aplicações de larga escala
  - Internet
  - Comércio eletrônico
  - *M-commerce*
  - Grandes corporações
  - Ambientes industriais

# Interfaces IDL

- A engenharia de software preconiza a separação entre
  - Interfaces de objetos, e
  - Implementações de objetos
- Interfaces são os aspectos visíveis de classes de objeto
  - O quê, para um cliente, é interessante em um objeto remoto?

# Interfaces IDL

- Separação entre interface e implementação permite gerenciar a evolução de software
  - Múltiplas implementações de uma mesma interface
- Interfaces podem ser estendidas por herança
- Interfaces também existem no Java
- Porém
  - IDL adiciona modificadores acesso de parâmetros (in, out e inout) e invocações oneway
  - IDL é independente de linguagem de programação

# Interfaces IDL

- Alguns aspectos de classes em UML (*Unified Modeling Language*) podem ser expressas em IDL

- Algumas ferramentas proveem mapeamento automático de UML para IDL

# Independência de Linguagem

- IDL pode ser mapeado em diversas linguagens de programação
- Antigamente, chamadas (invocações) remotas eram disponibilizadas por bibliotecas em uma determinada linguagem
- Com CORBA, cada objeto pode ser implementado em uma linguagem distinta

# Independência de Linguagem

- Duas preocupações
  - Envelopar (*wrapping*) aplicações legadas
  - Esconder heterogeneidade de linguagem e plataforma
- Entretanto é preciso um ORB para a linguagem a ser utilizada

# Transparência de localização

- Aplicações baseadas em soquetes e em URLs acessam um servidor especificando
  - um nome de *host*
  - um número de porta

- Com CORBA, um objeto
  - É identificado independente de sua localização física
  - Objeto pode mudar de localização sem "quebrar" a aplicação

# Ativação de Servidores

- CORBA oferece flexibilidade na configuração de servidores
- Adaptador de Objeto Portável

  POA – *Portable Object Adapter*
- POA permite ajustar políticas de comportamento de servidores
  - *Threading*
  - Ativação de objetos
  - Gerenciamento de ID de objetos
  - Ciclo de vida de objetos

# Geração automática de *stubs* e esqueletos

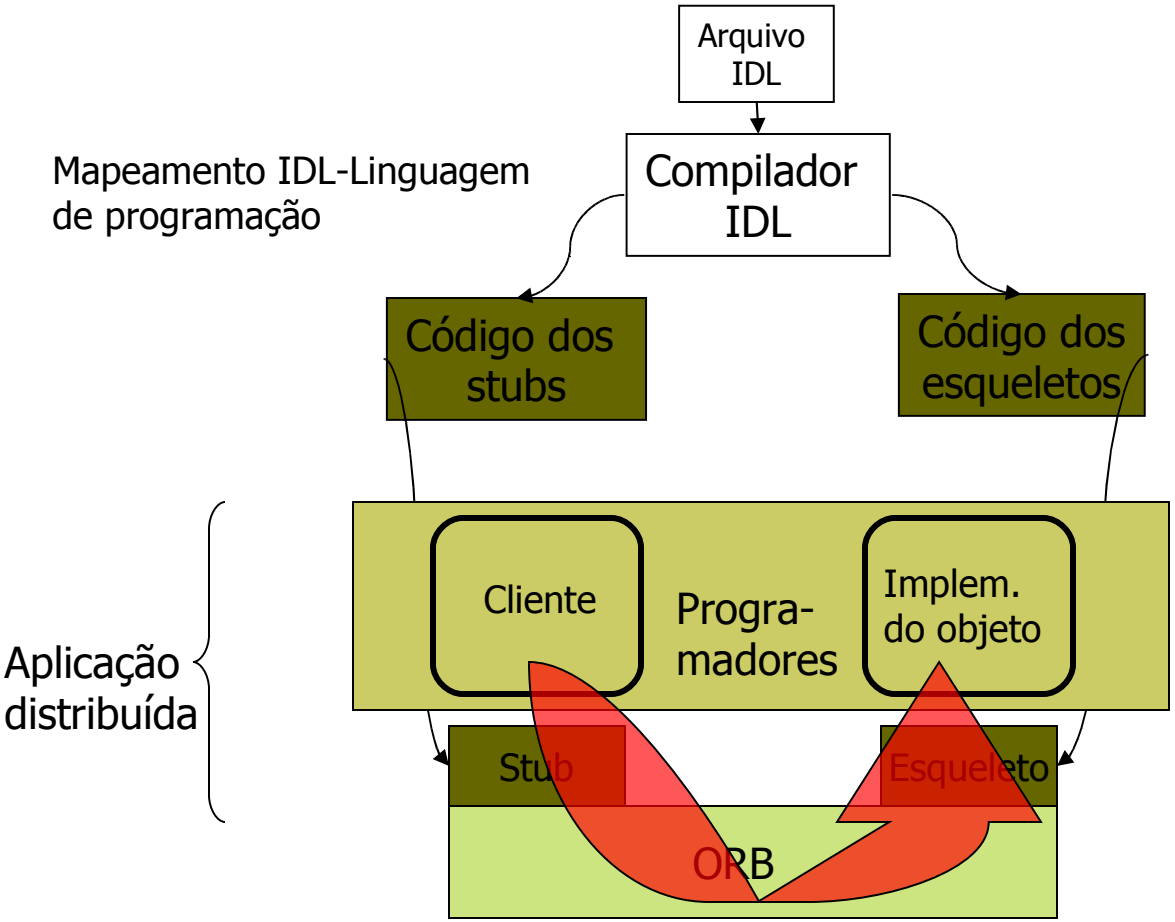
- Comunicação requer esforço de programação repetitivo
  - Abertura, controle e fechamento de conexões
  - Linearização (*marshalling*) e deslinearização (*unmarshalling*) de dados
    - Consideranto formatos de arquiteturas distintas
    - Nos dois sentindos (ida e volta)
  - Configuração de servidores para
    - Escuta de requisições que chegam as portas de soquetes
    - Encaminhamento de requisições aos objetos

# Geração automática de *stubs* e esqueletos

- O CORBA libera o programador deste esforço
  - Compiladores IDL
  - Sistemas *run-time* (ORB)

- Compiladores IDL - mapeamento
  - Cria representações de construções definidas em IDL em representações de linguagem de programação
  - Gera código de linearização e deslinearização
    - Cliente: *stub*
      - Código que invoca a operação de um objeto remoto
    - Servidor: esqueleto (*skeleton*)
      - Código que, com o ORB, provê mecanismos de *run-time* para o tratamento de invocações que chegam

# Geração automática de *stubs* e esqueletos



# Serviços e Facilidades

- O ORB provê apenas meios para a invocação de objetos potencialmente remotos, de maneira transparente
- Aplicacoes distribuídas requerem outras funcionalidades
- No CORBA estas funcionalidades são padronizadas
  - CORBAservices e
  - CORBAfacilities

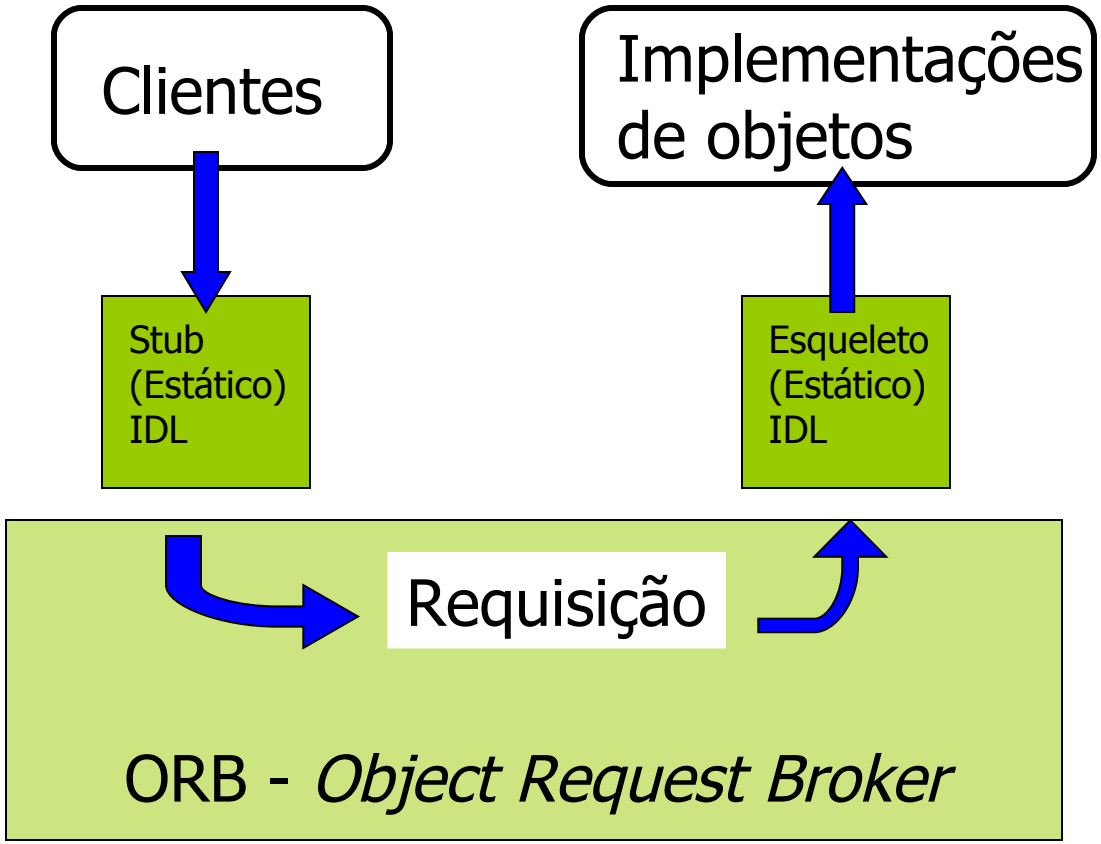
# Serviços e Facilidades

- Alguns CORBAservices
  - *Naming Service*: "catálogo" de objetos
  - *Trading Service*: "páginas amarelas" para objetos
  - *Notification Service*: notificações assíncronas baseadas em assinaturas
  - *Security Service*: autenticação, autorização, criptografia, etc.

# Arquitetura CORBA

- Visão geral
- Estrutura de um ORB
- OMG-IDL
- Interfaces do ORB
- Adaptador de Objeto Portável
- Arquitetura de interoperabilidade

# Visão Geral


Clientes
Implementações de objetos
Stub (Estático) IDL
Esqueleto (Estático) IDL
Requisição
ORB - *Object Request Broker*

# Visão Geral

- Tanto o *cliente* quanto a *implementação de objeto* estão separados através de uma interface definida em IDL (*Interface Definition Language*)
  - Os clientes vêem apenas a interface dos objetos, nunca os detalhes de implementação destes objetos

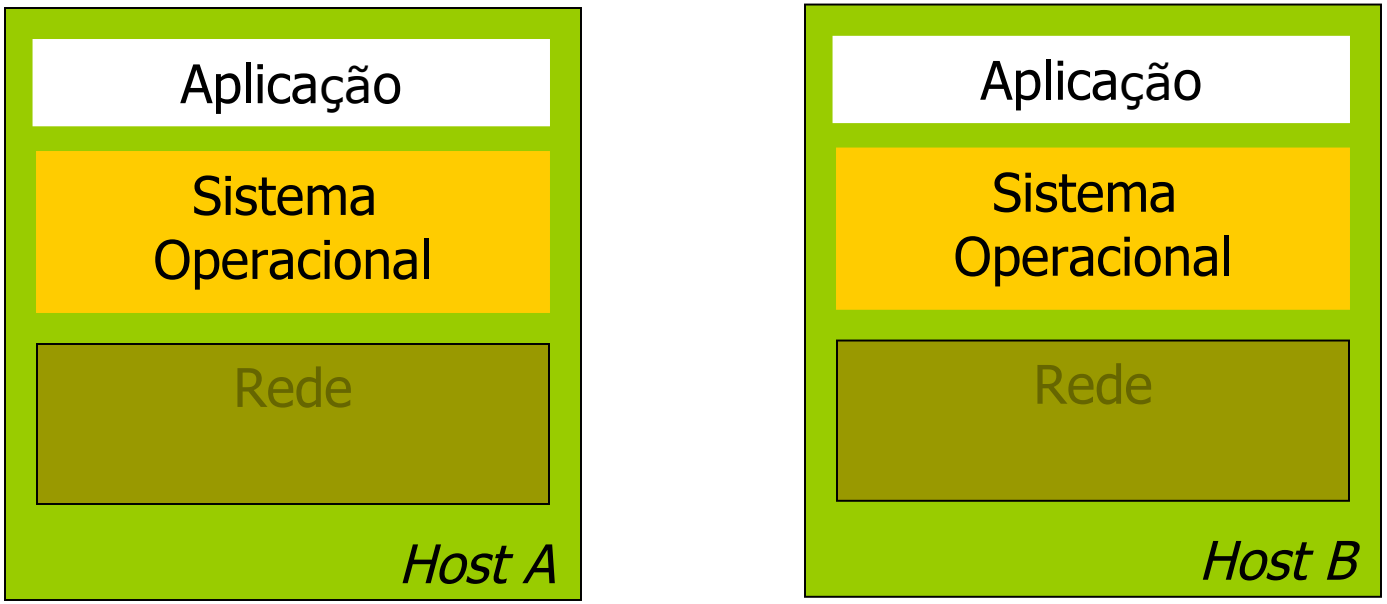
- Isto possibilita a substituição de implementações de uma dada interface

# Visão Geral

- As requisições de operações de objetos geradas pelos cliente são passadas para a implementação do objeto, através do ORB local ao cliente
- A forma padrão de invocação oferece transparência de acesso e localização
  - a sintaxe associada à invocação do objeto é a mesma, quer a implementação do objeto seja local (no mesmo espaço de memória) ou remota
  - Os efeitos da localização do objeto ficam ao encargo do ORB
- Assim, o programador da aplicação se preocupa com problemas do domínio da aplicação, e não com problemas de transporte de informações

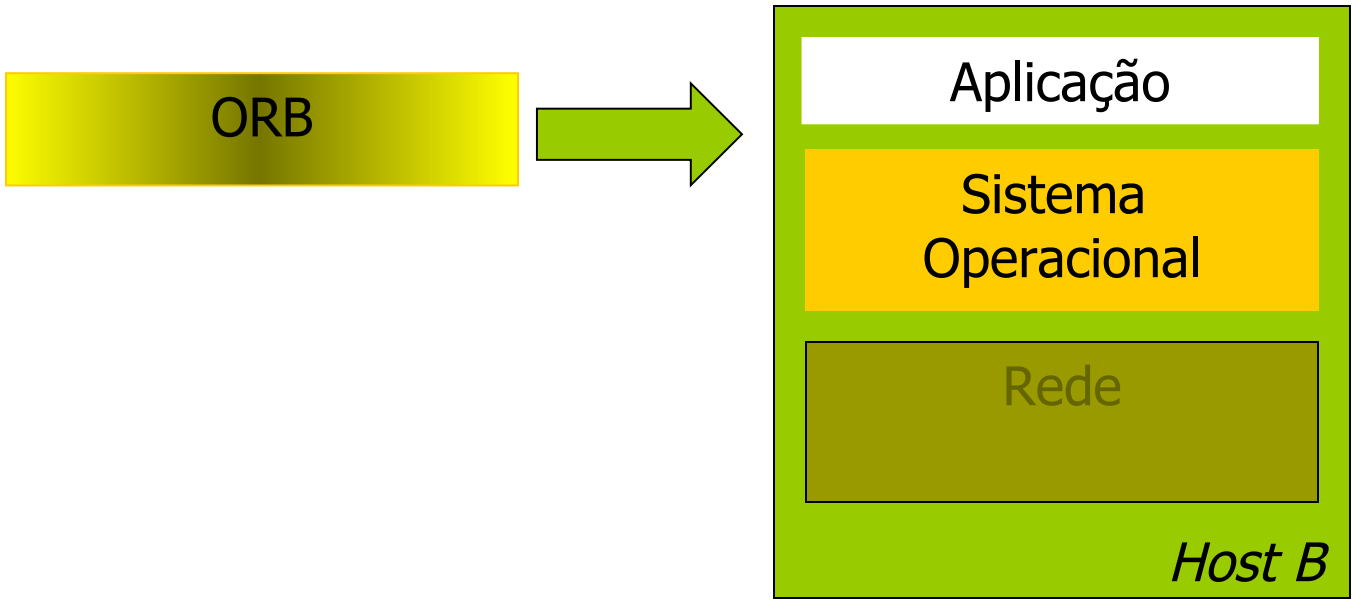
# Estrutura de um ORB

- ORB: *Object Request Broker*
- Arquitetura distribuída tradicional



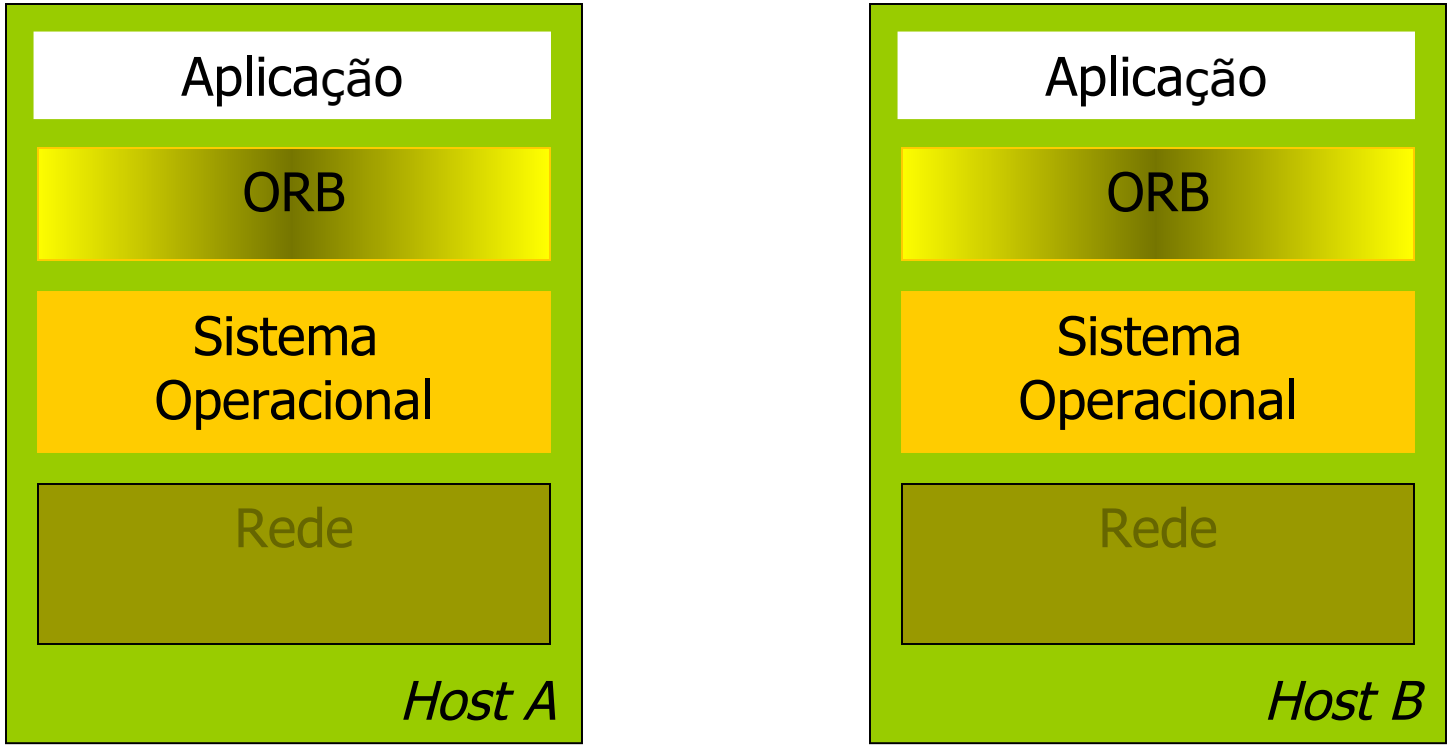
# Estrutura de um ORB

- ORB é um *middleware*

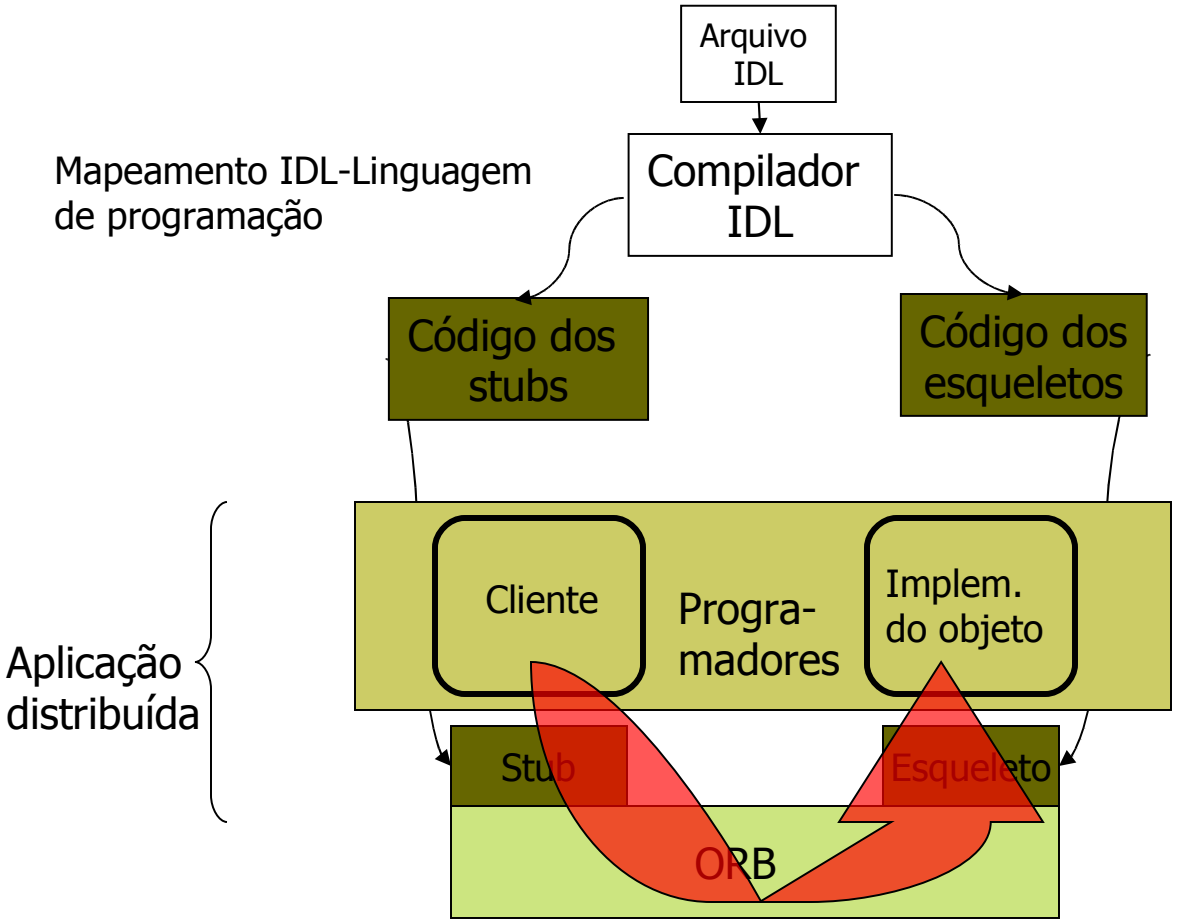


# Estrutura de um ORB

- ORB é um *middleware*



# Estrutura de um ORB - Invocações Estáticas



# Estrutura de um ORB - Invocações Estáticas

- *Stub*
  - Código gerado pela compilação de um arquivo IDL
    - É escrito na linguagem utilizada pelo cliente (mapeamento IDL-linguagem de programação)
  - Permite a um programa cliente chamar uma operação através de uma referência de objeto, como se fosse um método local
  - É instanciado como um objeto *proxy* local
  - O stub faz o *marshalling* de uma chamada de operação
  - E interage com o ORB local para encaminhar a chamada a um método da implementação do objeto

# Estrutura de um ORB - Invocações Estáticas

- Esqueleto (*skeleton*)
  - Código gerado pela compilação de um arquivo IDL
    - É escrito na linguagem utilizada pelo servidor (mapeamento IDL-linguagem de programação)
  - Permite a um programa ao servidor, que suporta um ou mais objetos, executar a operação correta da implementação de objeto correta
  - o ORB local interagem com o esqueleto para executar a chamada a um método da implementação de objeto
  - O esqueleto faz o *unmarshalling* da chamada da operação

# Estrutura de um ORB - Invocações Estáticas

- O ORB é responsável pelo transporte das chamadas executadas sobre os *stubs* (no cliente) até o *esqueleto* (no servidor)

# OMG-IDL

- IDL - *Interface Definition Language*
- A interface de um objeto especifica as operações que um objeto pode executar
- É uma linguagem declarativa
- É um padrão da OMG (CORBA 2.2), ISO (14750) e ITU-T

# OMG-IDL

- Provê uma separação (propositalmente) da interface de um objeto de sua implementação
- Encapsulamento
  - Imagine um programa C++ querendo acessar um método de um objeto escrito em Smalltalk, e vice-versa
  - Esta flexibilidade é oferecida se os objetos tem suas interfaces definidas em IDL

# OMG-IDL

- Uma interface IDL pode ser vista como um contrato:
  - Para um cliente, a interface de uma objeto representa uma promessa
    - Uma invocação de um método deve retornar um resultado
  - Para um servidor que implementa um objeto, a interface representa uma obrigação
    - O servidor precisa implementar a interface em alguma linguagem de programação

# OMG-IDL

- Mapeamento IDL-Linguagem de programação
- A OMG especifica o mapeamento de interfaces IDL em diversas linguagens de programação
- Alguns fabricantes desenvolveram mapeamentos proprietários (o que a OMG permite) para Objective C, Eiffel, Perl, etc.

OMG-IDL

| | | |
|---|---|---|
| C++ | Java | Smaltalk |
| C | Cobol | Ada |
| Lisp | Python | IDL Script |

# OMG-IDL

- Cada mapeamento especifica como os tipos IDL, operações e outras construções são convertidos em tipos e chamadas de funções ou métodos
  - Estes métodos (ou funções) são implementados nos esqueletos
- Este mapeamento é realizado por um compilador IDL oferecido pelo fornecedor do ORB
- A implementação de uma interface pode ser feita em qualquer das linguagens para a qual existe um mapeamento

# OMG-IDL

- Sintaxe “inspirada” pelo C++
- Incorpora construções que permitem declarações de
  - Constantes
  - Tipos de dados (em parâmetros)
  - Atributos
  - Operações
  - Interfaces
  - Tipos valorados (*ValueType*)
  - Módulos

# OMG-IDL – Módulos e Interfaces

- Módulos são utilizados para evitar conflitos de nomes
- Para evitar conflitos, módulos definem escopos de nome
- Módulos podem conter qualquer descrição IDL bem-formada, inclusive outros módulos

# OMG-IDL – Módulos e Interfaces

- O objetivo de uma descrição IDL é a definição de interfaces e suas operações
- Uma interface também define um novo escopo de nomes
- Interfaces contêm
  - Constantes
  - Declarações de tipos de dados,
  - Atributos e
  - Operações

# Arquitetura de Interoperabilidade

- Interoperabilidade
  - Habilidade de um cliente de um dado ORB de invocar as operações definidas em OMG-IDL em objetos de outros ORBs
- Considera que os ORBs foram desenvolvidos de maneira independente
  - Diferentes fabricantes (IONA, Borland, etc.)
  - Diferentes sistemas operacionais
  - Diferentes protocolos de comunicação
  - Diferentes requisitos de segurança

# Arquitetura de Interoperabilidade

- Interoperabilidade é alcançada através de
  - Pontes (*Bridges*)
  - Protocolos
  - IOR (*Interoperable Object Reference*)

# BomDia.idl

```
module Exemplo1
{
   module Ola {

      interface BomDia {

         string SimplesOla();

      };

   };

};
```

# BomDiaImpl.java

```
package Exemplo1.Ola;

public class BomDiaImpl extends BomDiaPOA {

public
   java.lang.String SimplesOla(){
        java.lang.String s = “Bom DIA !!!!!";
        return s;
        };
}
```

# Server.java

```
package Exemplo1.Ola;
import java.io.*;
import org.omg.CORBA.*;
import org.omg.PortableServer.*;

public class Server
{
    public static void main(String[] args)
    {
        if( args.length != 1 )
     {
            System.out.println(
                "Uso correto: jaco Exemplo1.Ola.Server <ior_file>");
            System.exit( 1 );
        }
```

# Server.java

```
try{
        //inicializa o ORB
         ORB orb = ORB.init( args, null );

       //inicializa o POA
          POA poa = POAHelper.narrow( orb.resolve_initial_references( "RootPOA" ));
          poa.the_POAManager().activate();

            // Cria um Objeto BomDia
            BomDiaImpl bomDiaImpl = new BomDiaImpl();

            // Cria a referëncia de objeto
                org.omg.CORBA.Object obj = poa.servant_to_reference( bomDiaImpl );

            // Imprime a referëncia textualizada em arquivo
            PrintWriter pw = new PrintWriter( new FileWriter( args[ 0 ] ));
            pw.println( orb.object_to_string( obj ));
```

# Server.java

```
 pw.flush();
 pw.close();
// Aguarda requisicoes
            orb.run();
        }
        catch( Exception e )
        {
            System.out.println( e );
        }
    }
}
```

# Cliente.java

```
package Exemplo1.Ola;
import java.io.*;
import org.omg.CORBA.*;

public class Client
{
   public static void main( String args[] )
   {
      if( args.length != 1 )
     {
         System.out.println( "Uso correto: jaco Exemplo1.Ola.Client <ior_file>" );
         System.exit( 1 );
       }
       try
     {
         File f = new File( args[ 0 ] );
         //verifica se o arquivo existe
         if( ! f.exists() )
         {
            System.out.println("Arquivo " + args[0] +  " nao existe.");

            System.exit( -1 );
         }
```

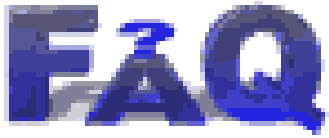
# Cliente.java

```
//verifica se args[0] aponta para um diretorio
        if( f.isDirectory() )
        {
           System.out.println("Arquivo " + args[0] + " eh um diretorio.");
           System.exit( -1 );
        }

        // inicialia o ORB.
        ORB orb = ORB.init( args, null );

        BufferedReader br =  new BufferedReader( new FileReader( f ));

        // recupera a referencia de um arquivo argumento de linha de comando
        org.omg.CORBA.Object obj =  orb.string_to_object( br.readLine() );

        br.close();
```

# Cliente.java

```
// converte  para BomDia
// se houver falha, e disparada a excecao BAD_PARAM

        BomDia bomDia = BomDiaHelper.narrow( obj );

        // Invoca a operacao e imprime o resultado
        System.out.println( bomDia.SimplesOla() );

    }

    catch( Exception ex )
        {
        ex.printStackTrace();
    }
  }
}
```

# Perguntas Sugestões Etc...




LaSDPC