Introdução ao Middleware de Objetos Distribuídos: conceitos e RMI

Referências:

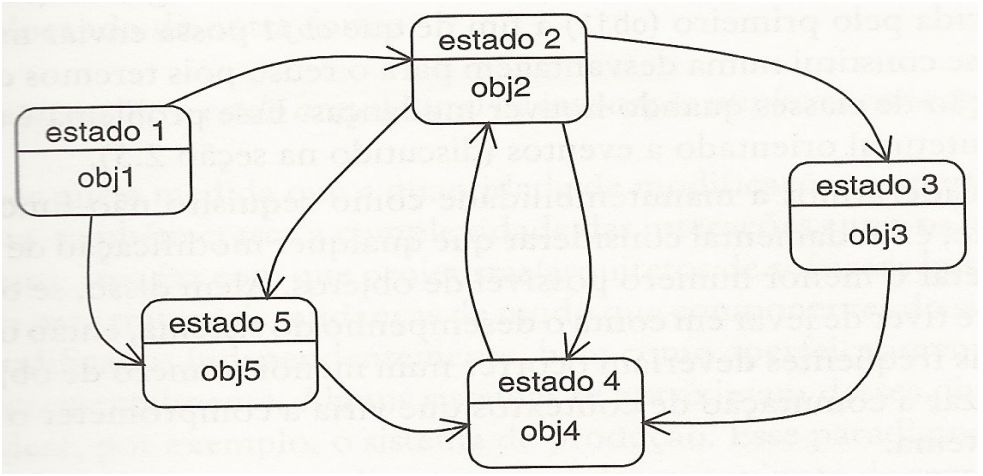
• Seção 10.3.4 - STEEN, V. M. TANENBAUM, A. S. , “Sistemas Distribuídos: princípios e paradigmas”, 2a. Edição, 2007.

• Seção 5.5 e Cap. 10 - Coulouris et al., “Sistema Distribuídos: conceitos e projeto”, 4a. Edição, 2007.

Middleware de Objetos Distribuídos: apresentação

"...capacidade de embutir software em componentes bem definidos e mais ou menos independentes..."

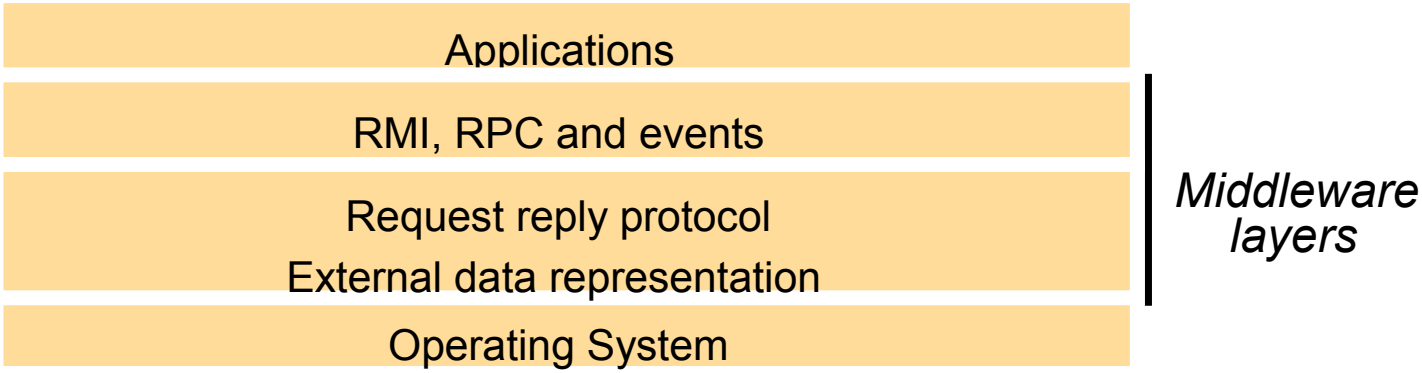
"Transparência na distribuição... tudo é tratado como objeto, e serviços e recursos são oferecidos a clientes na forma de objetos que eles possam invocar" (TANENBAUM & STEEN, 2008)



Middleware de Objetos Distribuídos: middleware

"Software que fornece um modelo de programação acima dos blocos de construção básicos de processos e de passagem de mensagens " (Coulouris *et al*., 2007).

- Usa protocolos baseados em mensagens entre processos para fornecer suas abstrações de mais alto nível
- Independência do protocolo de comunicação, sistemas operacionais e hardware.
- Algumas formas de middleware provêm independência da linguagem de programação



Middleware de Objetos Distribuídos: definições

De acordo com Tanenbaum (TANENBAUM & STEEN, 2008):

- **Estado**: refere-se à capacidade de armazenamento de dados provida por um objeto
- **Métodos**: operações executadas nos dados
- **Interface**: meio utilizado para disponibilizar os métodos
- Objetos podem implementar várias interfaces;
  - Uma definição de interface pode ser implementada por diferentes objetos

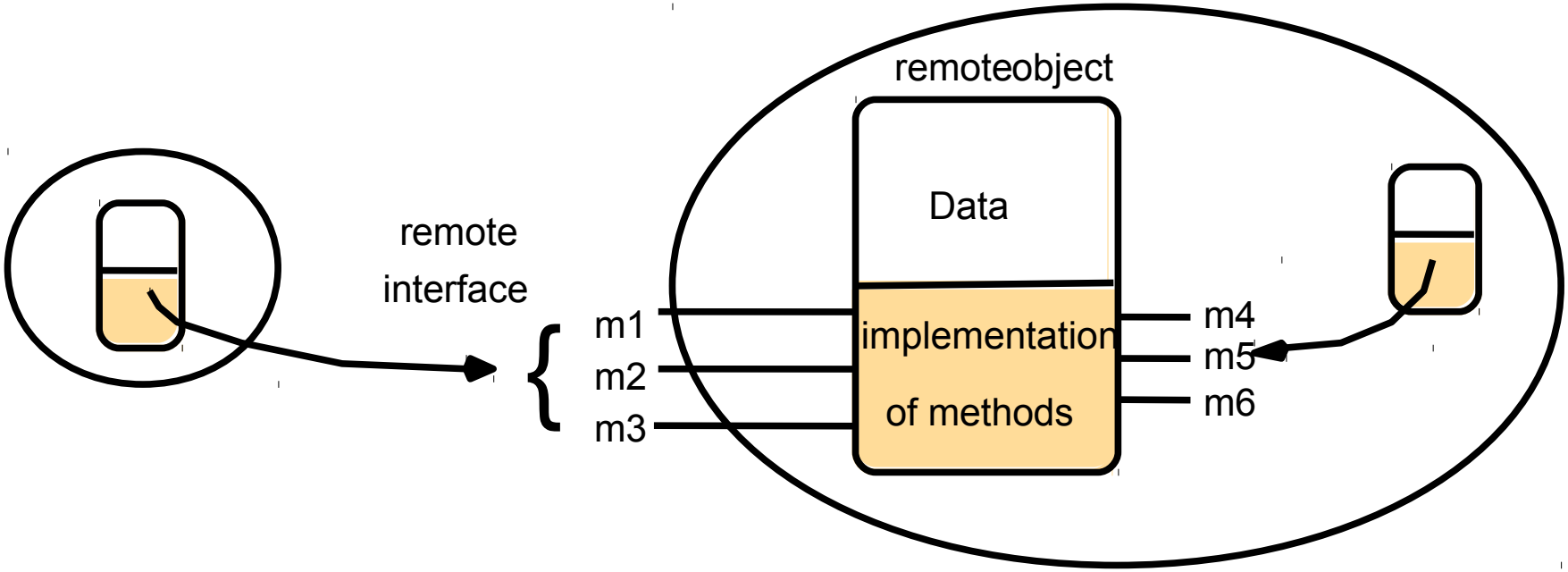
Middleware de Objetos Distribuídos: definições

- **Objeto Distribuído:** organização que possibilita a separação entre interface e objeto. Cada uma dessas entidades pode residir em um nó (máquina) diferente

- **Proxy ou Stub:** uma implementação da Interface do objeto carregada no espaço (máquina) do solicitante (cliente). É o responsável pela transformação das invocações a métodos em mensagens que posteriormente devem ser trocadas

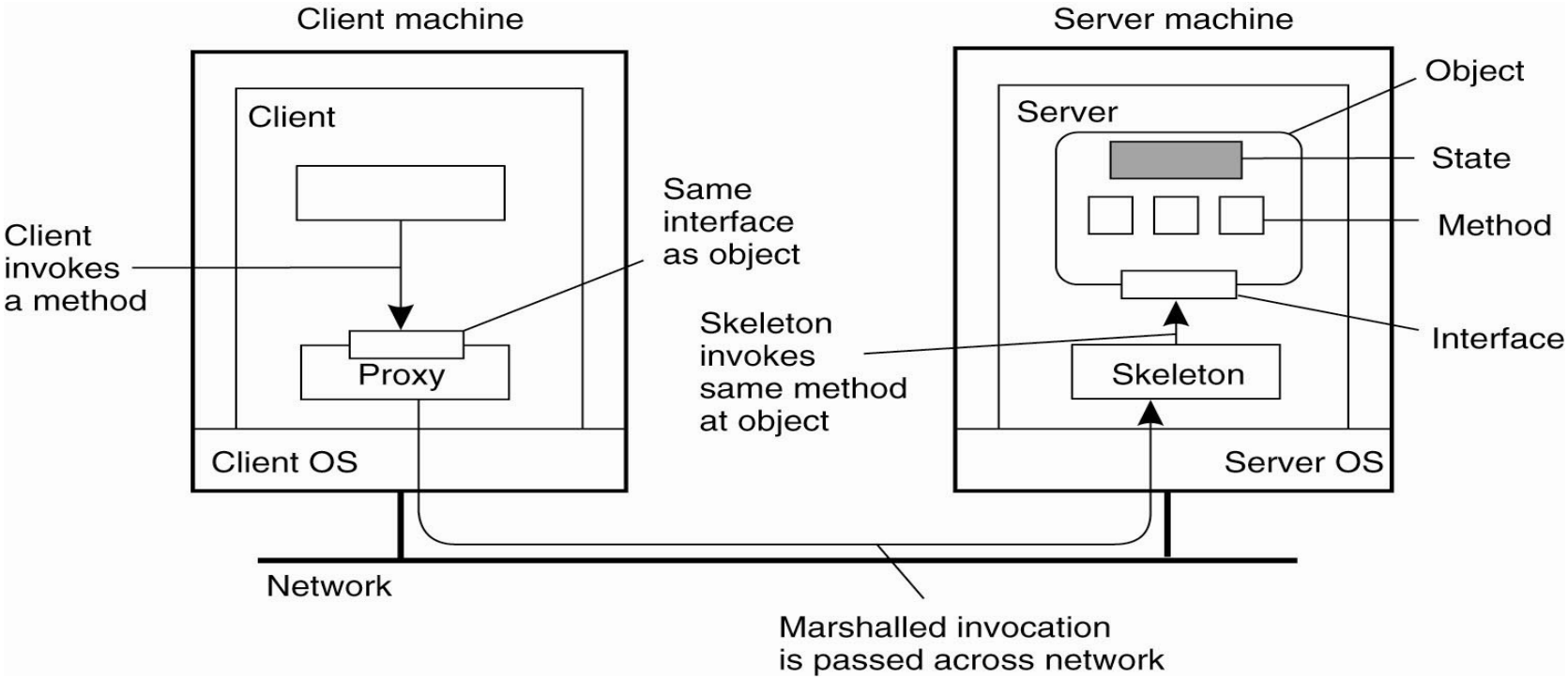
- **Esqueleto (ou *Skeleton*):** o apêndice do lado do servidor

Middleware de Objetos Distribuídos: definições



Um objeto com interfaces local e remota

Middleware de Objetos Distribuídos: definições



"O Estado de um objeto distribuído (ou objeto remoto) não é distribuído, ou seja, reside numa única máquina. Apenas as interfaces implementadas pelo objeto são disponibilizadas em outras máquinas, que também pode ficar oculta (transparente) para os clientes" (TANENBAUM & STEEN, 2008)

Middleware de Objetos Distribuídos: definições

Objetos de **Tempo de Compilação**:

- Abstração no âmbito da linguagem de programação. Por exemplo, Java, C++,...
  - Objeto definido como uma instância de uma classe
- Facilita a construção de SD
  - Após a compilação, geração das entidades que compõe o middleware, o programadores podem ficar completamente alheio à distribuição dos objetos.

Middleware de Objetos Distribuídos: definições

Objetos de **Tempo de Execução:**

- Possibilita a construção de uma aplicação com base em objetos escritos em várias linguagens
- Várias formas de implementar. Desde uma biblioteca de funções que atua sobre um arquivo até o uso de estruturas mais sofisticadas, por exemplo, Reflexão Computacional.
- **Adaptador de Objetos:** invólucro ao redor da implementação para criar a aparência de um objeto. Vincula dinamicamente à implementação

Middleware de Objetos Distribuídos: definições

Objetos Persistentes:

- Não dependem do servidor corrente
- Mantêm o estado corrente de um objeto

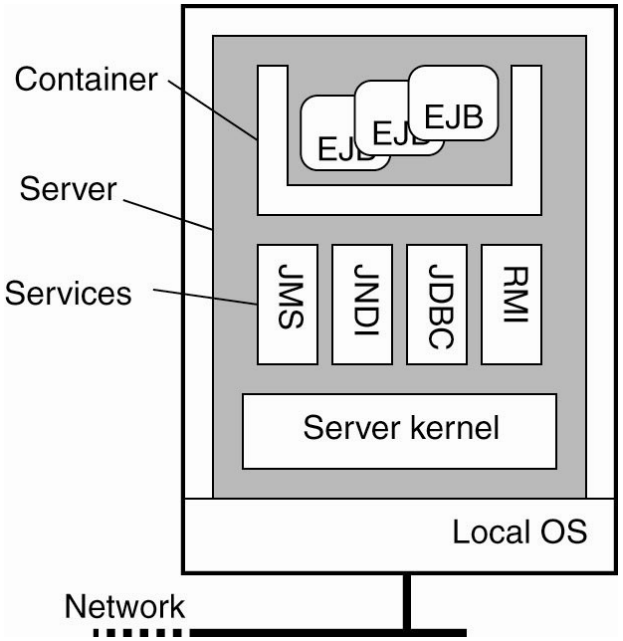
Objetos Transientes:

- Existe apenas enquanto existir o servidor que o está hospedando

Middleware de Objetos Distribuídos: exemplo1

Enterprise Java Beans (EJB)

- Beans de sessão sem estado
  - É invocado uma vez e descarta informações utilizadas na execução do serviço. Ex. uma consulta SQL
- Beans de sessão com estado
  - Mantém o estado relacionado ao cliente. Ex. um carrinho de compras no comércio eletrônico
- Beans de entidade
  - Objetos persistentes. Armazenam informações que poderão ser necessárias numa segunda sessão
- Beans acionados por mensagens
  - Tratadores de mensagens, sem estado.

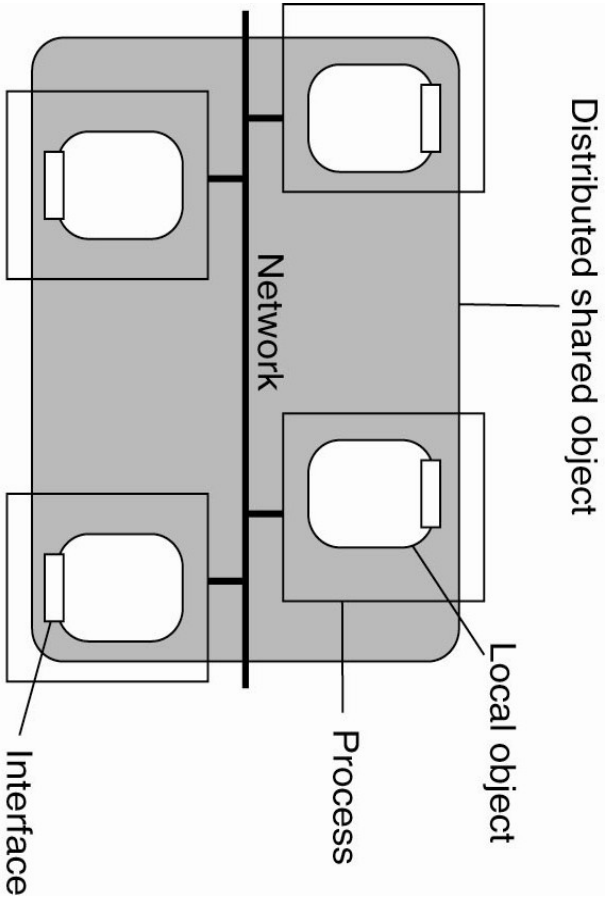


Middleware de Objetos Distribuídos: exemplo2

Globe

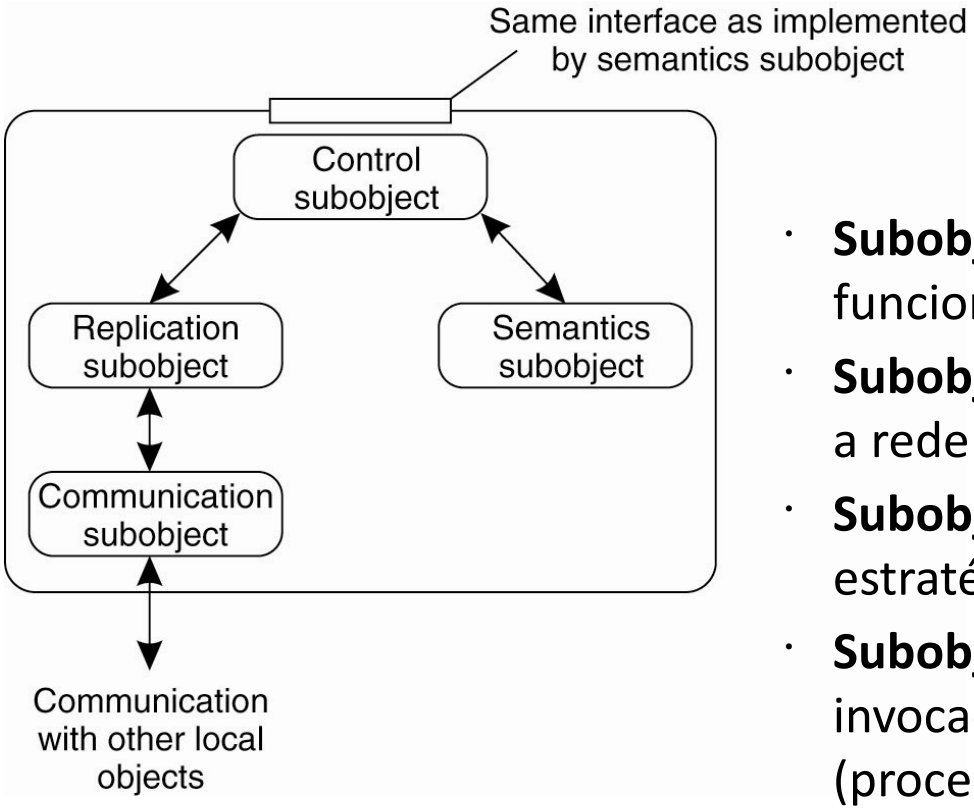
- Não utiliza o modelo de Objeto Remoto
- O estado de um objeto pode ser distribuído e replicado por vários processos.
  - **Objeto Local:** implementação específica das interfaces fornecidas pelo Objeto Compartilhado Distribuído
  - **Objeto Local Primitivo**: não contém nenhum outro objeto local
  - **Objeto Local Composto:** composto de vários objetos locais.

    Composição é usada para implementar objetos compartilhados distribuídos



Middleware de Objetos Distribuídos: exemplo2

Globe



- **Subobjeto de semântica:** implementa a funcionalidade oferecida
- **Subobjeto de comunicação:** interface com a rede subjacente
- **Subobjeto de replicação:** implementa a estratégia de distribuição do objeto
- **Subobjeto de controle:** cuida da invocações dos processos dos clientes (processos vinculados ao objeto compartilhado distribuído) e controla as interações entre o subobjeto de semântica e o subobjeto de controle

Middleware de Objetos Distribuídos: exemplo2

Globe: interface implementada pelo subobjeto de semântica

```
public interface GOSPersistentSemanticsSubobject
{
    public void setPerstID( long pid )
        throws GOSPersistentSubobjectException;

    public byte[] getIncoreState()
        throws GOSPersistentSubobjectException;

    public void setIncoreState( byte[] state, boolean recoverMode )
        throws GOSPersistentSubobjectException;

    public void prepareDestruction()
        throws GOSPersistentSubobjectException;
}
```

Figure 7: Interface to be implemented by a semantics subobject to become persistent.

http://www.cs.vu.nl/pub/papers/globe/IR-CS-003.03.pdf

Middleware de OD: processos servidores

"Hospedam serviços implementados pelos objetos"

Invocação de Objetos (ou Política de Ativação):

- Mesma política (tratamento igualitário) ou Políticas diferentes
  - Por exemplo, criar um objeto transiente na primeira requisição de invocação e destruí-lo tão logo não haja mais nenhum cliente vinculado a ele ou criar todos os objetos transientes no momento que o servidor é iniciado.

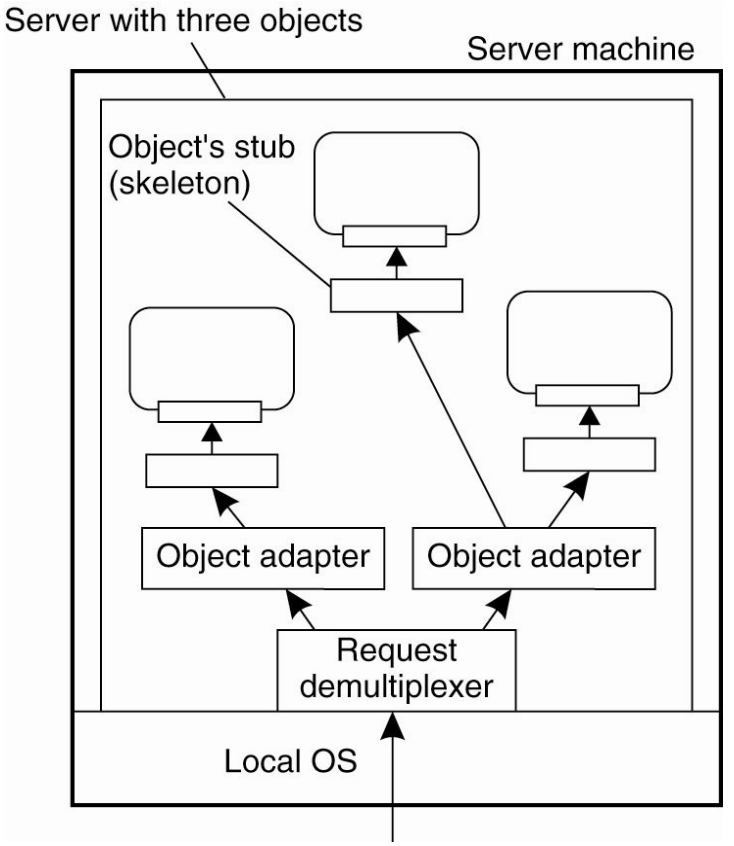
Endereços de memória:

- Compartilhados (ou não) entre objetos
  - Compartilhamento de código e/ou de dados
- Uso de Threads
  - Servidor com um único thread de controle; Um thread para cada objeto; Um thread para cada invocação

Middleware de OD: adaptadores de objetos

"Software que implementa uma política de ativação"

- Tem um ou mais objetos sob seu controle
- Genérico: desenvolvedores o configuram para uma política específica
- Podem suportar várias políticas de ativação tão-somente ao configurá-las em tempo de execução
- Vários adaptadores podem residir no mesmo servidor simultanemente
- Podem operar em thread único ou multithread
- Entrega as requisições aos *skeletons*, gerados de acordo com as definições do objeto



Middleware de OD: exemplo de adaptador

```
main(int argc, char* argv[]) {
    Ice::Communicator      ic;
    Ice::ObjectAdapter     adapter;
    Ice::Object            object;

    ic = Ice::initialize(argc, argv);
    adapter =  /*Criação do Adaptador de objetos*/
        ic->createObjectAdapterWithEnd Points( "MyAdapter","tcp -p 10000");
    object = new MyObject;  /*Novo objeto (genérico)*/
    adapter->add(object, objectID);  /*Adição ao adaptador*/
    adapter->activate();  /*Ativação do adaptador*/
    ic->waitForShutdown();
                           /*política de controle*/
}
```

Base do ambiente de execução. Gerencia Um reservatório de threads e meios para Configurar o ambiente. Por exemplo, tamanho máximo de resposta, número máximo de repetidas invocações,...

Conexões TCP

Porta 10000

The Ice Runtime System: criação de um objeto servidor

Vinculação a um objeto requisitado:

"A vinculação resulta na colocação de um proxy no espaço de endereços do processo, onde deve ser implementada uma interface que contém os métodos que o processo pode invocar"

"É necessário um meio para localizar o servidor que gerencia o objeto propriamente dito e colocar um proxy no espaço de endereços do cliente"

Tipos de Invocação:

- **Vinculação implícita:** invocação direta, usando somente a referência ao objeto. O cliente é vinculado com transparência após a resolução da referência, implementada pelo middleware.
- **Vinculação explícita:** em geral utiliza um método tipo "bind()" que retorna um ponteiro para o proxy que então fica disponível no local. Em seguida, a invocação deve ser executada com base no ponteiro retornado.

Middleware de Objetos Distribuídos: exemplo

```
Distr_object* obj_ref;          // Declare a systemwide object reference
obj_ref = ...;                  // Initialize the reference to a distrib. obj.
obj_ref→do_something( );        // Implicitly bind and invoke a method
```

Uma vinculação implícita (implicit binding) usando apenas refêrencias globais

```
Distr_object obj_ref;           // Declare a systemwide object reference
Local_object* obj_ptr;          // Declare a pointer to local objects
obj_ref = ...;                  // Initialize the reference to a distrib. obj.
obj_ptr = bind(obj_ref);        // Explicitly bind and get ptr to local proxy
obj_ptr→do_something( );        // Invoke a method on the local proxy
```

vinculação explícita que usa referências globais e locais.

(TANENBAUM & STEEN, 2008):

Referências de Objetos:

"Deve conter informações suficientes para possibilitar que um cliente se vincule a um objeto"

- **Servidor de localização**
  - Monitora a máquina em que o servidor de um objeto está executando no momento considerado.
  - Referência de objeto <end. do servidor de localização, identificador para o servidor no âmbito do sistema>
    - Pode incluir como parâmetros um **Manipulador de Implementação:** uma referência para uma implementação completa de um proxy que o cliente pode carregar dinamicamente quando estiver se vinculando ao objeto
      - Vantagem: o cliente não precisa se preocupar se tem à disposição um proxy específico. Por exemplo, que implementa um protocolo específico.

Invocações do método remoto

- **Invocação Estática**
  - Requer que as interfaces de um objeto sejam conhecidas quando a aplicação do cliente está em desenvolvimento (tempo de compilação)
  - Uma linguagem de definição de Interfaces (IDL) e um compilador para a geração dos apêndices.

- **Invocação Dinâmica**
  - Construir uma invocação de método em tempo de execução

Middleware de Objetos Distribuídos: comunicação

Invocação estática

fobjetc.append(int)

//invocando o método *append()* de um objeto *fobjetc.*

invoke(fobject, id(append), int)

// *id(append)* retorna um identificador para o método *append()*

Invocação dinâmica

Sintaxe:

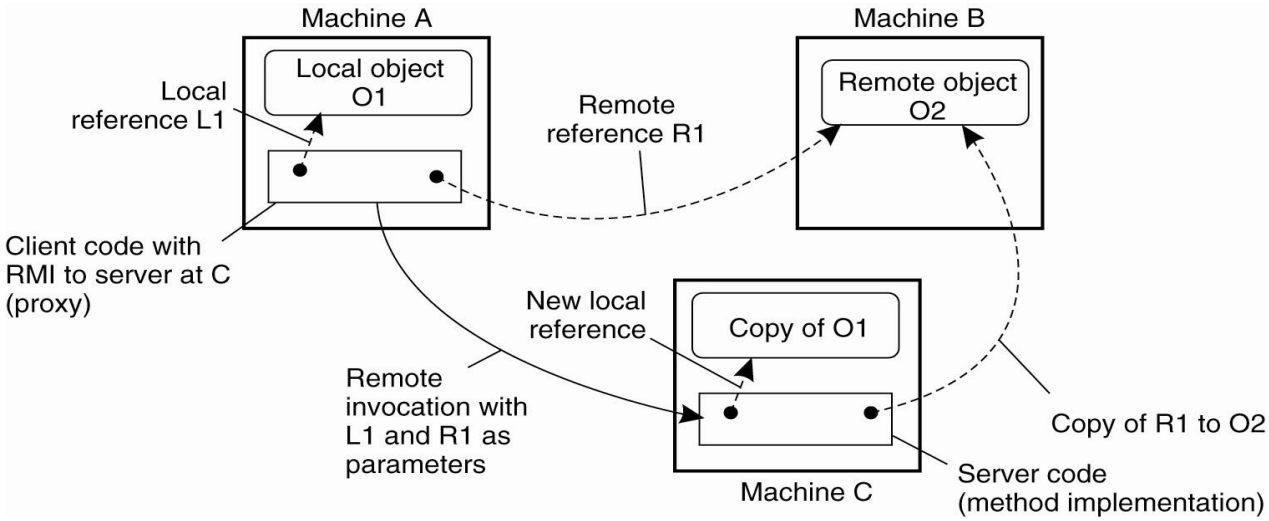
Invoke (objeto, método, parâmetros-de-entrada, parâmetros-de-saída)

- objeto: identifica o objeto distribuído
- método: especifica qual método deve ser invocado
- parâmetros-de-entrada: é uma estrutura de dados que contém os valores dos parâmetros de entrada daquele método
- parametros-de-saída: uma estrutura de dados na qual os parâmetros de saída podem ser armazenados

Middleware de Objetos Distribuídos: comunicação

Passagem de parâmetros

- **Referência a um objeto local**
  - Passagem por valor, ou seja, objeto é copiado e passado junto com a invocação (serialização de objetos)
- **Referência a um objeto remoto**
  - Passagem por referência, ou seja, a referência é copiada e transferida como parâmetro.



Middleware de Objetos Distribuídos: Java RMI

- **Objeto remoto:** reside em outra máquina virtual
- **Interfaces:** são implementadas por um proxy, que oferece exatamente as mesmas interfaces e aparece como um objeto local no espaço do cliente
- **Clonagem**
  - **de objeto local:** novo objeto do mesmo tipo com o mesmo estado
  - **de objeto remoto** não é suportado!
    - Uma cópia do objeto remoto exige clonar o objeto em seu servidor e também o proxy em cada cliente no momento considerado
    - O acesso a um objeto clonado no servidor exige nova vinculação do cliente

Invocação

- Objeto local é passado por valor
  - Quando um objeto é passado por valor, um novo objeto é criado no processo destino
  - Os métodos desse novo objeto podem ser invocados de forma local, possivelmente fazendo com que seu estado seja diferente do estado do objeto original no processo remetente.
  - Serialização de objetos

Person p = new Person("Smith", "London", 1934);

Person	8-byte version number		h0	*nome da classe, número da versão*
3	int year	java.lang.String name:	java.lang.String place:	*Número, tipo de variáveis*
1934	5 Smith	6 London	h1	*Valores da variáveis*

- Objeto remoto é passado por referência

  <endereço de rede, porta, identificador local para o objeto>

Middleware de Objetos Distribuídos: Java RMI

Invocação

- Skeleton é gerado de acordo com as especificações do objeto
- Proxy (ou stub): converte as invocações em mensagens; estabelece conexão com o servidor (uma para cada chamada); são serializáveis, ou seja, podem ser usados como referência para um objeto remoto
- Manipulador de implementação: especifica quais classes são necessárias para construir o Proxy. É utilizado na serialização de um proxy para diminuir a quantidade de dados a ser transmitido. Substitui parte do código montado como parte de uma referência a um objeto remoto

Middleware de Objetos Distribuídos: Java RMI

Um objeto remoto precisa implementar a interface ***Remote*** e estender a classe ***rmi.server.RemoteServer*** ou uma sua descendente, como a ***UnicastRemote Object***.

Todos objetos que são parâmetros ou retorno de métodos remotos devem implementar a interface ***Serializable***

- Em java, qualquer objeto que pode ser serilizado pode ser passado como parâmetro*, basta (...implements Serializable)*

Todos os métodos desta interface, e que portanto podem ser invocados remotamente, devem declarar a excessão ***RemoteException***.

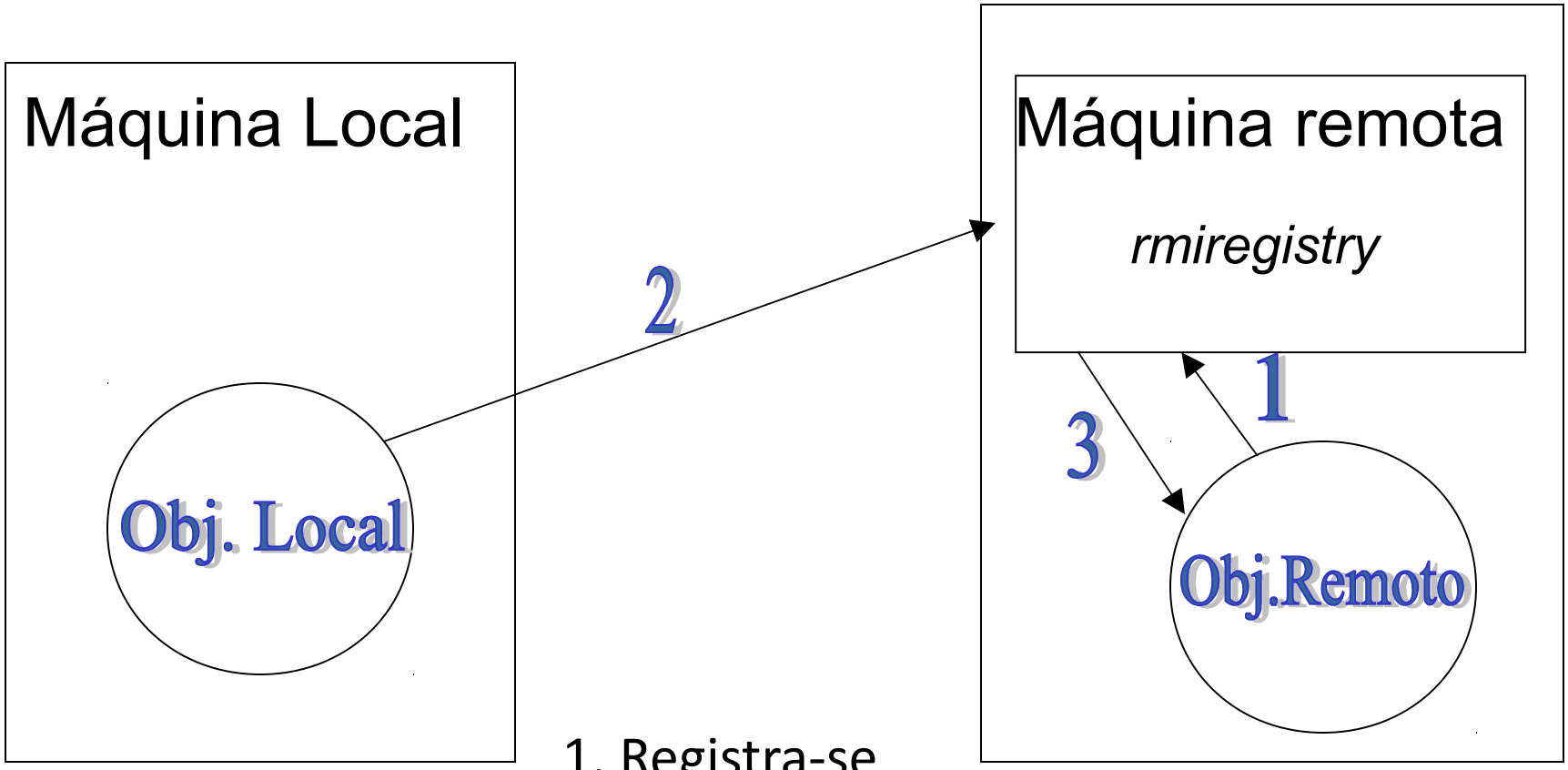
Vinculação:

Para que um objeto remoto exporte sua interface para outros objetos, ele precisa registrar-se num servidor chamado ***rmiregistry***.

Este servidor reside ***na mesma máquina*** do objeto remoto.

o objeto cliente consulta primeiramente o **vinculador** para obter uma referência do objeto remoto.

Middleware de Objetos Distribuídos: Java RMI



1. Registra-se
2. Solicita referência.
3. Acessa

Middleware de Objetos Distribuídos: exemplo

Servidor de tempo:

- Disponibiliza o serviço pelo objeto remoto *CalendarImpl*
- O nome do método de acesso remoto é *getDate()*

Middleware de Objetos Distribuídos: exemplo

```
import java.rmi.* ;


public interface iCalendar extends Remote
{
  java.util.Date getDate()
  throws RemoteException ;
}
```

Middleware de Objetos Distribuídos: exemplo

```
import java.util.Date ;
import java.rmi.* ;         // classes básicas RMI
import java.rmi.registry.* ; // registrar métodos na rede
import java.rmi.server.* ;   // realizar a "escuta“

public class CalendarImpl extends UnicastRemoteObject  implements
     iCalendar {

public CalendarImpl ( )  throws RemoteException
{ System.out.println ("Server object was created.") ; }

  public Date getDate ( )  throws RemoteException
{ System.out.println ("Calendar server accessed.") ;
   return new Date () ;  }
```

Continua...

Middleware de Objetos Distribuídos: exemplo

```
public static void main (String args []) {
CalendarImpl cal ;
try{ LocateRegistry.createRegistry (1299);
//usada para registrar um objeto remoto e aceita chamadas numa determinada porta
      cal = new CalendarImpl () ;
      Naming.bind ("rmi://CalendarImpl", cal) ;
      System.out.println  ("Server object is ready for RMIs") ;}
      catch (Exception e) {e.printStackTrace () ;}
                                                    }
}//finaliza o programa
```

Middleware de Objetos Distribuídos: vinculação

"Uma instância do RMIRegistry deve ser executada em cada computador que contenha objetos remotos"

- Mantém uma tabela mapeando nomes em referências para objetos remotos contidos em cada computador
- Acessado por métodos da classe Naming e recebem como argumento:

rmi://nomeComputador:porta/nomeObjeto

nomeComputador e porta se referem à localização do RMIRegistry;Caso sejam omitido, considera-se localhost e porta padrão.

Middleware de Objetos Distribuídos: vinculação

Alguns métodos para Vinculação:

- void rebind (String nome, Remote obj)
  - Usado por um servidor para registrar o identificador de um objeto remoto pelo nome.
- void bind (String nome, Remote obj)
  - Usado alternativamente por um servidor para registrar um objeto remoto pelo nome. Mas, se o nome já estiver vinculado a uma referência de objeto remoto, uma exceção será disparada.
- void unbind (String nome, Remote obj)
  - Remove um vínculo
- void lookup (String nome)
  - Usado pelos clientes para procurar um objeto remoto pelo nome. É retornada uma referência de objeto remoto (vinculação explícita)

Middleware de Objetos Distribuídos: exemplo

```
import java.util.Date ;
import java.rmi.* ;
public class CalendarUser {
  public static void main (String args[]){
 long t1=0, t2=0 ;  Date date ;  iCalendar remoteCal ;
 try{
  remoteCal = (iCalendar) Naming.lookup  ("rmi://localhost/CalendarImpl") ;
              t1 = remoteCal.getDate().getTime() ;
              t2 = remoteCal.getDate().getTime() ;}
      catch (Exception e){ e.printStackTrace () ;}
  System.out.println ("This RMI call took " + (t2 -t1) + " milliseconds") ;
                                                  }
}
```

Middleware de Objetos Distribuídos: exemplo

O próximo passo é compilar o cliente e o servidor com o compilador javac, da maneira usual:

```
javac  *.java
```

Depois, deve-se usar o compilador rmi, o rmic, para gerar o ***stub*** e o ***skeleton***:

```
rmic  CalendarImpl
```

Middleware de Objetos Distribuídos: exemplo

O compilador rmic gera os seguintes arquivos:

CalendarImpl_Skel.class

CalendarImpl_Stub.class

O próximo passo é iniciar o servidor de registro na máquina onde o objeto remoto vai ser executado:

rmiregistry

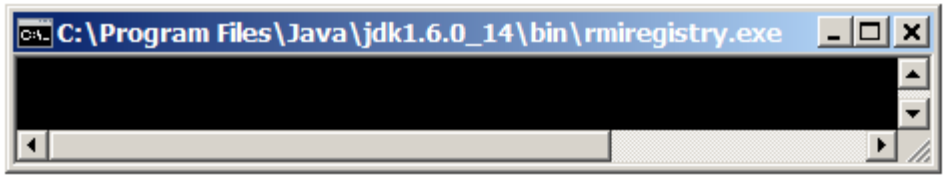
Em seguida, iniciar o objeto remoto e o objeto cliente

java CalendarImpl (objeto remoto)

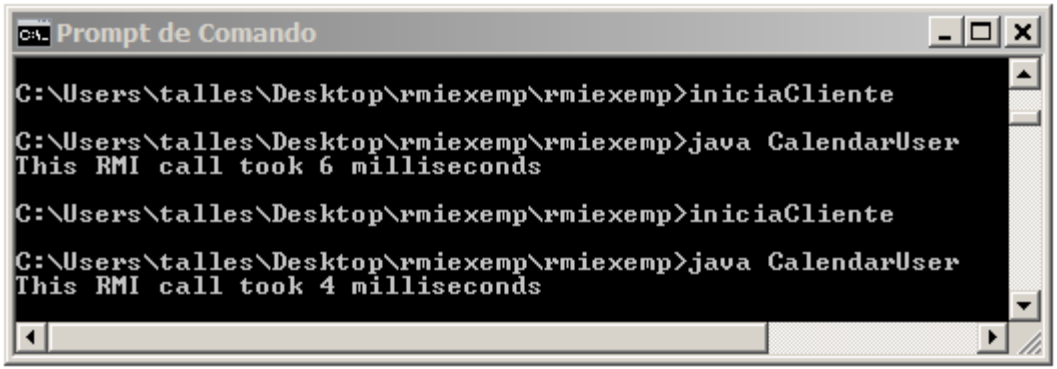
java CalendarUser (objeto cliente)

Middleware de Objetos Distribuídos: exemplo

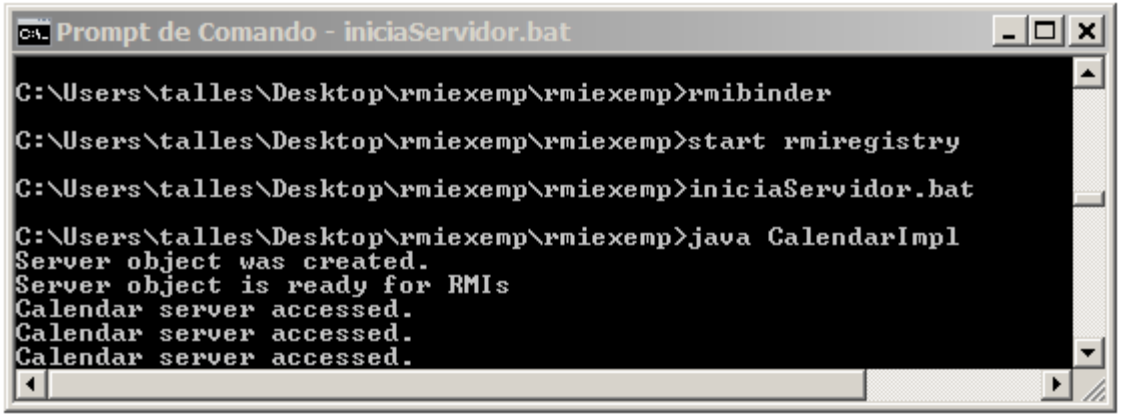
Resultado



ativação do RMIregistry



ativação do cliente



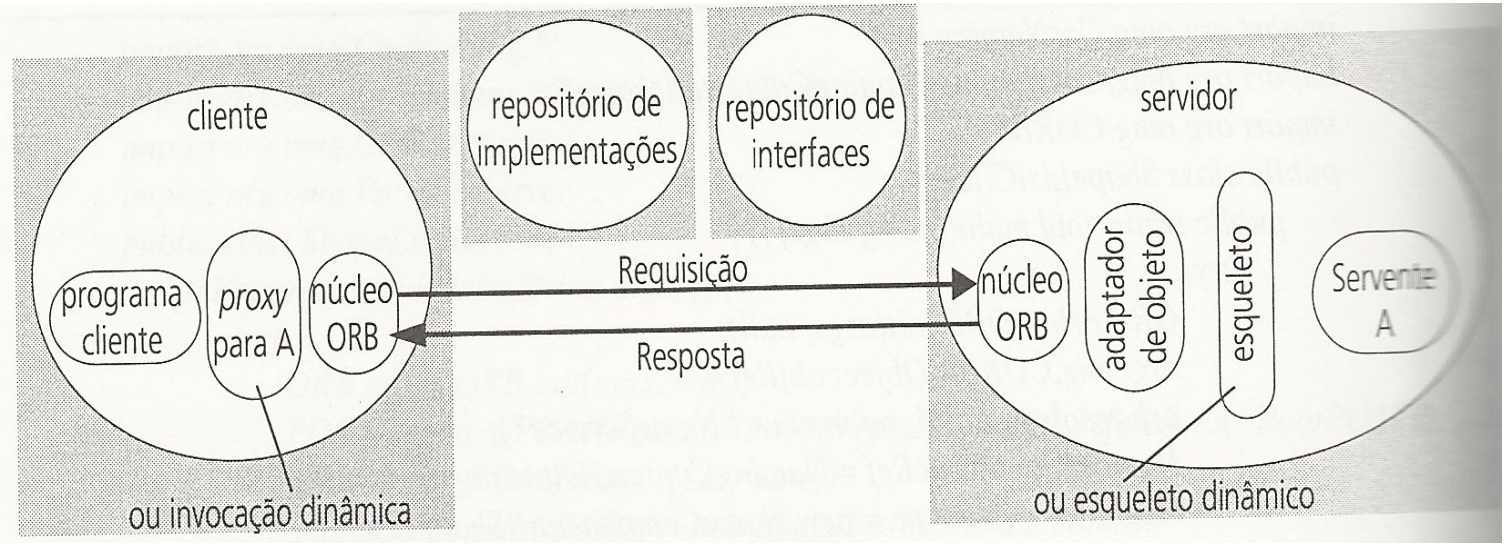
ativação do objeto remoto

Middleware de Objetos Distribuídos: CORBA

Em 1989, criada a OMG (*Object Management Group*)

- Estimular a adoção de sistemas de objetos distribuídos
  - Sistemas Abertos baseados em interfaces orientadas a objetos
  - Permitir que objetos distribuídos fossem implementados em qualquer linguagem de programação
- Em 1991, o CORBA (*Common Object Request Broker Architecture*)
- Em 1996, a especificação CORBA 2.0
  - Com ela o *General Inter-ORB Protocol ou GIOP*
- Em 1999, a especificação CORBA 3
  - Uma linguagem de definição de interface IDL
  - Uma arquitetura
  - O protocolo GIOP
  - Implementação do GIOP conhecida como IIOP (*Inter-ORB Protocol)*

Middleware de Objetos Distribuídos: CORBA



- Adaptadores de Objeto
  - Cria as referências de objeto remoto para objetos CORBA
  - Envia cada RMI, por meio de um esqueleto

Middleware de Objetos Distribuídos: CORBA

- Alternativa ao RMI
- Um exemplo: Sistemas Corba
  - Combina invocação de método com comunicação orientada a mensagens
- Duas forma de invocações assíncronas de métodos
  - Modelo de chamada de retorno

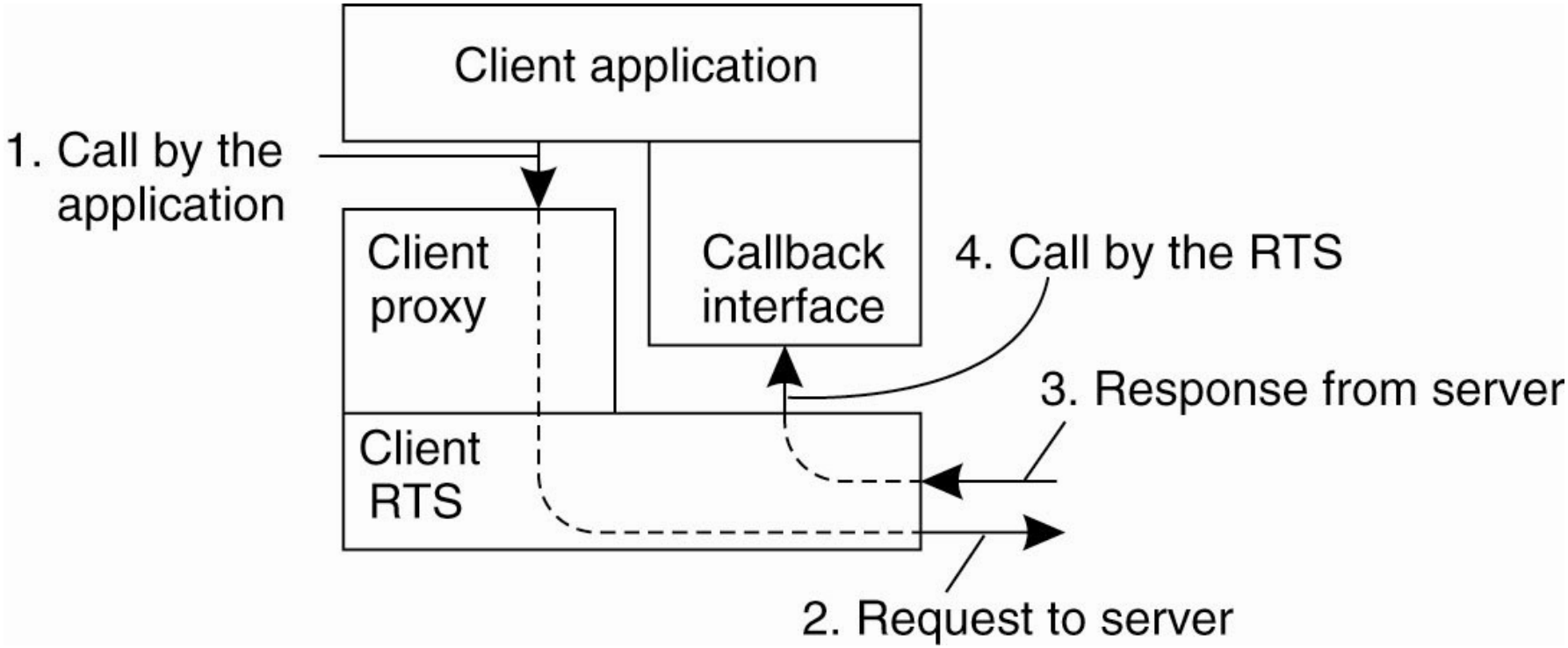
    Um cliente fornece um objeto para implementar uma interface que contém métodos de chamada de retorno. Esses métodos podem ser chamados pelo sistema de comunicação para passar o resultado de uma invocação assíncrona. Ou seja, cabe ao cliente a responsabilidade de transformar a invocação síncrona original em uma assíncrona; ao servidor é apresentada uma requisição de invocação normal (síncrona).

Middleware de Objetos Distribuídos: CORBA

A construção de uma chamada assíncrona é feita em duas etapas:

- A interface original é substituida por duas novas interfaces que devem ser implementadas por software do lado do cliente (apenas)
- Uma interface para especificação dos métodos. Nenhum dos métodos retorna valor ou tem qualquer parâmetro de saída
- Interface de chamada de retorno: para cada operação na interface original , ela contém um método que será chamado pelo sistema de execução do cliente para passar os resultados do método associado como chamado pelo cliente

Middleware de Objetos Distribuídos: CORBA



O Modelo de Consulta

- Um conjunto de operações é fornecido ao cliente para consultar seu sistema de execução (RTS) local para os resultados que chegam
- O cliente também é responsável por transformar as invocações síncronas em assíncronas
- A diferença reside no fato do RTS do cliente ter que implementar a interface de *polling* do cliente, que poderá ser gerada automaticamente com base nas especificações da interface