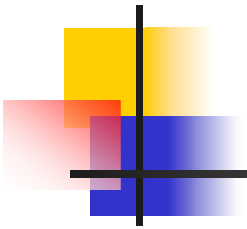
Aplicações e Arquitetura CORBA

Edvar Oliveira

edvar@ufpa.br

setembro/ 2004

Tópicos

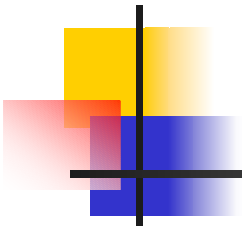


- Modelo Cliente/Servidor;
  - Comunicação no modelo cliente/servidor;
- A arquitetura CORBA;
  - Componentes da arquitetura CORBA;
- Etapas para o desenvolvimento de aplicações CORBA;
  - A IDL;
  - Implementações CORBA;
  - Exemplo.
- Perspectivas e tendências...

Conceitos: Modelo C/S

- Cliente
  - Necessita do acesso aos recursos administrados pelos servidores para realizar suas tarefas. É dele que partem as solicitações de serviços;
  - É um elemento *pró-ativo*, consumidor de serviços.
- Servidor
  - É responsável por gerenciar um determinado tipo de recurso do sistema, administrando o acesso concorrente dos clientes ao mesmo;
  - É um elemento *reativo*, fornecedor de serviços.

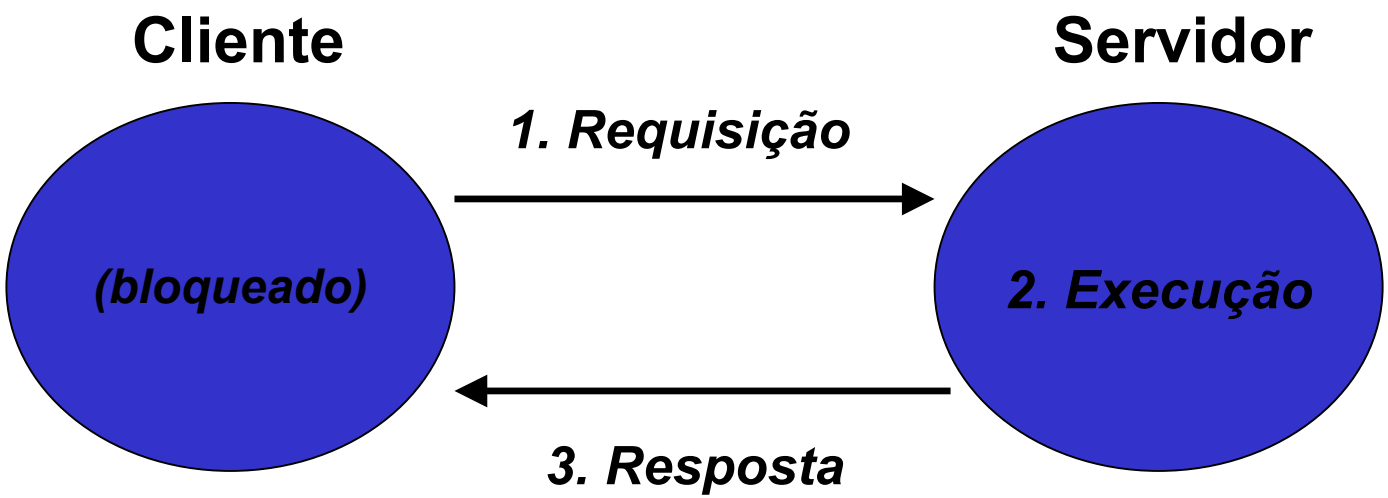
Vantagens e Desvantagens



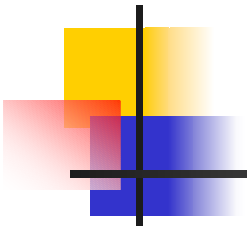
- Vantagens
  - Compartilhamento de recursos;
  - Balanceamento de carga;
  - Tolerância a falhas.

- Desvantagens:
  - Administração do sistema mais complexa;
  - Variados pontos de falha no sistema;
  - Dificuldades na interação entre componentes de fornecedores diferentes.

Comunicação no Modelo C/S

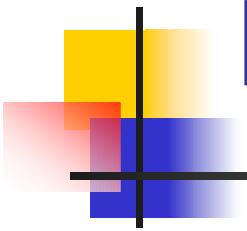


Comunicação no Modelo C/S (cont.)



- No Modelo Cliente-Servidor, a comunicação objetiva principalmente a realização de serviços;
- Passos:
  - Cliente envia requisição para o servidor;
  - Servidor recebe a mensagem e processa a solicitação;
  - Servidor envia os resultados ao cliente;
- A comunicação entre as partes é implementada usando-se *Passagem de Mensagens;*
- Em nível de programação, normalmente trabalha-se com abstrações como *RPCs* e Objetos Distribuídos *(RMI)*.

Passagem de Mensagens



- Os processos se comunicam através do envio de mensagens, utilizando primitivas do tipo *send/receive*;
- Solução de "baixo nível";
- Porém, oferece melhor desempenho;
- Algumas interfaces de passagem de mensagens:
  - Sockets (TCP/IP), IPX/SPX (Netware), NetBIOS (IBM e Microsoft).

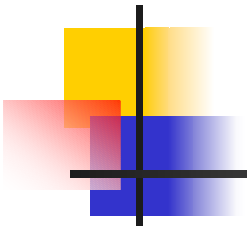
Passagem de Mensagens (cont.)

- Primitivas de Comunicação:
  - *Send*(msg, dest)
  - *Receive*(msg, orig)
- De maneira geral, a comunicação pode ser:
  - **Bloqueante:**
    - No *Send*, o processo emissor fica bloqueado até que ocorra o *Receive*;
    - No *Receive,* o receptor fica bloqueado até a chegada de uma mensagem.

Passagem de Mensagens (cont.)

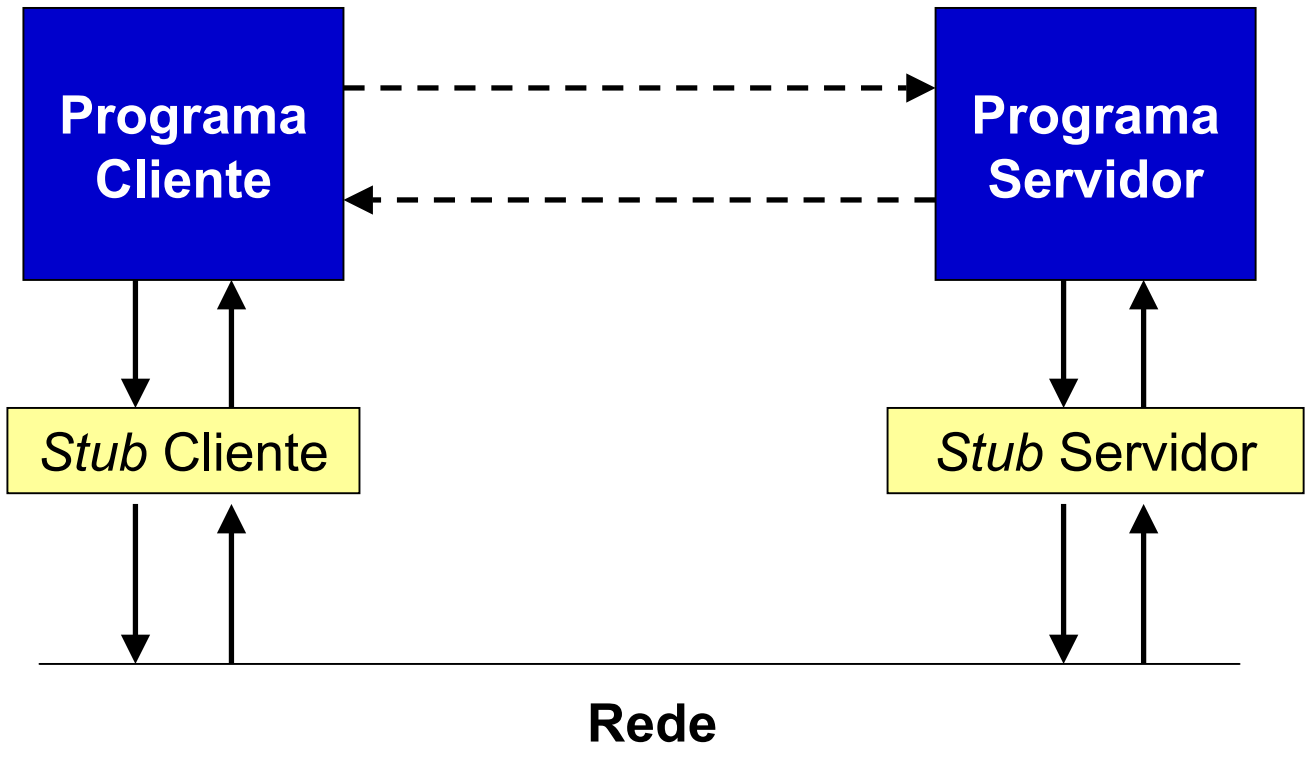
- A comunicação pode ser:
  - **Não-Bloqueante**
    - Após o *Send*, o processo emissor está liberado tão logo a mensagem tenha sido copiada para um *buffer* local;
    - O programa receptor apenas informa sua intenção de receber uma mensagem, alocando um *buffer* para recepção;
    - Esse tipo de comunicação pode oferecer melhor desempenho, mas sua programação pode ser mais complexa.

RPC *(Remote Procedure Call)*

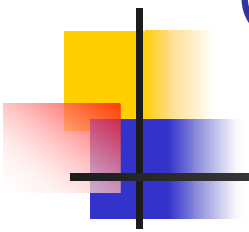


- Com as RPCs, um serviço apresenta-se a seus clientes como um módulo que possui uma interface bem definida de acesso a suas operações;
- Os parâmetros são enviados ao servidor em uma mensagem, assim como são retornados os resultados ao cliente;
- *Interface Definition Language* (IDL):
  - Descreve a interface do servidor;
  - Especifica os nomes dos procedimentos de acesso aos serviços, sua lista de parâmetros, tipos e se são de entrada ou saída.

Funcionamento de uma RPC



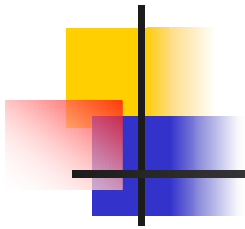
Funcionamento de uma RPC (cont.)



- Os *stubs* escondem do programador os detalhes de comunicação através da rede;
- Durante todo o processo, cliente e servidor "acreditam" que tudo esteja acontecendo localmente;
- Os *stubs* são gerados automaticamente a partir da descrição da interface do servidor na IDL do sistema (compilador de *interface*);
- Mecanismos como interfaces de passagem de mensagens, RPCs e RMIs são genericamente conhecidos como *middleware.*

A Arquitetura CORBA

Introdução

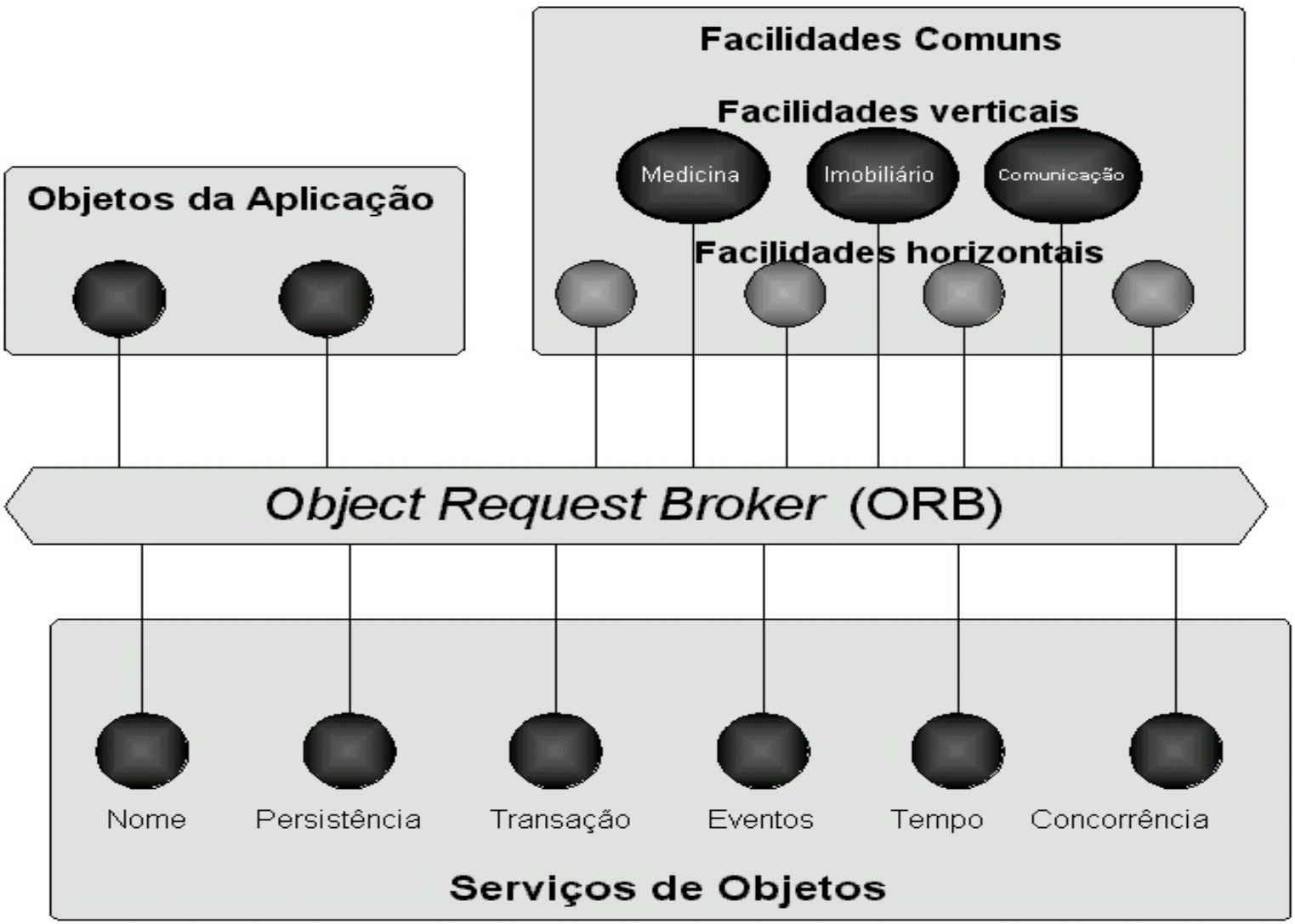


- CORBA - 1991
  - *Common Object Request Broker Architecture;*
  - Solução aberta para o desenvolvimento de aplicações distribuídas em ambientes heterogêneos.
- OMG: *Object Management Group* - 1989
  - Tem o objetivo de desenvolver, adotar e promover padrões para o desenvolvimento de aplicações em ambientes heterogêneos distribuídos;
  - Contribuições dos seus membros através de RFPs (*Requests for Proposals*).

OMA - *Object Management Architecture*

- OMA:
  - um Modelo de Objetos;
- Na visão da OMA, um objeto é:
  - Uma entidade encapsulada, com uma identidade única e que oferece serviços aos seus clientes através de uma interface bem definida;
  - Para o cliente, é transparente a localização do objeto, a linguagem em que ele foi escrito, sua forma de implementação e em que tipo de plataforma ele executa.

OMA - *Object Management Architecture*



Componentes da OMA

- ORB (*Object Request Broker*)
  - Funciona como o "elo" para comunicação entre clientes e servidores (objetos);
  - É o barramento de objetos.

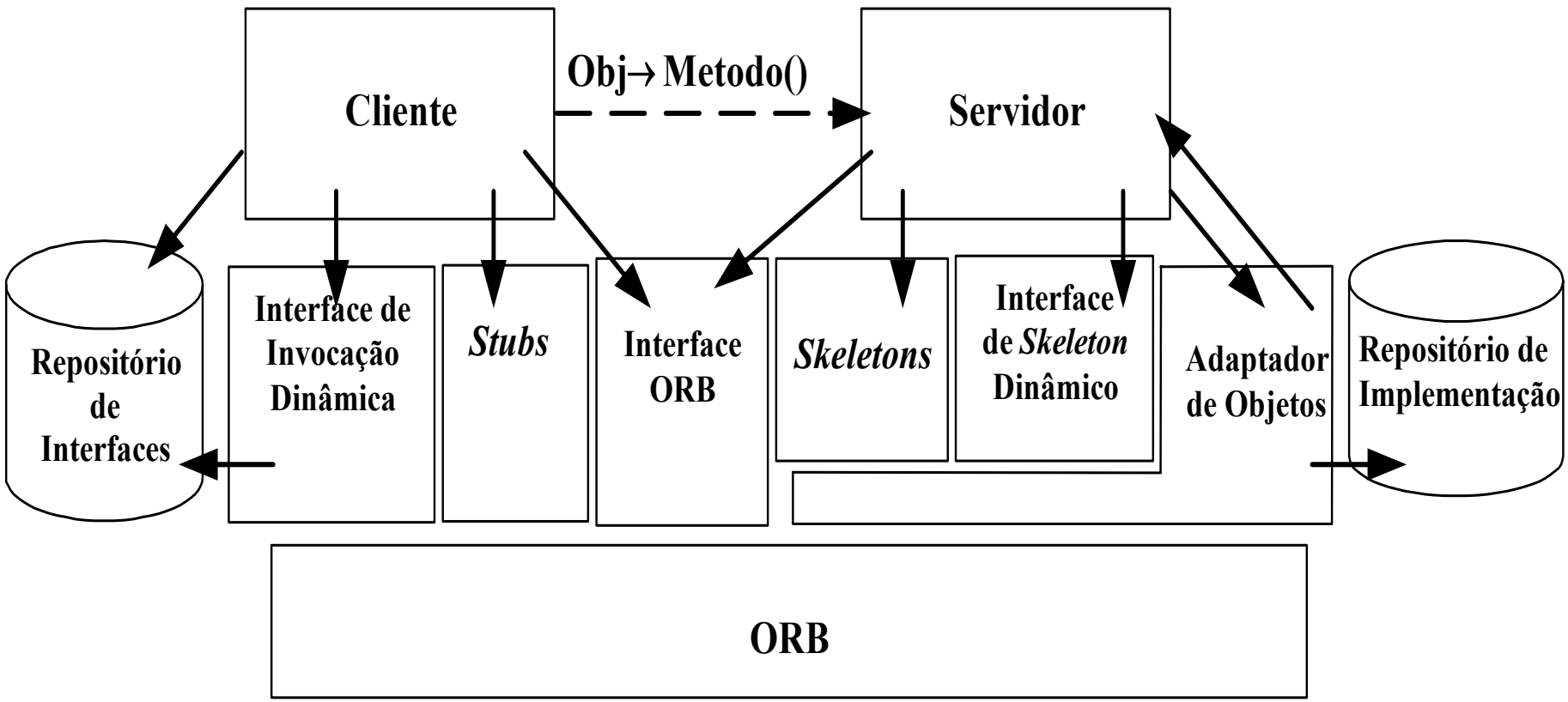
- Serviços de Objetos
  - Serviços de sistema que complementam a funcionalidade do ORB;
  - Ex: serviço de nomes, eventos, ciclo de vida, tempo, persistência etc.

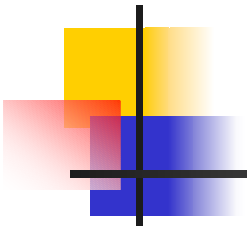
- Facilidades Comuns
  - Similares ao anterior, só que orientadas às aplicações. Podem ser de dois tipos: Horizontais e Verticais.

A arquitetura CORBA

- Em linhas gerais, a arquitetura CORBA é composta pelos seguintes componentes:
  - *Stubs* clientes;
  - *Skeletons;*
  - Repositório de Interfaces;
  - Interface de Invocação Dinâmica (DII);
  - Interface de *Skeleton* Dinâmica (DSI);
  - Repositório de Implementação;
  - Adaptador de Objetos (OA);
  - Interface do ORB.

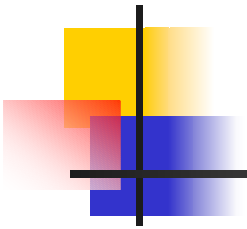
Arquitetura CORBA





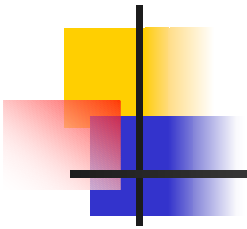
Stubs

- Os *stubs* clientes são as interfaces estáticas (geradas a partir do compilador IDL) para os serviços (objetos);
- O *stub* cliente compõe uma mensagem com a identificação do método invocado e seus parâmetros e a envia ao servidor;
- Em seguida, bloqueia-se para aguardar a resposta à solicitação feita.



Skeletons (ou *stubs* do servidor)

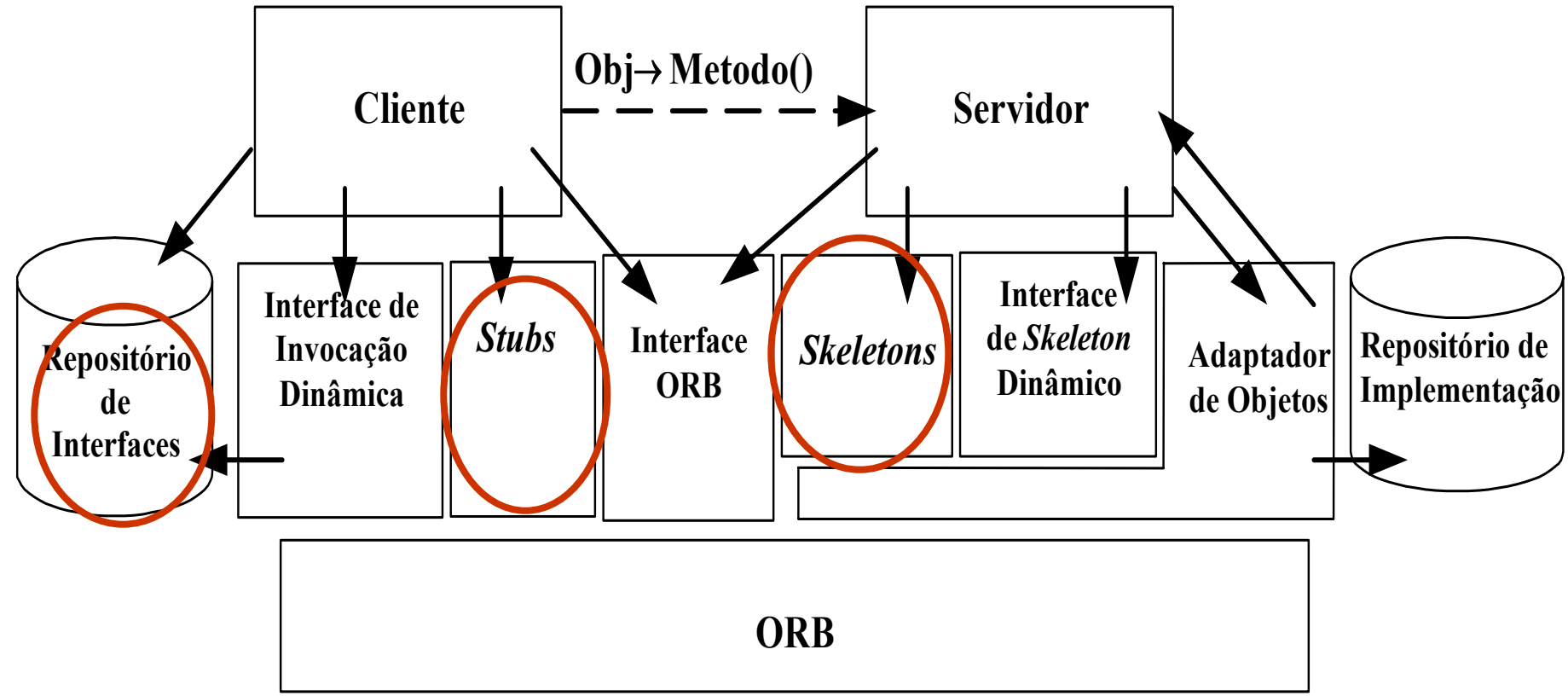
- São a parte correspondente aos *stubs* clientes, no ambiente servidor;
- Fornecem uma interface estática para os serviços definidos previamente na IDL;
- O *skeleton* obtém a identificação do método e os seus parâmetros da mensagem recebida e faz uma chamada local ao servidor;
- Ao ser completada a solicitação, envia uma mensagem com os resultados ao cliente.



Repositório de Interfaces

- O Repositório de Interfaces contém uma base de dados com a definição das interfaces de todos os serviços conhecidos pelo ORB;
- É um repositório de metadados dinâmico para o ORB.

Arquitetura CORBA



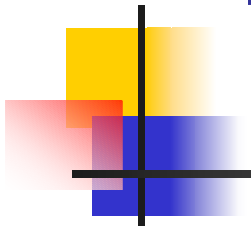
Interface de Invocação Dinâmica - (DII)

- Permite que o cliente invoque um método no servidor sem que tenha conhecimento, em tempo de compilação, de sua interface;
- Neste tipo de interação, portanto, não se usam *stubs IDL*;
- Os dados sobre o objeto remoto são obtidos a partir da base de dados do Repositório de Interfaces.

Interface de Skeleton Dinâmica - (DSI)

- É o correspondente à DII, no lado servidor;
- Permite que os servidores sejam escritos sem que se tenha *skeletons IDL* previamente compilados no código do programa;
- É útil na implementação de pontes entre "ferramentas CORBA" (ORBs) e também para fazer a comunicação entre CORBA e outras plataformas de computação distribuída.

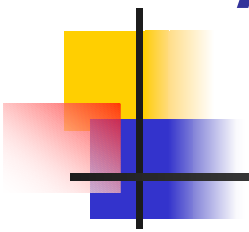
Repositório de Implementação



- O Repositório de Implementação é um repositório, em tempo de execução, para as classes que um servidor suporta, os objetos instanciados e suas identificações;

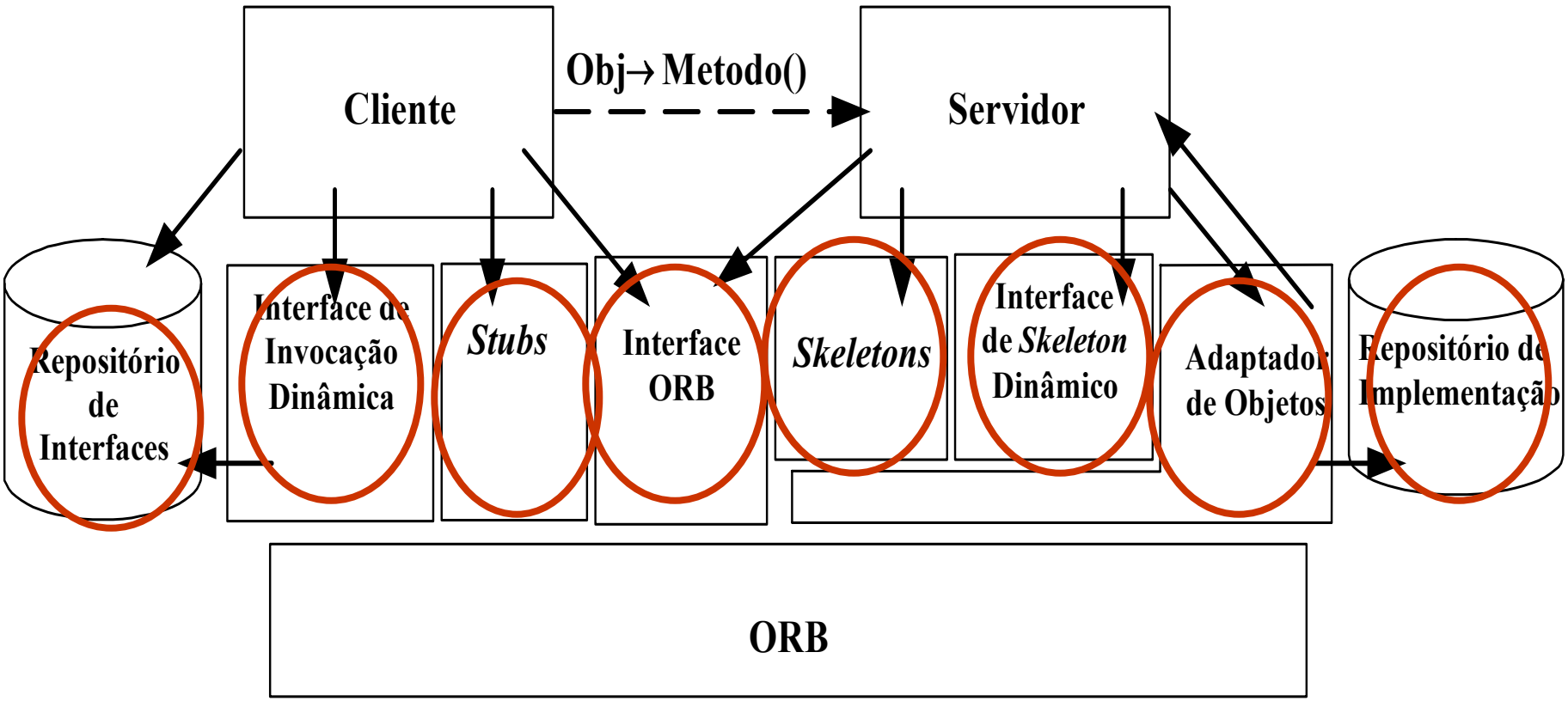
- Permite armazenar informações como o "nome do programa servidor" que deve ser inicializado.

Adaptador de Objetos

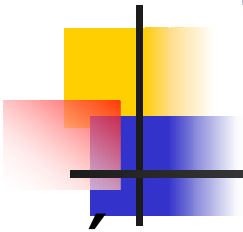


- O Adaptador de Objetos é a "cola" entre o ORB e as implementações dos objetos no lado servidor;
- É responsável por:
  - Registrar as classes servidoras no Repositório de Implementação;
  - Ativar (instanciar) os objetos chamados, em tempo de execução, segundo a demanda dos clientes;
  - Receber as chamadas para os objetos e repassá-las aos mesmos (invocação de métodos);
  - Gerar e gerenciar as *referências de objetos* (ORs).

Arquitetura CORBA



Invocação Estática

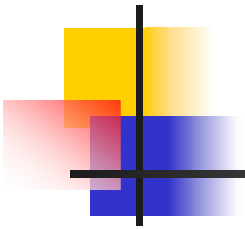


- É a forma mais usual de chamada de métodos em CORBA;
- É chamada de estática porque utiliza *stubs* e *skeletons* estáticos (gerados a partir de um compilador IDL);
- É útil quando se conhece, em tempo de compilação, as particularidades das operações que serão chamadas;
- Pode ser dividida em: síncronas (bloqueantes), assíncronas e *oneway;*

Invocação Estática (cont.)

- Vantagens:
  - Mais fácil de programar (similar a uma RPC);
  - Verificação de tipos mais robusta (em tempo de compilação);
  - Auto-documentação;
  - Melhor desempenho que as outras formas de interação;
- Porém. . .
  - É menos flexível.

Invocação Dinâmica



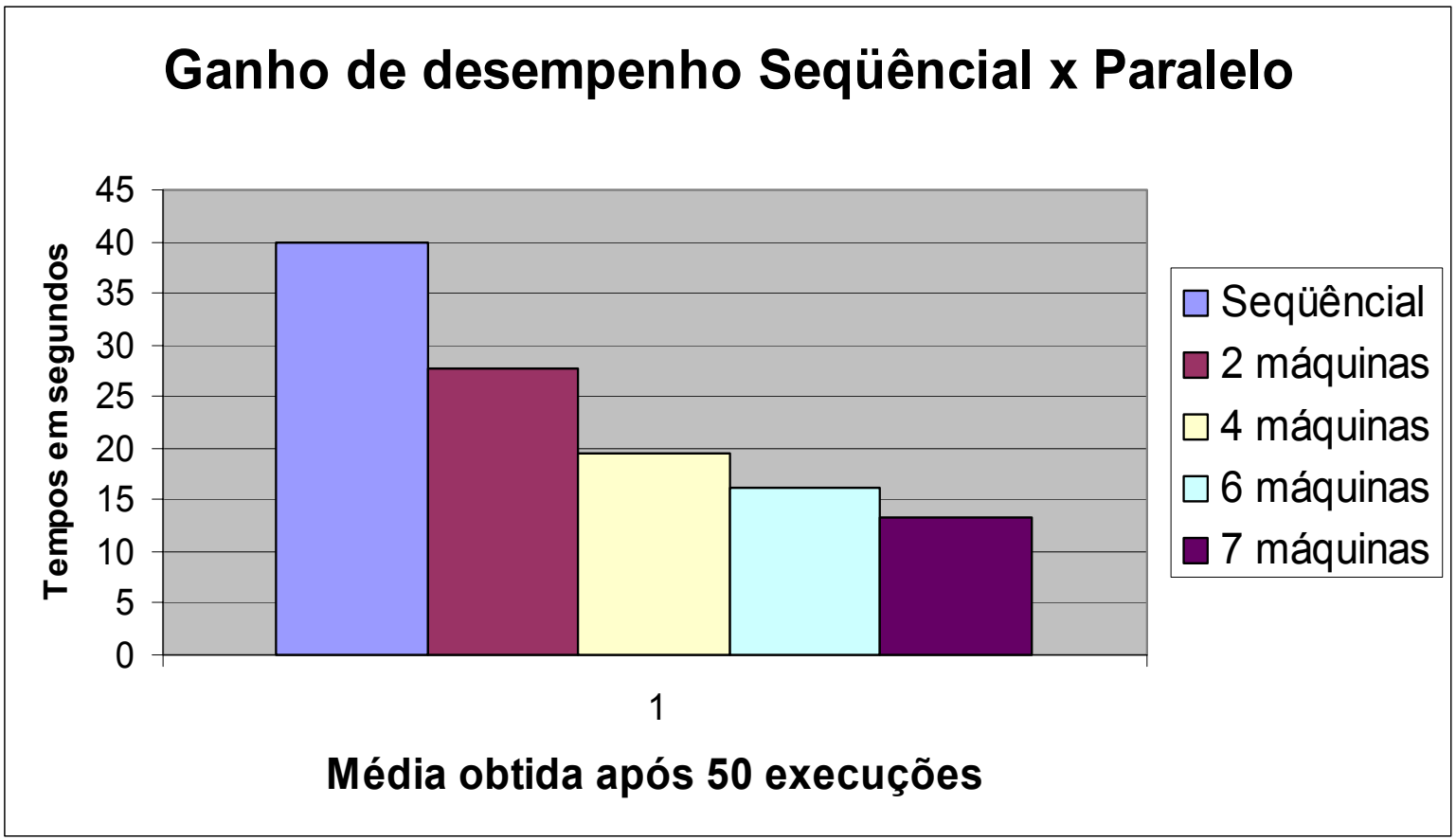
- Permite que o cliente especifique, em tempo de execução, o objeto e o método que deseja invocar;
- O cliente, então, "constrói" a sua chamada através de uma seqüência de troca de mensagens com o Repositório de Interfaces;
- É mais flexível que o método estático, porém é mais difícil de programar e apresenta desempenho inferior;

Invocações Assíncronas

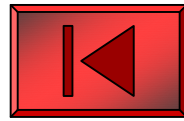
- Vantagens:
  - Maior flexibilidade para o programa cliente;
  - Menor complexidade em relação às chamadas DII;
  - Possibilidade de ganhos de desempenho.
- Desvantagens:
  - Maior complexidade de implementação para o programa cliente;
  - Necessidade de estabelecer critérios **bem definidos** para localizar os programas servidores.

Exemplo de Aplicação

Caixeiro Viajante: Ganho de Desempenho



Referências de Objetos



- As referências de objetos identificam, de forma única, um objeto dentro de um dado sistema CORBA;
- Uma referência de objeto pode ser um nome ou identificador, a sua representação interna fica a cargo de cada ferramenta ou implementação CORBA;
- Os clientes podem obter essas referências a partir de arquivos, serviços de diretórios, do Repositório de Interfaces ou como resultado de invocação de métodos.

Desenvolvimento de Aplicações CORBA

IDL
Implementações CORBA
Exemplo

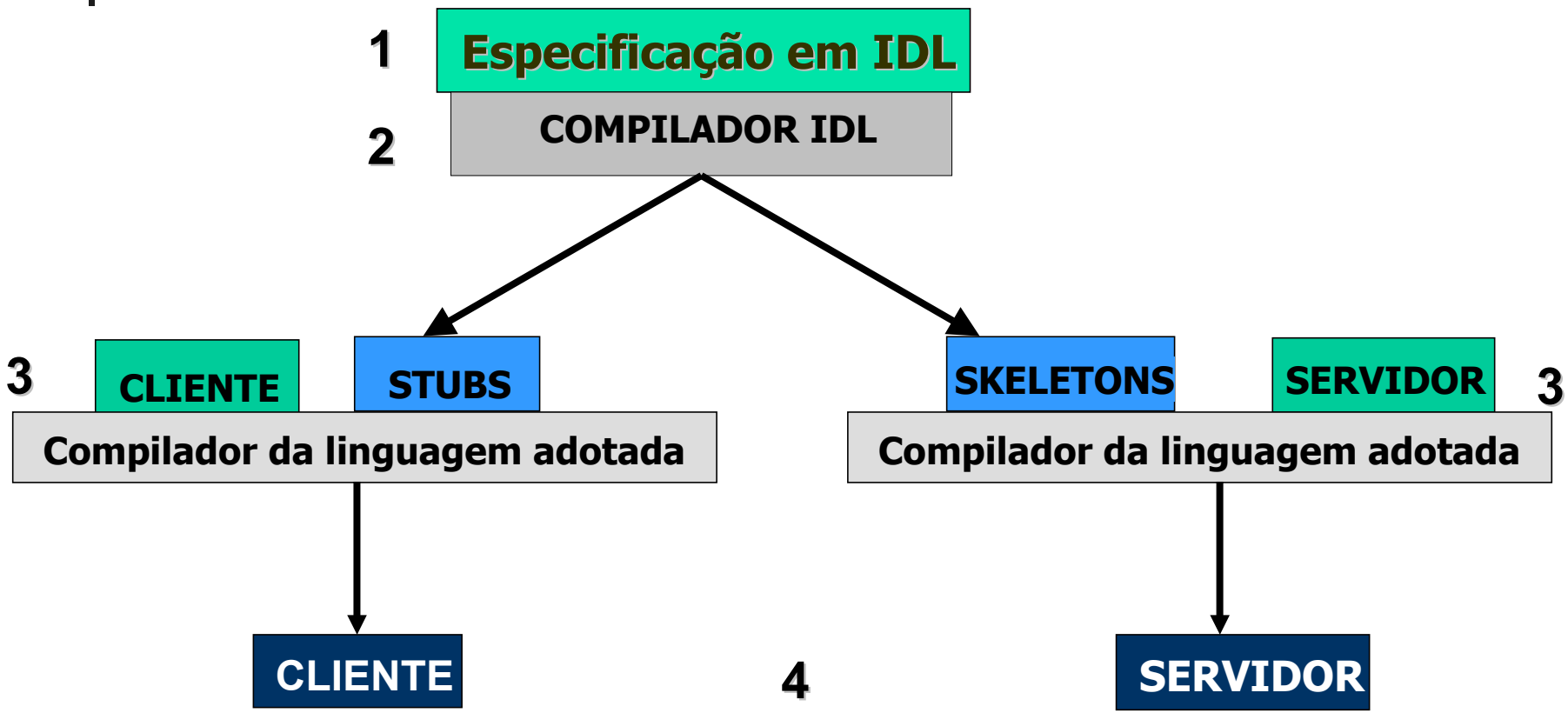
Desenvolvimento de aplicações CORBA

- Escolher uma ferramenta/implementação CORBA;
- Nessa escolha, algumas características devem ser analisadas:
  - Conformidade com a especificação;
  - Documentação disponível;
  - Mapeamento para linguagens de programação;
- Pode ser necessário verificar a viabilidade da ferramenta para o problema em questão;
- Conhecer os recursos fornecidos pela IDL da especificação CORBA.

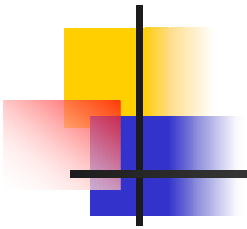
Desenvolvimento de aplicações CORBA (cont.)

- O Processo de desenvolvimento é geralmente realizado em etapas:
  1 - Definir as interfaces (utilizando a IDL), métodos e parâmetros, que poderão ser invocados por clientes;
  2 - Utilizar um compilador IDL (fornecido pela implementação CORBA) para gerar os *stubs* e *skeletons*;
  3 - Implementar as aplicações cliente e servidor;
  4 - Compilar (utilizando o compilador da linguagem adotada) essas aplicações com os *stubs* e *skeletons* para gerar os arquivos executáveis.

Etapas de desenvolvimento para uma aplicação CORBA



A IDL



- A IDL é utilizada para descrever as interfaces dos objetos servidores, definindo suas funcionalidades;
- A OMG CORBA IDL é um padrão ISO e ANSI (X3H7);
- A IDL não é uma linguagem de programação. Ao contrário, ela permite que as interfaces sejam definidas independentemente das linguagens utilizadas para implementá-las;

A IDL (cont.)

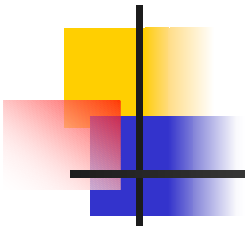
- Com o uso da IDL, é possível especificar:
  - Os
  - Se pos e v
  - As
  - Ex
- A gr linguagem C++ com mais algumas palavras-chave para suportar conceitos de sistemas distribuídos.

Deve haver um mapeamento de IDL para uma linguagem de programação, a fim de que *stubs, skeletons* e servidor sejam implementados nessa linguagem

Exemplo de IDL

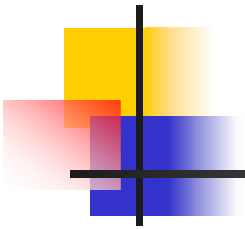
```
module Banco
{
    interface Conta{
        void deposito(in long conta, in long valor);
        long saldo(in string nome, in long conta);
    };
    interface Cliente{
        boolean cadastro(in string nome, in string endereco);
        boolean pesquisa(inout string nome, inout string endereco);
    };

};
```

A IDL - Tipos de Dados



- Tipos de dados
  - Básicos: *short, long* (32 bits)*, long long, unsigned long, float, double, long double, fixed<x, y>, char, wchar, boolean, octet, any*;
  - Construídos: *string, wstring, array, sequence, struct, union, enum;*

IDL - Operações *oneway*



- Uma operação pode ser definida na IDL como *oneway*, a fim de não provocar o bloqueio do cliente durante a chamada;
  - Envio pelo "melhor esforço" (*best effort*);
- A especificação CORBA determina que qualquer requisição a um objeto deve ser executada no máximo uma vez (semântica *at-most-once*).

```
interface Bilheteria {
    oneway void fazComentario (in string coment);
};
```

Algumas implementações CORBA

- ORBit:http://orbit-resource.sourceforge.net/
- TAO:http://www.cs.wustl.edu/~schmidt/TAO
- OmniORB:
  http://www.uk.research.att.com/omniORB/
- ORBacus: http://www.ooc.com
- MiCO: http://www.mico.org/
- Visibroker: http://www.inprise.com/visibroker
- Orbix: http://www.iona.com

OK!
Sei tudo de IDL.

O que devo escrever nos programas cliente e servidor?

Programa Cliente

- Em linhas gerais, o cliente segue o seguinte roteiro:
  - Inicializar ou "contactar" o ORB;
  - Declara um objeto da classe implementada no servidor e definida como interface no arquivo IDL;
  - Obtém a Referência do Objeto Servidor;
  - A partir da Referência obtida, inicializa o objeto de acordo com a classe implementada no servidor;
  - Realiza requisições aos servidores.

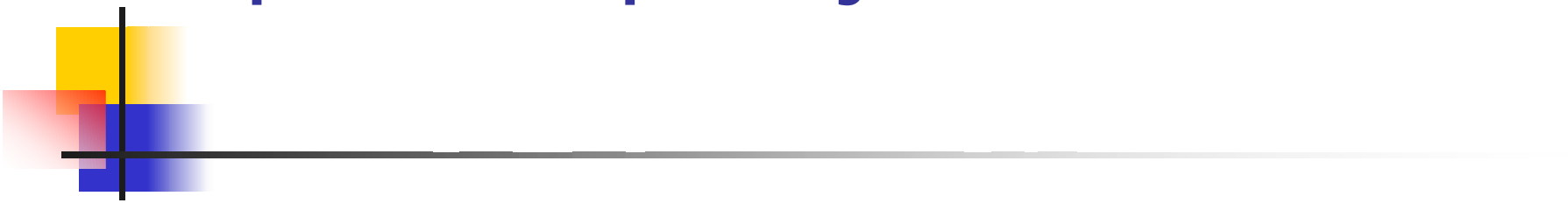
Programa Servidor

- **Além de implementar a classe e os métodos definidos em IDL, o servidor deve:**
  - Inicializar o ORB e o Adaptador de Objetos;
  - Declarar um objeto da classe implementada no servidor e definida no arquivo IDL;
  - Criar uma referência do objeto e registrar essa referência em algum tipo de Serviço de Nomes;
  - Aguardar por requisições de clientes.

Exemplo Hello World!

☺

Exemplo de Aplicações



- **Exemplo Hello World - Definição em IDL**

```
interface Hello
{
   void say_hello();
};
```

Exemplos de Aplicações

Exemplo Hello World - Aplicação Servidor

```
public class Server
{
   public static void main(String args[])
   {
      orb = org.omg.CORBA.ORB.init(args, props);

      // Resolve Root POA
      org.omg.PortableServer.POA rootPOA = org.omg.PortableServer.POAHelper.narrow(
                                          orb.resolve_initial_references("RootPOA"));


      org.omg.PortableServer.POAManager manager = rootPOA.the_POAManager();
      manager.activate();

       // Cria o objeto
       Hello_impl helloImpl = new Hello_impl();
       Hello hello = helloImpl._this(orb);
```

Inicializa o ORB

Inicializa o Adaptador de Objetos

Exemplos de Aplicações

Exemplo Hello World - Aplicação Servidor

```
    //Grava a referência do objeto em um arquivo
      String ref = orb.object_to_string(hello);
      out.println(ref);
      out.close();
      orb.run();
  }
}

public class Hello_impl extends HelloPOA
{
  public void say_hello()
  {
    System.out.println("Hello world!");
  }
}
```

Converte o objeto hello para uma string

Servidor fica aguardando por requisições de clientes

Implementação do método say_hello()

Exemplos de Aplicações

Exemplo Hello World - Aplicação Cliente

```
package hello;

public class Client
{
   public static void main(String args[])
   {
      org.omg.CORBA.Object obj = null;
      int status = 0;
      org.omg.CORBA.ORB orb = null;

      orb = org.omg.CORBA.ORB.init(args, props);


     try{
          String refFile = "Hello.ref";
          BufferedReader in = new BufferedReader(new FileReader(refFile));
          String ref = in.readLine();
          obj = orb.string_to_object(ref);
      }
```

Inicializa o ORB

Obtém a Ref. do Objeto remoto através de uma string armazenada em arquivo

Exemplos de Aplicações

Exemplo Hello World - Aplicação Cliente

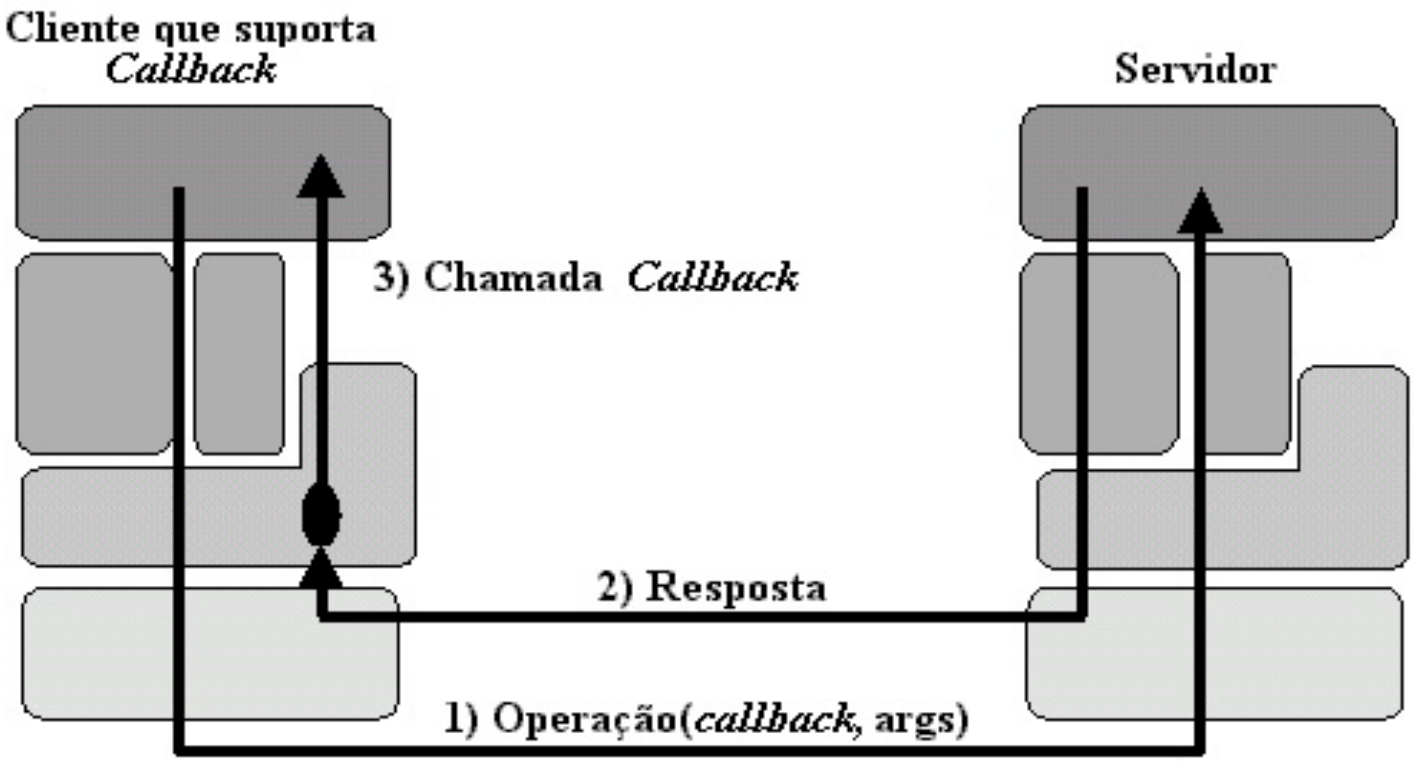
```
// Converte para uma referencia ao objeto Hello
        Hello hello = HelloHelper.narrow(obj);

// Chama o metodo remoto
        hello.say_hello();

    }
}
```

Converte a Referência para um objeto do tipo Hello

Realiza a chamada ao método say_hello no objeto remoto

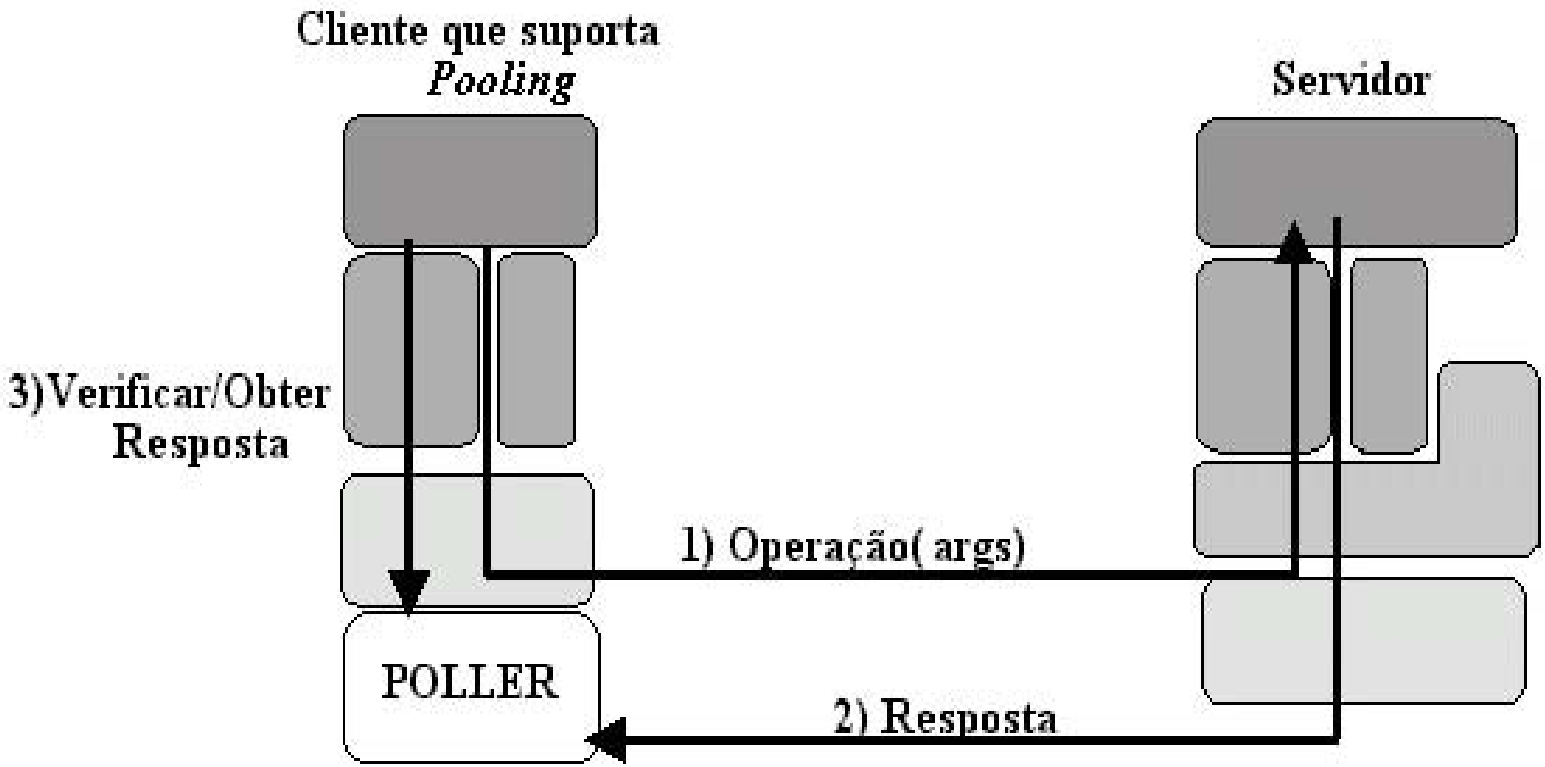
Chamadas Assíncronas

- **Modelo *Callback***



Chamadas Assíncronas

Modelo *Polling*



Perspectivas, Tendências e outras coisas que vêm por aí...

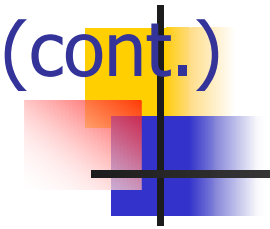
Evolução das especificações CORBA

- **CORBA 1.1** (1991)
  - Especificava a IDL, a API do ORB e *bindings* com algumas linguagens.

- **CORBA 2.0** (1994)
  - Superconjunto do CORBA 1.1;
  - Permite a interoperabilidade entre ORBs de diferentes fornecedores (introdução do IIOP).

- **CORBA 2.4** (Outubro/2000)
  - Especificação CORBA *Messaging* (chamadas assíncronas), Novos tipos de dados para a IDL.

Evolução das especificações CORBA (cont.)



- **Atualmente: CORBA 2.5 (Setembro/2001)**

- **Futuro: CORBA 3.0** (início 1998)
  - Messaging, CORBA Beans, CORBA/DCOM, Agentes Móveis, Workflow, Telecom, E-commerce, ...

Utilização da arquitetura CORBA em empresas

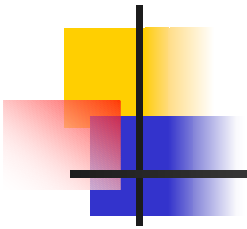
- Netscape/AOL
  - Coloca um ORB Java Visibroker em cada browser;
  - Usa CORBA para a comunicação no *Netscape Enterprise Server;*
- Oracle
  - Adotou CORBA/Java como a plataforma para sua NCA (*Network Computing Architecture*);
  - Oracle 8i : toda a comunicação via ORB (Visibroker);
  - O *engine* de banco de dados está sendo subdividido em componentes usando CORBA;
  - Oracle Application Server 4.0;

Utilização da arquitetura CORBA em empresas

- Sun/JavaSoft
  - CORBA está sendo integrado ao núcleo de Java;
  - Sun adotou o Visibroker como seu ORB para o Solaris;
- IBM/Lotus
  - Está baseando sua plataforma de computação distribuída em CORBA;
- Outros
  - HP, Iona, Visigenic/Borland, Novell, GemStone, ODI, Versant, Sybase, Symantec, Expersoft.

Perspectivas e tendências...

- **Interoperabilidade:**
  - **Entre ORBs;**
  - **CORBA x COM;**
- **Invocação assíncrona:**
  - **Modelos *Callback* e *Polling*;**
- **Especificação *Data Parallel* CORBA;**
- **Novos serviços:**
  - **Monitoramento da aplicação;**
  - **Escalonamento de requisições.**



Perguntas?
Dúvidas?
Questões?

Referências Bibliográficas

- LO, Sai-Lai. RIDDOCH, David. GRISBY, Duncan. **The omniORB verison 3.0 User's Guide**. Cambridge: AT&T Laboratories, 2000.
- MAFFEIS, Silvano. LANDIS, Sean. **Building Reliable Distributed Systems with CORBA**. IN: Theory and Practice of Object Systems, New York: John Wiley & Sons, April, 1997.
- MAFFEIS, Silvano. RENESSE, Robert van. BIRMAN, Kenneth P. Horus: A Flexible Group Communications System. **Communications of the ACM.**. v. 39, n. 4, April, 1996.
- OBJECT ORIENTED CONCEPTS. **ORBacus for C++ and Java User's Manual**. http://www.ooc.com/ob/download4.html, 2000.
- ORFALI, Robert. **The Essential Distributed Objects Survival Guide**. New York: John Wiley & Sons, 1996.
- PUDER, Arno. RÖMER, Kay. PILHOFER, Frank. **MiCO: An Open Source CORBA 2.3 Implementation**. http://www.mico.org/doc-book.ps, 2000.
- SCHMIDT, Douglas C. VINOSKI, Steve. Object Interconnections: An Introduction to CORBA Messaging. **C++ Report Magazine**. Washington-DC, v. 15, November/December, 1998.
- SANTOS, Ricardo R., **Desenvolvimento de aplicações distribuídas na arquitetura CORBA,** setembro, 2001.