5 – Invocação Remota

Prof. Marco Aurélio S Birchal

PUC Minas

5 – Invocação Remota

5.1 Introdução

5.2 Protocolos de requisição-resposta

5.3 Chamada de procedimento remoto

5.4 Invocação a método remoto

5.5 Estudo de caso: RMI Java

5.1 Invocação Remota - Introdução

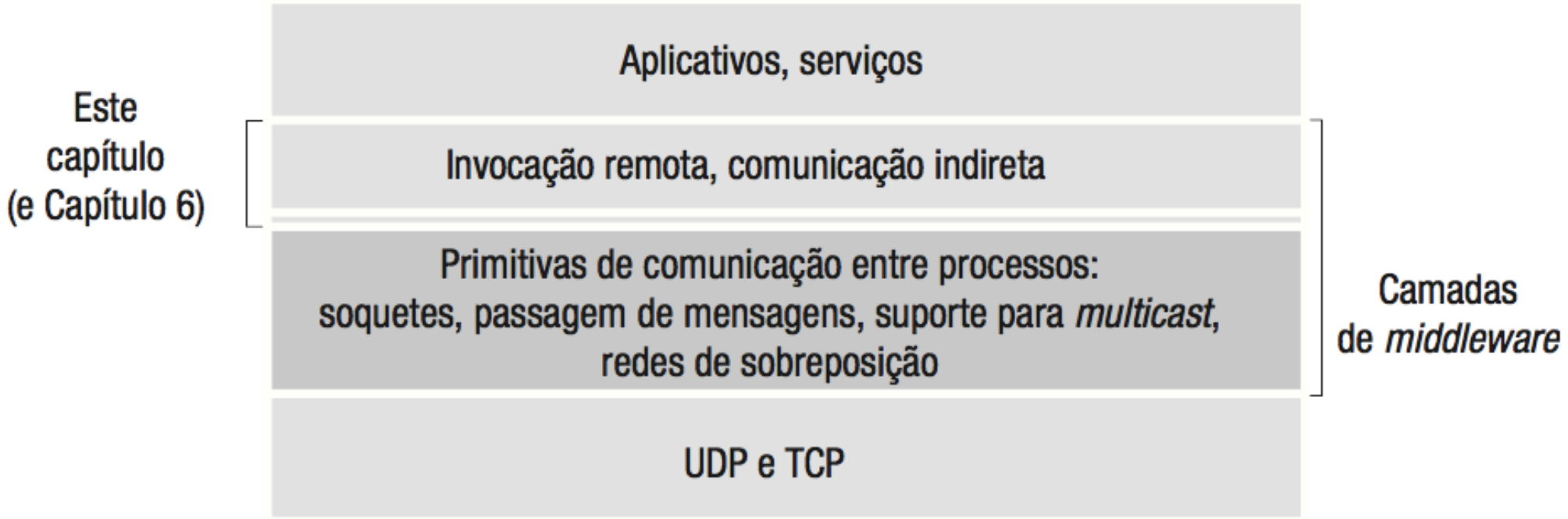


Figura 5.1 Camadas de *middleware*.

- Protocolos de requisição-resposta (serviço mais primitivo)
- Chamada de procedimento remoto (RPC)
- Invocação a método remoto (RMI)

5.2 Requisição-Resposta (Request-Reply)

- Representam um padrão sobre a passagem de mensagens e suportam a troca bilateral de mensagens, como a encontrada na computação cliente-servidor.
- Em particular, tais protocolos fornecem suporte de nível relativamente baixo para solicitar a execução de uma operação remota.
- No caso normal, a comunicação por requisição-resposta é síncrona, pois o processo cliente é bloqueado até que a resposta do servidor chegue.
- Ela também pode ser confiável, pois a resposta do servidor é efetivamente uma confirmação para o cliente.
- A comunicação por requisição-resposta assíncrona é uma alternativa útil em situações em que os clientes podem recuperar as respostas posteriormente

5.2 Requisição-Resposta (Request-Reply)

- As trocas entre cliente e servidor estão aqui descritas em termos das operações *send* e *receive* para datagramas UDP, embora muitas implementações atuais usem TCP. UDP evita sobrecargas desnecessárias associadas ao protocolo TCP. Em particular:
    - As confirmações são redundantes, pois as requisições são seguidas por respostas.
    - O estabelecimento de uma conexão envolve dois pares extras de mensagens, além do par exigido por uma requisição e uma resposta.
    - O controle de fluxo é redundante para a maioria das invocações, que passam apenas pequenos argumentos e resultados.

5.2 Requisição-Resposta (Request-Reply)

O protocolo aqui descrito é baseado em um trio de primitivas de comunicação: doOperation, getRequest e sendReply,

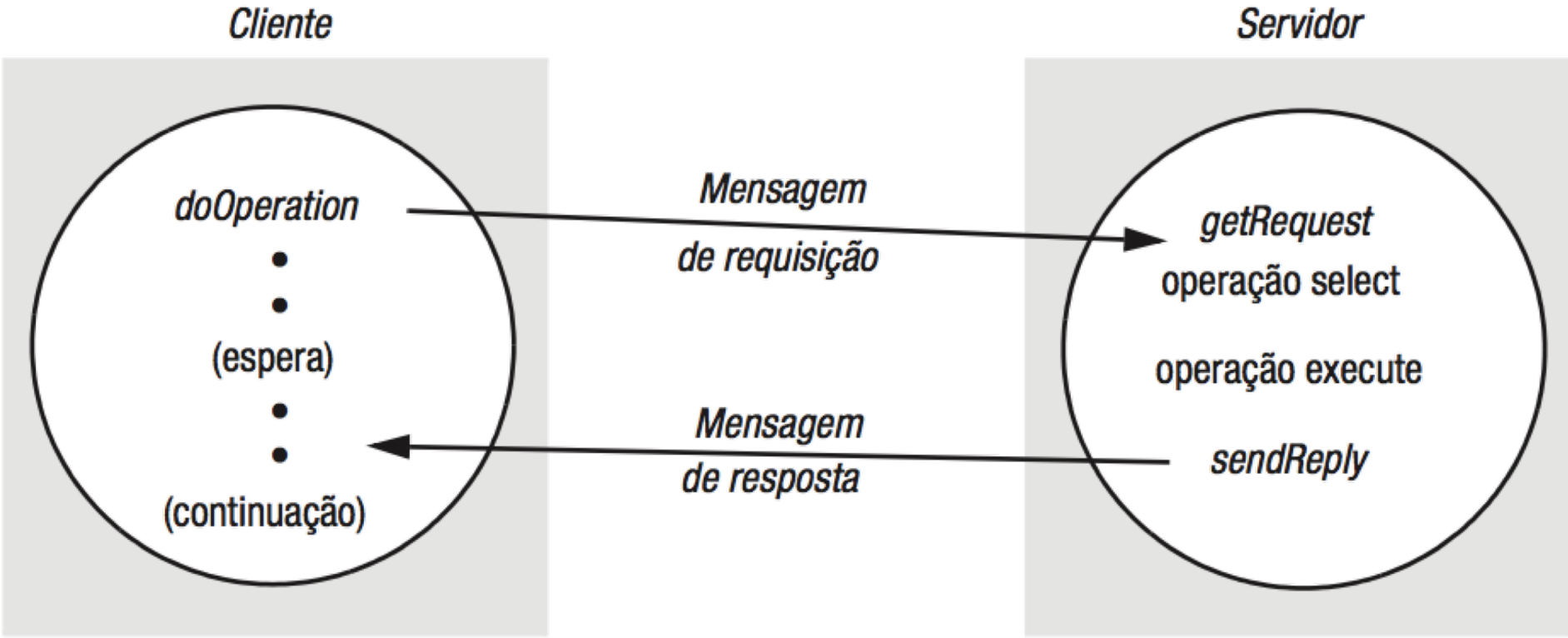


Figura 5.2 Comunicação por requisição-resposta.

5.2 Requisição-Resposta (Request-Reply)

public byte[ ] doOperation (RemoteRef s, int operationId, byte[ ] arguments)

Envia uma mensagem de requisição para o servidor remoto e retorna a resposta (bloqueia o cliente até a resposta). Os argumentos especificam o servidor remoto, a operação a ser invocada e os argumentos dessa operação.

public byte[ ] getRequest ( );

Lê uma requisição do cliente por meio da porta do servidor.

public void sendReply (byte[ ] reply, InetAddress clientHost, int clientPort);

Envia a mensagem de resposta reply para o cliente como seu endereço de Internet e porta.

5.2 Requisição-Resposta (Request-Reply)

Estrutura da Mensagem

messageType	*int (0=Request, 1=Reply)*
requestId	*int*
remoteReference	*RemoteRef*
OperationId	*int ou operação*
arguments	*// vetor de bytes*

1º Campo: Tipo da mensagem; se Request ou Reply

5.2 Requisição-Resposta (Request-Reply)

Estrutura da Mensagem

messageType	*int (0=Request, 1=Reply)*
requestId	*int*
remoteReference	*RemoteRef*
OperationId	*int ou operação*
arguments	*// vetor de bytes*

2º Campo: requestId – identificador da mensagem. O doOperation (cliente) gera o id exclusivo para cada mensagem. O servidor o copia e envia de volta. Permite que o cliente verifique se a resposta é da requisição atual ou uma resposta atrasada (de uma operação anterior).

5.2 Requisição-Resposta (Request-Reply)

Estrutura da Mensagem

messageType	*int (0=Request, 1=Reply)*
requestId	*int*
remoteReference	*RemoteRef*
OperationId	*int ou operação*
arguments	*// vetor de bytes*

2º Campo: requestId – Formatato. Duas partes:

- Um id sequencial gerado pelo cliente (torna o requestId exclusivo dentro do cliente)
- Um identificador do cliente (porta, IP). Torna o requestId exclusivo dentro do sistema distribuído.

requestId: #seq_no_cliente, porta_cliente + IP_cliente

5.2 Requisição-Resposta (Request-Reply)

Estrutura da Mensagem

messageType	*int (0=Request, 1=Reply)*
requestId	*int*
remoteReference	*RemoteRef*
OperationId	*int ou operação*
arguments	*// vetor de bytes*

3º Campo: Referência remota: IP e porta do servidor com o qual o cliente quer comunicar.

5.2 Requisição-Resposta (Request-Reply)

Estrutura da Mensagem

messageType	*int (0=Request, 1=Reply)*
requestId	*int*
remoteReference	*RemoteRef*
OperationId	*int ou operação*
arguments	*// vetor de bytes*

4º Campo: identificador da operação (OperationId) a ser invocada. Por exemplo, as operações de uma interface poderiam ser numeradas como 1, 2, 3,...;

5.2 Requisição-Resposta (Request-Reply)

Estrutura da Mensagem

messageType	*int (0=Request, 1=Reply)*
requestId	*int*
remoteReference	*RemoteRef*
OperationId	*int ou operação*
arguments	*// vetor de bytes*

5º Campo: arguments. Argumentos da operação à ser invocada no servidor.

5.2 Requisição-Resposta (Request-Reply)

Modelo de falhas do protocolo de requisição-resposta

Se as três primitivas doOperation, getRequest e sendReply forem implementadas sobre datagramas UDP, elas sofrerão das mesmas falhas de comunicação. Ou seja:

- Sofrerão de falhas por omissão.
- Não há garantia de entrega das mensagens na ordem do envio.

Além disso, o protocolo pode sofrer uma falha de processos (se servidor falhou ou uma mensagem de requisição ou resposta é perdida, doOperation usa um tempo limite – *timeout* - quando está esperando receber a mensagem de resposta do servidor.)

5.2 Requisição-Resposta (Request-Reply)

Timeout – Tempo Limite

Após a ocorrência de um timeout, o doOperation pode:

- Simplesmente retornar imediatamente após o erro indicando para o cliente que a operação falhou (não é o mais comum pois o timeout pode ter sido por causa da requisição ou da resposta e, neste caso, a operação foi executada no servidor).
- Enviar a mensagem de requisição de novo (repetidas vezes) até receber uma resposta (se havia tido perda da resposta) ou estar razoavelmente seguro da falta de resposta (falha no servidor).

O retorno do doOperation, nesses casos, será através de exceção, indicando o tipo de falha para o cliente.

5.2 Requisição-Resposta (Request-Reply)

Descarte de Mensagem de Requisição duplicada

Nos casos em que a mensagem de requisição é retransmitida, o servidor pode recebê-la mais de uma vez:

- Por exemplo, o servidor pode receber a primeira mensagem de requisição, mas demorar mais do que o tempo limite do cliente para executar o comando e retornar a resposta. Isso pode levar o servidor a executar uma operação mais de uma vez para a mesma requisição.

Para evitar isso, o protocolo é projetado de forma a reconhecer mensagens sucessivas (do mesmo cliente) com o mesmo identificador de requisição e eliminar as duplicatas. Se o servidor ainda não enviou a resposta, não precisa executar nenhuma ação especial – ele transmitirá a resposta quando tiver terminado de executar a operação.

5.2 Requisição-Resposta (Request-Reply)

Mensagens de resposta perdidas

(e o problema da idem potência)

- Se o servidor já tiver enviado a resposta quando receber uma requisição duplicada, precisará executar a operação novamente para obter o resultado, a não ser que tenha armazenado o resultado da execução original.

- Alguns servidores podem executar suas operações mais de uma vez e obter os mesmos resultados.

5.2 Requisição-Resposta (Request-Reply)

Mensagens de resposta perdidas

(e o problema da idem potência)

Operação idempotente é aquela que pode ser efetuada repetidamente com o mesmo efeito, como se tivesse sido executada exatamente uma vez.

Por exemplo, uma operação para pôr um elemento em um conjunto é idempotente, pois sempre terá o mesmo efeito no conjunto, toda vez que for executada, enquanto uma operação para incluir um elemento em uma sequência não é idempotente, pois amplia a sequência sempre que é executada.

Um servidor cujas operações são todas idempotentes não precisa adotar medidas especiais para evitar suas execuções mais de uma vez.

5.2 Requisição-Resposta (Request-Reply)

Histórico

- Para os servidores que exigem retransmissão das respostas sem executar novamente as operações, pode-se usar um histórico.
- O termo histórico é usado para se referir a uma estrutura que contém um registro das mensagens (respostas) que foram transmitidas.
- Uma entrada em um histórico contém um identificador de requisição, uma mensagem e um identificador do cliente para o qual ela foi enviada.

5.2 Requisição-Resposta (Request-Reply)

Histórico

- Seu objetivo é permitir que o servidor retransmita as mensagens de resposta quando os processos clientes as solicitarem.
- Um problema associado ao uso de um histórico é seu consumo de memória.
- Um histórico pode se tornar muito grande, a menos que o servidor possa identificar quando não há mais necessidade de retransmissão das mensagens.

5.2 Requisição-Resposta (Request-Reply)

Estilos de Protocolos de Troca

Três protocolos, que produzem diferentes comportamentos na presença de falhas de comunicação, são usados para implementar vários tipos de comportamento de requisição. Eles foram identificados originalmente por Spector [1982]:

- o protocolo request (R);
- o protocolo request-reply (RR);
- o protocolo request-reply-acknowledge reply (RRA).

5.2 Requisição-Resposta (Request-Reply)

Estilos de Protocolos de Troca

Nome	*Mensagens enviadas pelo*		
	Cliente	*Servidor*	*Cliente*
R	*Requisição*		
RR	*Requisição*	*Resposta*	
RRA	*Requisição*	*Resposta*	*Resposta de confirmação*

5.2 Requisição-Resposta (Request-Reply)

Estilos de Protocolos de Troca

No **protocolo R,** uma única mensagem Request é enviada pelo cliente para o servidor.

- Usado quando não existe nenhum valor a ser retornado do método remoto e o cliente não exige confirmação de que a operação foi executada.
- O cliente pode prosseguir imediatamente após a mensagem de requisição ser enviada, pois não há necessidade de esperar por uma mensagem de resposta.
- Esse protocolo é implementado sobre datagramas UDP e, portanto, sofre das mesmas falhas de comunicação.

5.2 Requisição-Resposta (Request-Reply)

Estilos de Protocolos de Troca

O **protocolo RR** é útil para a maioria das trocas cliente-servidor, pois é baseado no protocolo de requisição-resposta.

- Não são exigidas mensagens de confirmação especiais, pois uma mensagem de resposta (reply) do servidor é considerada como confirmação do recebimento da mensagem de requisição (request) do cliente.
- Analogamente, uma chamada subsequente de um cliente pode ser considerada como uma confirmação da mensagem de resposta de um servidor.
- As falhas de comunicação podem ser tratadas com um histórico para retransmissão.

5.2 Requisição-Resposta (Request-Reply)

Estilos de Protocolos de Troca

O **protocolo RRA** é baseado na troca de três mensagens: requisição, resposta e confirmação.

- A mensagem de confirmação contem o requestId da mensagem de resposta que está sendo confirmada.
- Isso permitirá que o servidor descarte entradas de seu histórico.
- A chegada de um requestId em uma mensagem de confirmação será interpretada como a acusação do recebimento de todas as mensagens de resposta com valores de requestId menores; portanto, a perda de uma mensagem de confirmação não é muito prejudicial ao sistema.

5.2 Requisição-Resposta (Request-Reply)

HTTP: protocolo requisição-resposta

- As requisições do cliente especificam um URL que inclui o nome DNS de um servidor Web e um número de porta opcional no servidor Web, assim como o identificador de um recurso nesse servidor.
- O protocolo HTTP especifica as mensagens envolvidas em uma troca de requisição-resposta, os métodos, os argumentos, os resultados e as regras para representá-los (empacotá-los) nas mensagens.
- Ele suporta um conjunto fixo de métodos (GET, PUT, POST, etc.) que são aplicáveis a todos os seus recursos.

5.2 Requisição-Resposta (Request-Reply)

HTTP: protocolo requisição-resposta

- O protocolo HTTP é implementado sobre TCP. Na versão original do protocolo, cada interação cliente-servidor consiste nas seguintes etapas:
  - O cliente solicita uma conexão com o servidor na porta HTTP padrão ou em uma porta especificada no URL.
  - O cliente envia uma mensagem de requisição para o servidor.
  - O servidor envia uma mensagem de resposta para o cliente.
  - A conexão é fechada.

Entretanto, estabelecer e fechar uma conexão para cada troca de requisição-resposta é dispendioso, sobrecarregando o servidor e causando o envio de muitas mensagens pela rede. A versão HTTP 1.1, usa conexões persistentes .

5.2 Requisição-Resposta (Request-Reply)

HTTP: protocolo requisição-resposta

- Uma conexão persistente pode ser encerrada a qualquer momento, tanto pelo cliente como pelo servidor, pelo envio de uma indicação para o outro participante.
- O navegador enviará novamente as requisições, sem envolvimento do usuário, desde que as operações envolvidas sejam idempotentes.
- Por exemplo, o método GET, descrito a seguir, é idempotente. Onde estão envolvidas operações não idempotentes, o navegador deve consultar o usuário para saber o que fazer em seguida.

5.2 Requisição-Resposta (Request-Reply)

HTTP: protocolo requisição-resposta

- GET: solicita o recurso cujo URL é dado como argumento. Pode ser usado para enviar o conteúdo de um formulário como argumento para um programa.
- HEAD: esta requisição é idêntica a GET, mas não retorna nenhum dado. Entretanto, retorna todas as informações sobre os dados, como a hora da última modificação, seu tipo ou seu tamanho.

5.2 Requisição-Resposta (Request-Reply)

HTTP: protocolo requisição-resposta

- POST: especifica o URL de um recurso (por exemplo, um programa) que pode tratar dos dados fornecidos no corpo do pedido. O processamento executado nos dados depende da função do programa especificado no URL.
- PUT: solicita que os dados fornecidos na requisição sejam armazenados no URL informado, como uma modificação de um recurso já existente ou como um novo recurso.

5.2 Requisição-Resposta (Request-Reply)

HTTP: protocolo requisição-resposta

- DELETE: o servidor exclui o recurso identificado pelo URL fornecido. Nem sempre os servidores permitem essa operação; nesse caso, a resposta indicará a falha.
- OPTIONS: o servidor fornece ao cliente uma lista de métodos que podem ser aplicados no URL dado (por exemplo, GET, HEAD, PUT) e seus requisitos especiais.
- TRACE: o servidor envia de volta a mensagem de requisição. Usado para propósi-tos de diagnóstico.

5.2 Requisição-Resposta (Request-Reply)

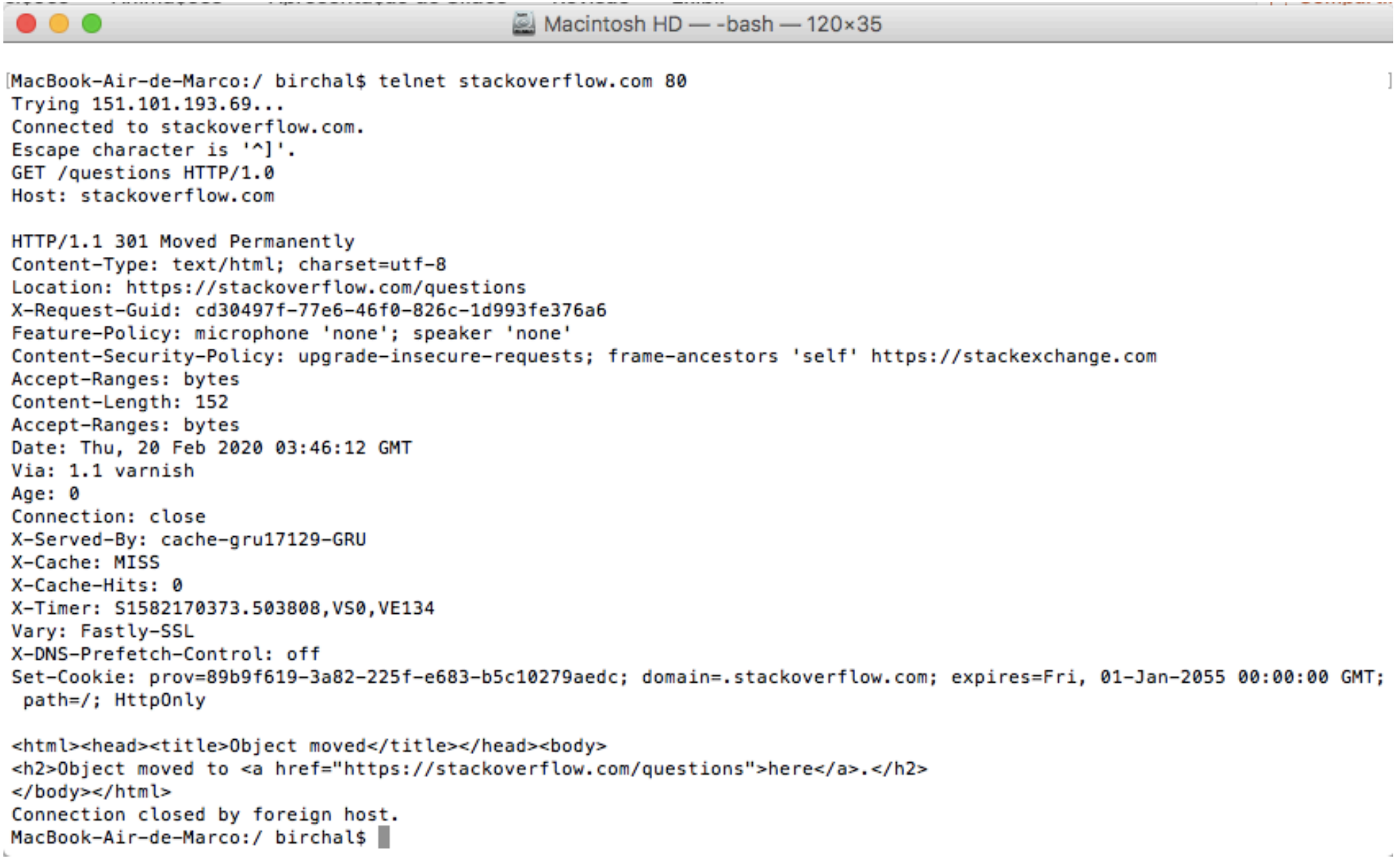
HTTP: protocolo requisição-resposta

método	*URL ou nome de caminho*	*versão de HTTP*	*cabeçalhos*	*corpo da mensagem*
GET	http://www.dcs.qmul.ac.uk/index.html	HTTP/ 1.1		

Figura 5.6 Mensagem de requisição HTTP.

versão de HTTP	*código de status*	*motivo*	*cabeçalhos*	*corpo da mensagem*
HTTP/1.1	200	OK		dados de recurso

Figura 5.7 Mensagem de resposta HTTP.


Macintosh HD — -bash — 120×35
MacBook-Air-de-Marco:/ birchal$ telnet stackoverflow.com 80
Trying 151.101.193.69...
Connected to stackoverflow.com.
Escape character is '^]'.
GET /questions HTTP/1.0
Host: stackoverflow.com
HTTP/1.1 301 Moved Permanently
Content-Type: text/html; charset=utf-8
Location: https://stackoverflow.com/questions
X-Request-Guid: cd30497f-77e6-46f0-826c-1d993fe376a6
Feature-Policy: microphone 'none'; speaker 'none'
Content-Security-Policy: upgrade-insecure-requests; frame-ancestors 'self' https://stackexchange.com
Accept-Ranges: bytes
Content-Length: 152
Accept-Ranges: bytes
Date: Thu, 20 Feb 2020 03:46:12 GMT
Via: 1.1 varnish
Age: 0
Connection: close
X-Served-By: cache-gru17129-GRU
X-Cache: MISS
X-Cache-Hits: 0
X-Timer: S1582170373.503808,VS0,VE134
Vary: Fastly-SSL
X-DNS-Prefetch-Control: off
Set-Cookie: prov=89b9f619-3a82-225f-e683-b5c10279aedc; domain=.stackoverflow.com; expires=Fri, 01-Jan-2055 00:00:00 GMT;
path=/; HttpOnly
<html><head><title>Object moved</title></head><body>
<h2>Object moved to here.</h2>
</body></html>
Connection closed by foreign host.
MacBook-Air-de-Marco:/ birchal$

Invocação Remota

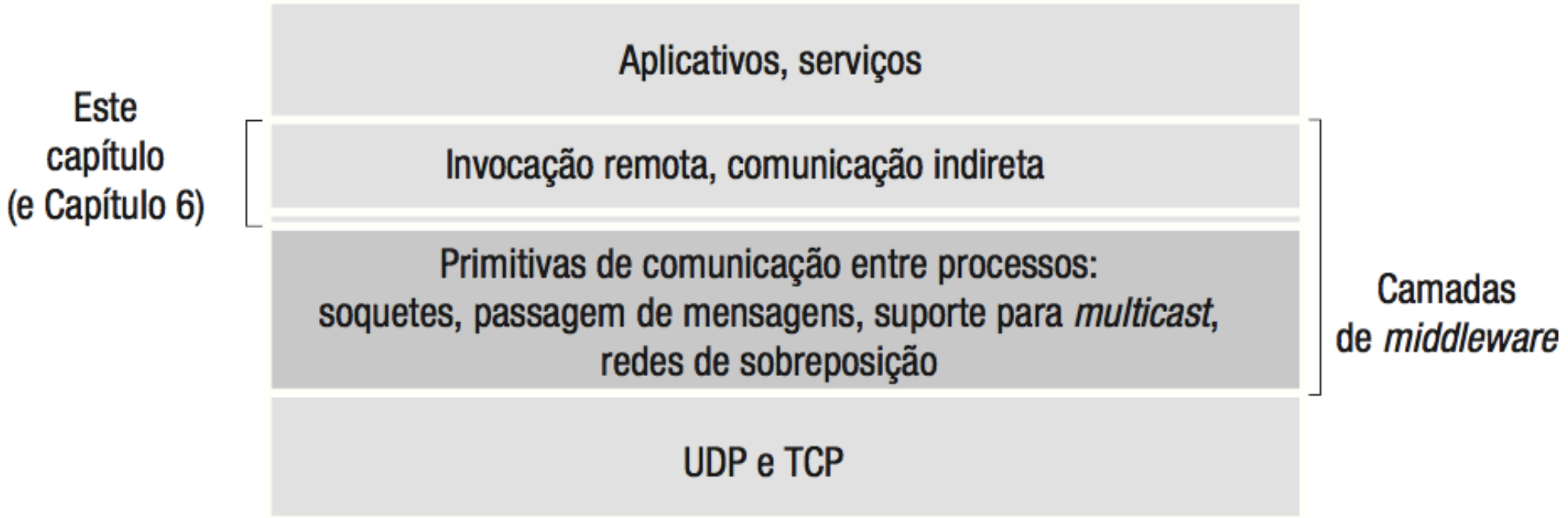


Figura 5.1 Camadas de *middleware*.

- Protocolos de requisição-resposta (serviço mais primitivo)
- Chamada de procedimento remoto (RPC)
- Invocação a método remoto (RMI)

5.3 Chamada de Procedimento Remoto (RPC)

Introdução

- O mais antigo, e talvez mais conhecido exemplo de modelo mais amigável para o programador foi a extensão do modelo de chamada de procedimentos convencional para os sistemas distribuídos, a chamada de procedimento remoto (RPC, *Remote Procedure Call*), que permite aos programas clientes chamarem procedimentos de forma transparente em programas servidores que estejam sendo executados em processos separados e, geralmente, em computadores diferentes do cliente.

5.3 Chamada de Procedimento Remoto (RPC)

Introdução

- um importante avanço intelectual na computação distribuída, tornando a programação de sistemas distribuídos semelhante (se não idêntica) à programação convencional – isto é, obter transparência de distribuição de alto nível.
- Estende a abstração de chamada de procedimento para os ambientes distribuídos.
- Os procedimentos em máquinas remotas podem ser chamados como se fossem procedimentos no espaço de endereçamento local.

5.3 Chamada de Procedimento Remoto (RPC)

Introdução

- Oculta os aspectos importantes da distribuição, incluindo a codificação e a decodificação de parâmetros e resultados, a passagem de mensagens e a preservação da semântica exigida para a chamada de procedimento.
- Esse conceito foi apresentado pela primeira vez por Birrell e Nelson [1984] e abriu o caminho para muitos dos desenvolvimentos na programação de sistemas distribuídos utilizados atualmente.

5.3 Chamada de Procedimento Remoto (RPC)

Questões de Projeto

- o estilo de programação promovido pela RPC – programação com interfaces;
- a semântica de chamada associada à RPC;
- o problema da transparência e como ele se relaciona com as chamadas de procedimento remoto.

5.3 Chamada de Procedimento Remoto (RPC)

INTERFACES

- Linguagens de programação modernas permitem a criação de um programa como um conjunto de módulos ou objetos que comunicam entre si
- Interfaces definem o que pode ser acessado a partir de um objeto ou módulo remoto
  - em geral: métodos e variáveis
- Papel fundamental no encapsulamento
  - Permite ocultar a implementação de um método

5.3 Chamada de Procedimento Remoto (RPC)

INTERFACES

- A interface de um modulo especifica os procedimentos e as variáveis que podem ser acessadas a partir de outros módulos.
  - Em sistemas distribuídos: apenas métodos são acessíveis através de interfaces
- cada servidor fornece um conjunto de procedimentos que estão disponíveis para uso pelos clientes.
- interface de serviço: especificação dos procedimentos oferecidos por um servidor, definindo os tipos dos argumentos de cada um dos procedimentos.

5.3 Chamada de Procedimento Remoto (RPC)

INTERFACES

- Em sistemas distribuídos, as interfaces podem definidas como:
  - Interfaces de serviço para RPC
  - Interfaces remotas para RMI
- A diferença é que nas interfaces remotas objetos e referências de objetos remotos podem ser passados como parâmetros de entrada e saída
- Nem interfaces de serviço e nem interfaces remotas permitem acesso a variáveis.

5.3 Chamada de Procedimento Remoto (RPC)

IDL – LINGUAGEM DE DEFINIÇÃO DE INTERFACE

- Permitir que procedimentos implementados em diferentes linguagens invoquem uns aos outros.
- Uma IDL fornece uma notação para definir interfaces, na qual cada um dos parâmetros de uma operação pode ser descrito como sendo de entrada ou de saída, além de ter seu tipo especificado.

EXEMPLOS:

- XDR da Sun como exemplo de IDL para RPC;
- IDL do CORBA como exemplo de IDL para RMI;
- WSDL linguagem de descrição de serviços Web (Web Services Description Language).

5.3 Chamada de Procedimento Remoto (RPC)

IDL – LINGUAGEM DE DEFINIÇÃO DE INTERFACE

Exemplo de IDL simples em CORBA

```
// Arquivo de entrada Person.idl
struct Person {
        string name;
        string place;
        long year;
};
interface PersonList {
        readonly attribute string listname;
        void addPerson(in Person p) ;
        void getPerson(in string name, out Person p);
        long number( );
};
```

5.3 Chamada de Procedimento Remoto (RPC)

Semânticas de Chamada

Várias possibilidades de garantia de entrega:

- Retentativa de mensagem de requisição: para retransmitir a mensagem de requisição até que uma resposta seja recebida ou que se presuma que o servidor falhou.
- Filtragem de duplicatas: quando são usadas retransmissões para eliminar requisições duplicadas no servidor.
- Retransmissão de resultados: para manter um histórico das mensagens de respostas a fim de permitir que os resultados perdidos sejam retransmitidos sem uma nova execução das operações no servidor.

5.3 Chamada de Procedimento Remoto (RPC)

Semânticas de Chamada

Várias possibilidades de garantia de entrega:

- Retentativa de mensagem de requisição
- Filtragem de duplicatas
- Retransmissão de resultados

Combinações dessas escolhas levam a uma variedade de semânticas possíveis para a confiabilidade das invocações remotas.

5.3 Chamada de Procedimento Remoto (RPC)

Semânticas de Chamada

- Semântica talvez (maybe, best-effort)
- Semântica pelo menos uma vez (at least once)
- Semântica no máximo uma vez (at most once)

5.3 Chamada de Procedimento Remoto (RPC)

Semântica talvez (maybe, best-effort)

- O método remoto pode ser executado uma vez ou nenhuma.
- Não se utiliza nenhum tipo de tolerância a falhas
- Sujeito a falhas de omissão ou falhas do servidor (crash)
  - Mensagem de requisição foi perdida
  - Mensagem de resposta perdida
  - Falha antes da execução do método remoto

5.3 Chamada de Procedimento Remoto (RPC)

Semântica pelo menos uma vez (at least once)

- Recebe um resultado ou uma exceção
- Realiza retransmissão de mensagens
- Sujeito a falhas arbitrárias ou falhas do servidor *(crash)*
  - Mensagem repetidas podem causar respostas inconsistentes

5.3 Chamada de Procedimento Remoto (RPC)

Semântica no máximo uma vez (at most once)

- Recebe um resultado ou uma exceção
- Realiza retransmissão de mensagens e filtragem de mensagens duplicadas
- Semântica utlizada por Java RMI e CORBA

5.3 Chamada de Procedimento Remoto (RPC)

Semânticas de Chamada

Medidas de tolerância a falhas			*Semântica de chamada*
Reenvio da mensagem de requisição	*Filtragem de duplicatas*	*Reexecução de procedimento ou retransmissão da resposta*	
Não	Não aplicável	Não aplicável	*Talvez*
Sim	Não	Executa o procedimento novamente	*Pelo menos uma vez*
Sim	Sim	Retransmite a resposta	*No máximo uma vez*

5.3 Chamada de Procedimento Remoto (RPC)

Transparência

- RPC tenta oferecer pelo menos transparência de localização e de acesso, ocultando o local físico do procedimento (potencialmente remoto) e também acessando procedimentos locais e remotos da mesma maneira.
  - falhas parciais
  - latência
- As chamadas remotas devem se tornar transparentes: mesma sintaxe de uma chamada local.

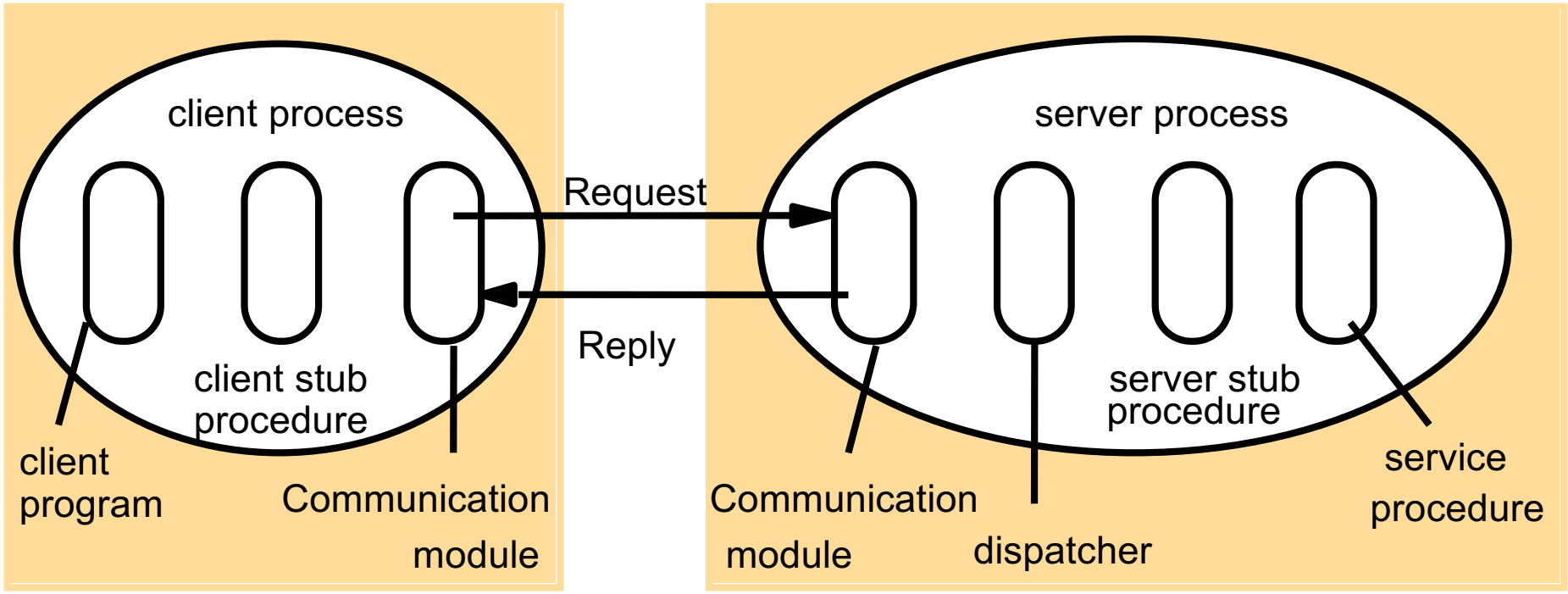
5.3 Chamada de Procedimento Remoto (RPC)

Implementação de RPC – Componentes de Software

- O cliente inclui um procedimento stub para cada procedimento da interface de serviço.
- O stub se comporta como um procedimento local para o cliente, mas empacota o identificador e os argumentos em uma mensagem de requisição, enviada ao servidor.
- O servidor desempacota os resultados contém um despachante junto a um procedimento stub de servidor e um procedimento de serviço para cada procedimento na interface de serviço.

5.3 Chamada de Procedimento Remoto (RPC)

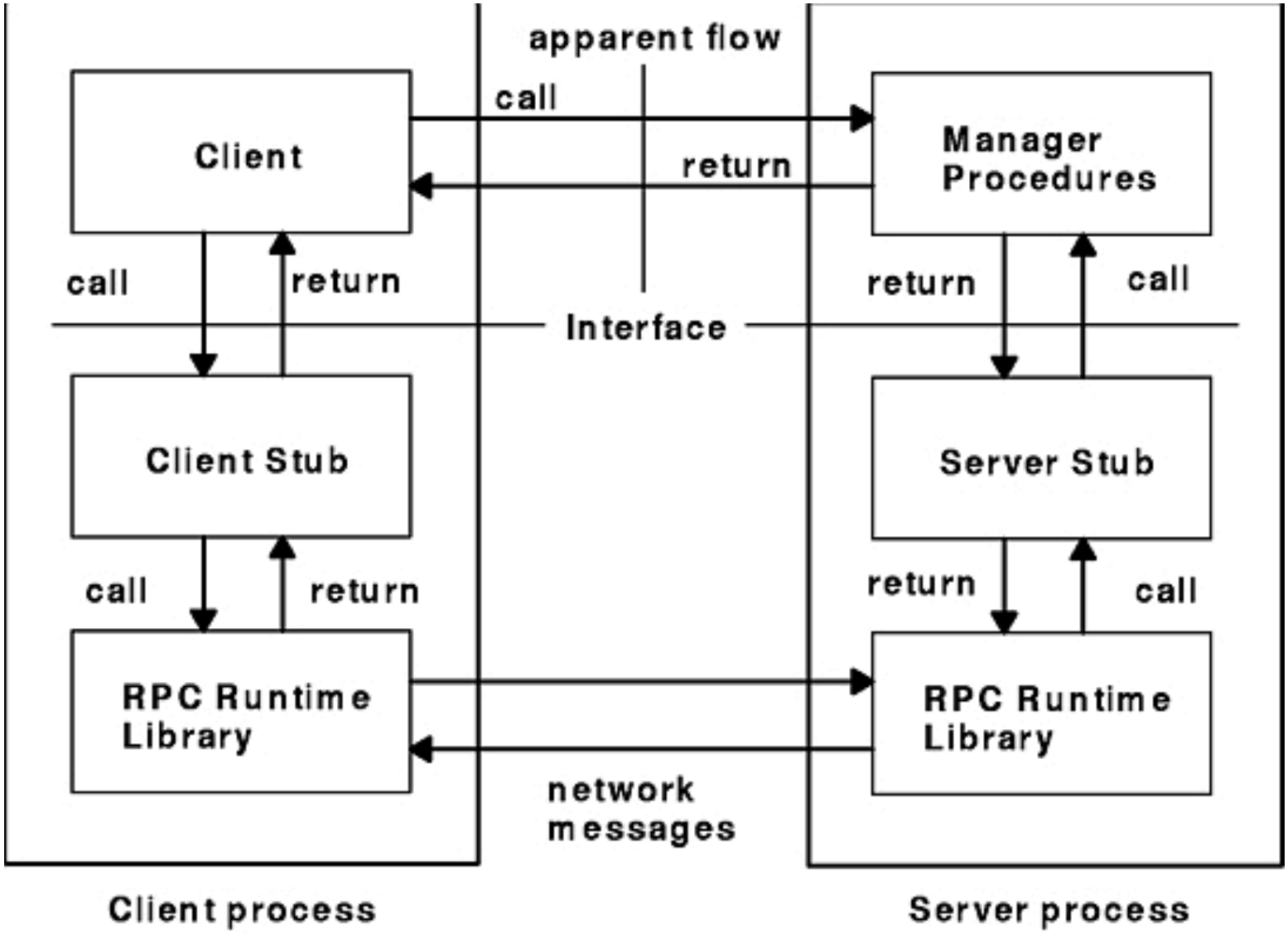
Implementação de RPC – Componentes de Software



Função dos procedimentos stub no cliente e no servidor RPC

5.3 Chamada de Procedimento Remoto (RPC)

Implementação de RPC – Componentes de Software



Remote Procedure Call Flow

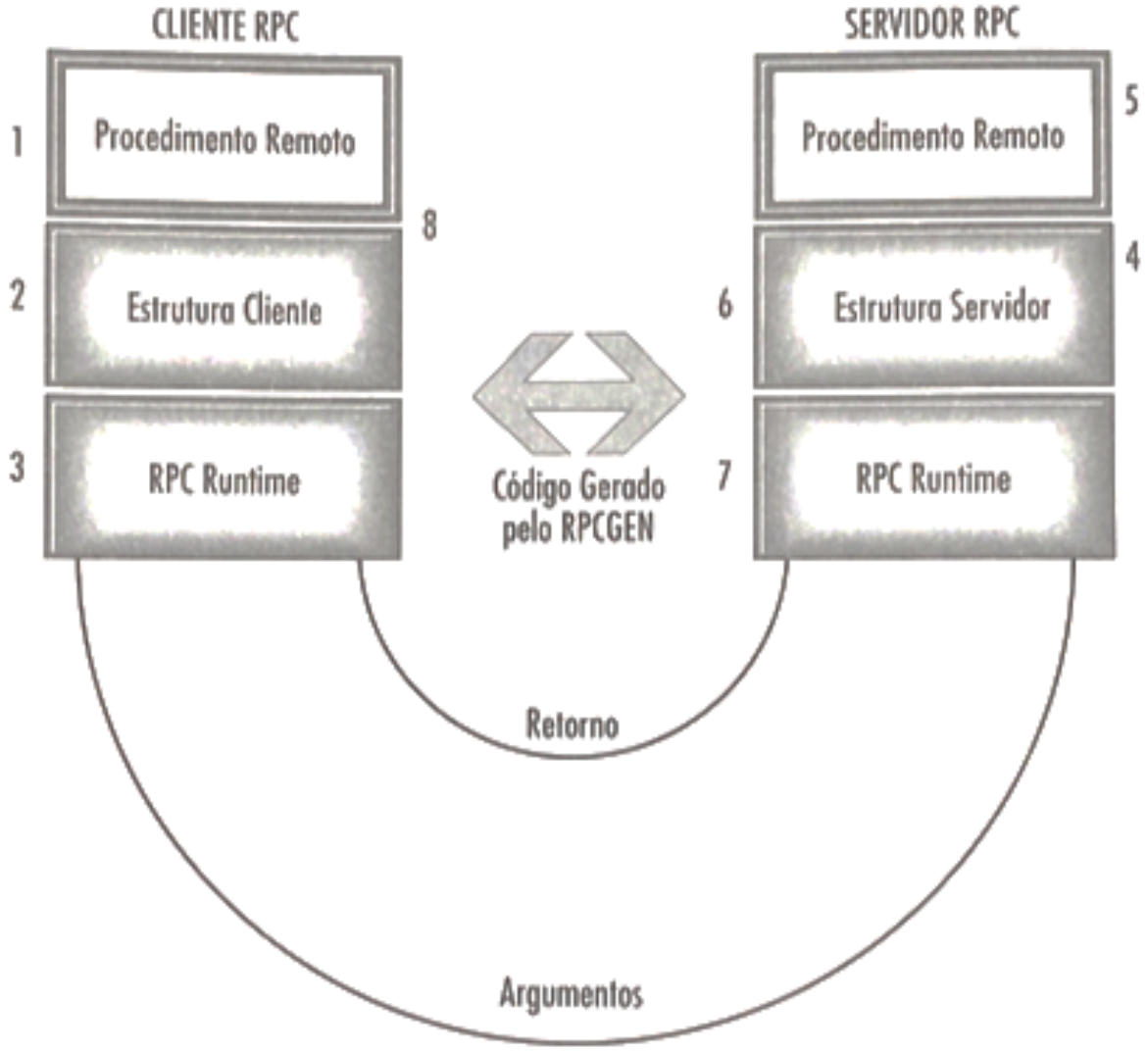
5.3 Chamada de Procedimento Remoto (RPC)

Sun RPC

- Um RPC utilizado para comunicação cliente-servidor no sistema de arquivos de rede NFS
- Pode utilizar UDP ou TCP
- Possui uma linguagem de definição interface chamada XDR. Mais simples em comparação com CORBA IDL.
- Possui um serviço para a descoberta de recursos chamado *port mapper*
- Possui recursos de autenticação

5.3 Chamada de Procedimento Remoto (RPC)

Sun RPC



5.3 Chamada de Procedimento Remoto (RPC)

Sun RPC

Interface Sun XDR – RFC 1832

Um protocolo deve acompanhar a seguinte especificação básica:

```
program NOMEDOPROGRAMA {
    version NOMEDAVERSAO {
       TIPO NOMEDOPROCEDIMENTO(TIPO ARGUMENTO) = X;
    } = Y;
} = 123456789;
```

5.3 Chamada de Procedimento Remoto (RPC)

Sun RPC

Interface Sun XDR – RFC 1832

```
program PROGRAMAPING {
    /*
    *  ÚLTIMA VERSÃO DO PROGRAMA EM TESTE
    */
    version PING_ULTIMAVERSAO {
       void PING_PROCEDIMENTO_NULL(void) = 0;
       int PING_PROCEDIMENTO_BACK(int NUM) = 1;
    } = 2;
    /*
    *  VERSÃO ORIGINAL EM PRODUÇÃO
    */
    version PING_VERSAOORIGINAL {
       void PING_PROCEDIMENTO_NULL(void) = 0;
    } = 1;
} = 200000001;
```

5.3 Chamada de Procedimento Remoto (RPC)

Sun RPC

Interface Sun XDR para arquivos

```
const MAX = 1000;
typedef int FileIdentifier;
typedef int FilePointer;
typedef int Length;
struct Data {
        int length;
        char buffer[MAX];
};
struct writeargs {
        FileIdentifier f;
        FilePointer position;
        Data data;
};
```

```
struct readargs {
        FileIdentifier f;
        FilePointer position;
        Length length;
};

program FILEREADWRITE {
   version VERSION {
        void WRITE(writeargs)=1;        1
        Data READ(readargs)=2;          2
   }=2;
} = 9999;
```

5.3 Chamada de Procedimento Remoto (RPC)

Sun RPC

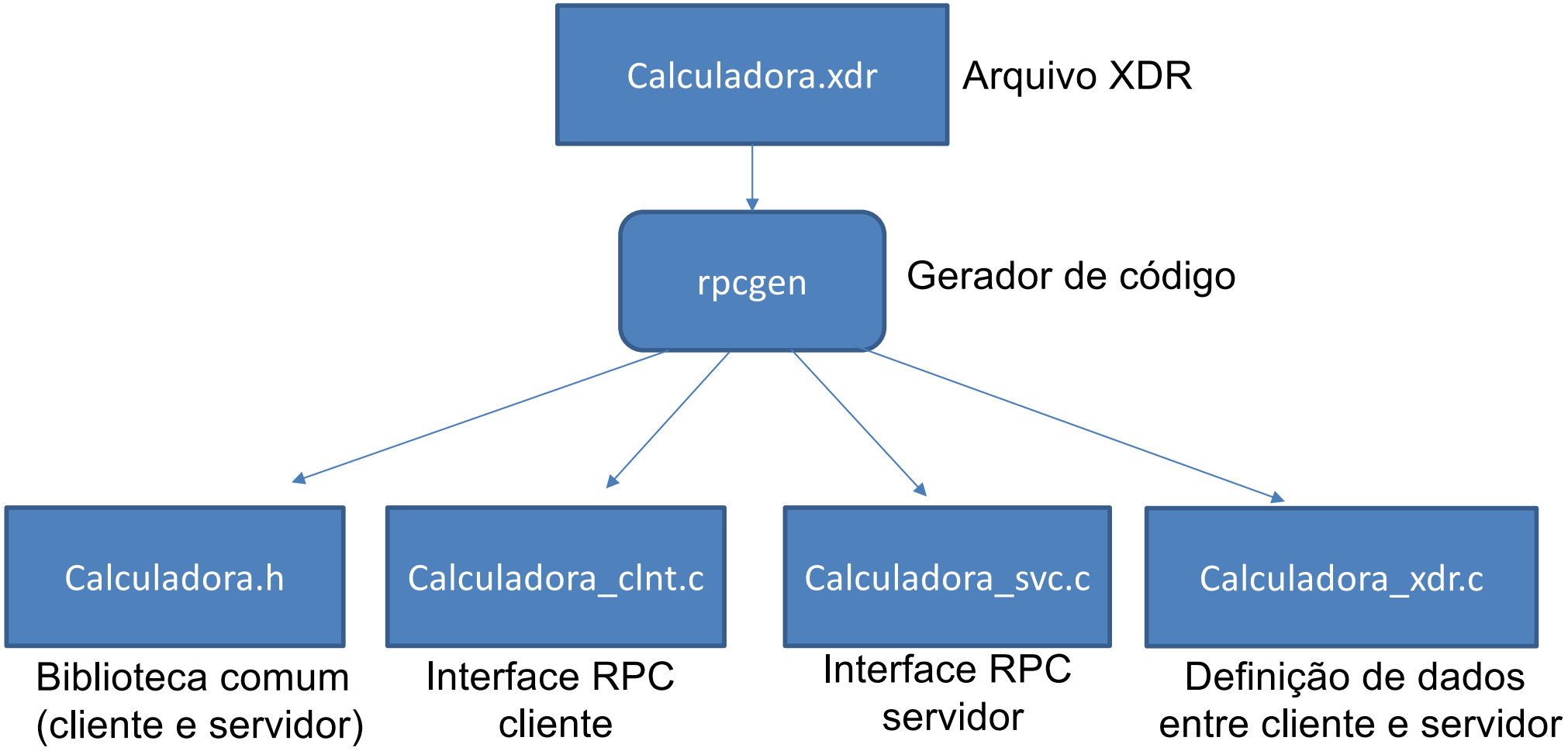
Interface Sun XDR para Calculadora

```
#define VERSAO 1
struct operando {
        int x;
        int y;
};

program CALCULADORA {
   version CALCULADORA_VERSAO {
     int SOMAR(operando) = 1;
     int SUBTRAIR(operando) = 2;
   } = VERSAO;
} = 300000001;
```

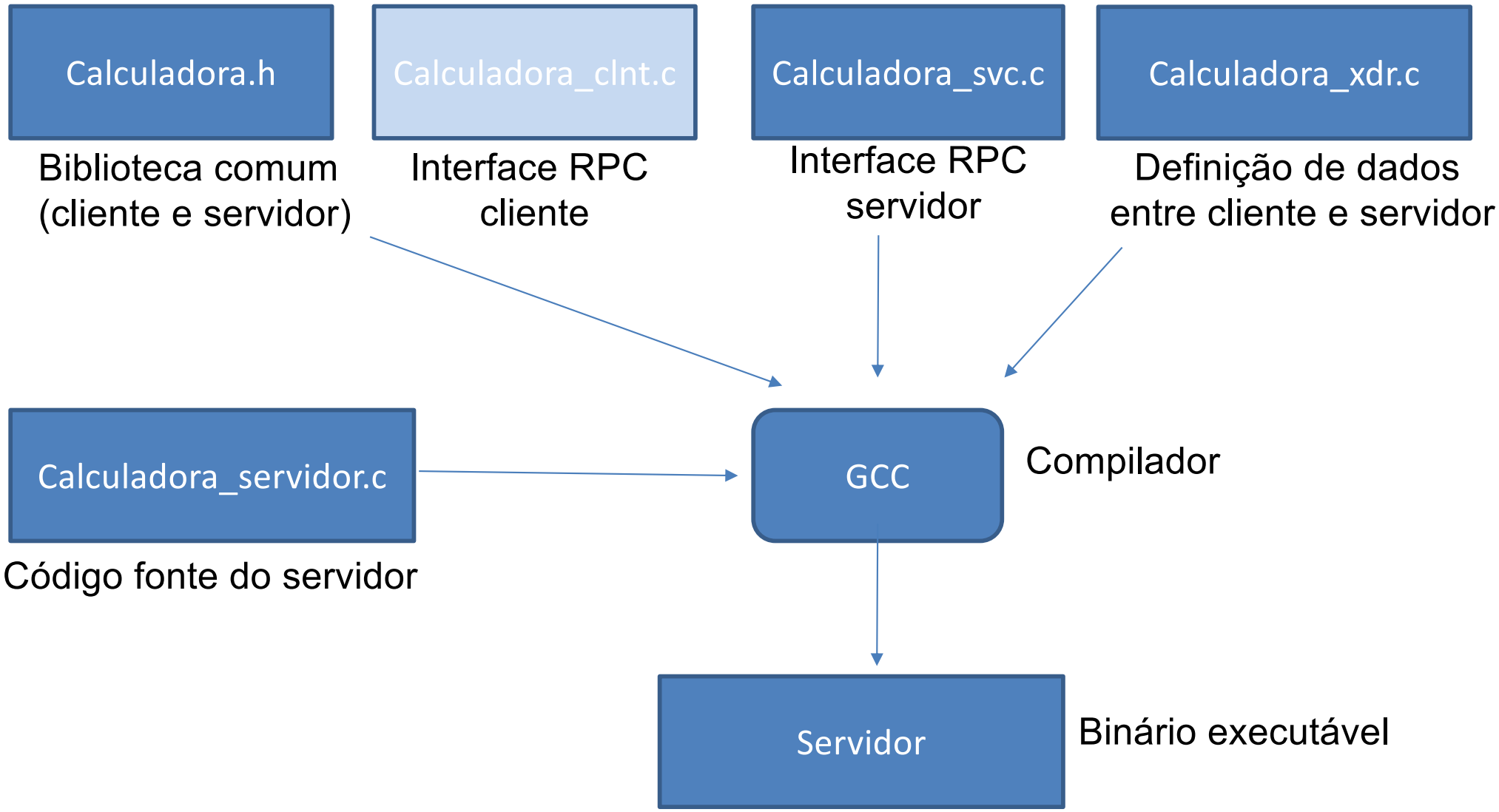
5.3 Chamada de Procedimento Remoto (RPC)

Sun RPC



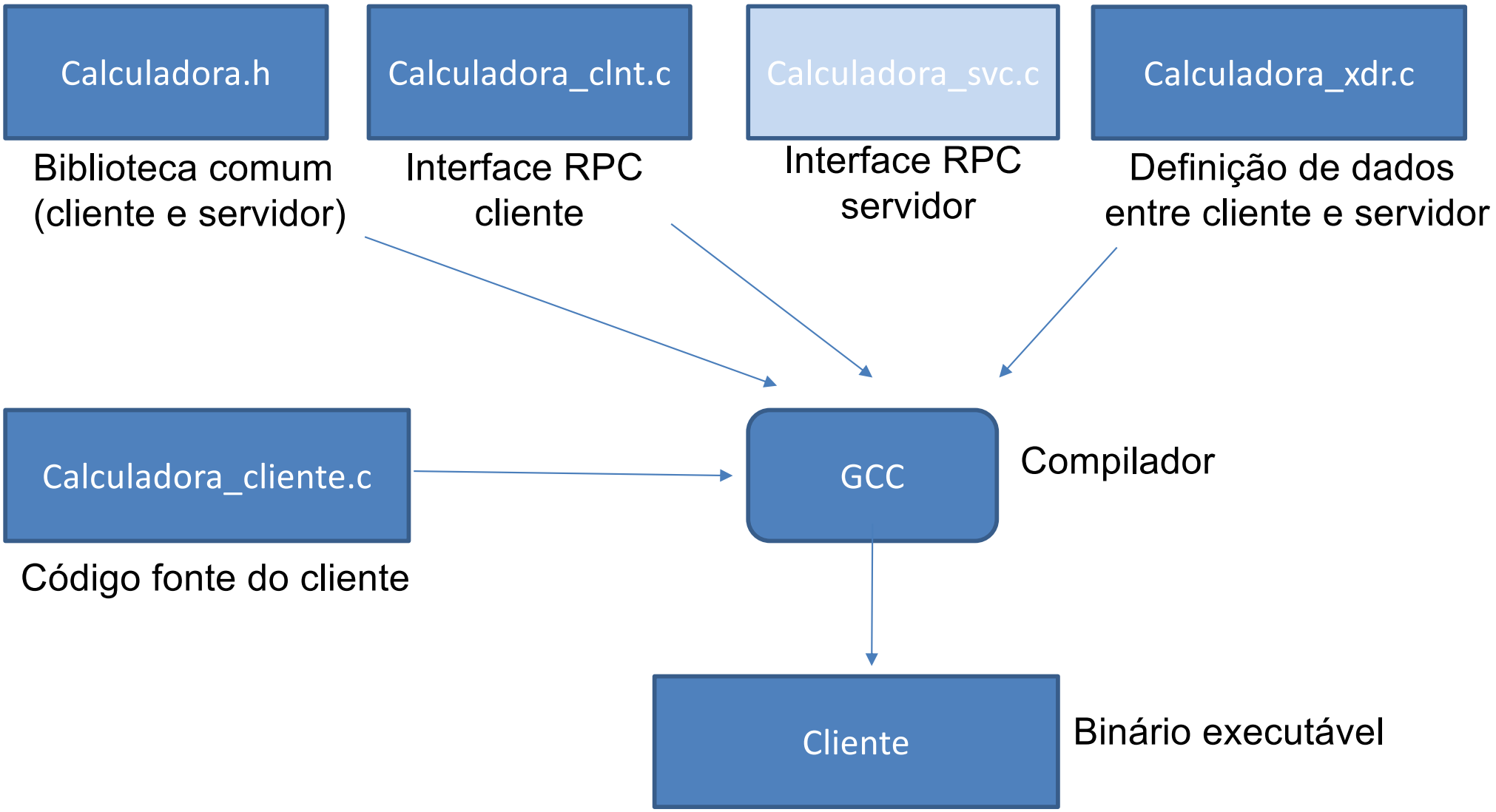
5.3 Chamada de Procedimento Remoto (RPC)

Sun RPC



5.3 Chamada de Procedimento Remoto (RPC)

Sun RPC



5.4 Invocação a Método Remoto (RMI)

Introdução

- A RMI (*Remote Method Invocation* – invocação a método remoto) está intimamente relacionada à RPC, mas é estendida para o mundo dos objetos distribuídos.
- Na RMI, um objeto chamador pode invocar um método em um objeto potencialmente remoto.
- Assim como na RPC, geralmente os detalhes subjacentes ficam ocultos para o usuário.

5.4 Invocação a Método Remoto (RMI)

RMI X RPC - Semelhanças

- Ambas suportam programação com interfaces.
- Normalmente, ambas são construídas sobre protocolos de requisição-resposta.
- Ambas fornecem um nível de transparência semelhante.
- As chamadas locais e remotas empregam a mesma sintaxe.

5.4 Invocação a Método Remoto (RMI)

RMI X RPC - Diferenças

- Em um sistema RMI, todas as chamadas tem referências exclusivas (sejam locais ou remotas).
- No RMI, as referências podem ser passadas como parâmetros, fornecendo, assim, uma semântica de passagem de parâmetros mais rica que no RPC.
- RMI permite passar parâmetros não somente por valor, como os parâmetros de entrada e saída, mas também por referência de objeto.

5.4 Invocação a Método Remoto (RMI)

5.4.1 Questões de Projeto

O Modelo de Objeto (convencional)

Um programa é um conjunto de objetos (compostos de dados e métodos) interagindo entre si, e eventualmente acessando diretamente variáveis uns dos outros. Características:

- **Referências de objetos**
- **Interfaces**
- **Ações**
- **Exceções**
- **Coleta de lixo (garbage collection)**

5.4 Invocação a Método Remoto (RMI)

5.4.1 Questões de Projeto

O Modelo de Objeto (convencional)

- **Referências de objetos**: usadas para que os objetos possam ser localmente acessados por outros objetos.

- **Interfaces**: fornecem a definição das assinaturas de um conjunto de métodos (os tipos de seus argumentos, valores de retorno e exceções), sem especificar sua implementação.

5.4 Invocação a Método Remoto (RMI)

5.4.1 Questões de Projeto

O Modelo de Objeto (convencional)

- **Ações**: a ação é iniciada por um objeto invocando um método em outro objeto (receptor).
- O receptor executa o método apropriado e depois retorna o controle para o objeto que fez a invocação, podendo fornecer um resultado.
- A ativação de um método pode ter três efeitos:
  1. O estado do receptor pode ser alterado.
  2. Um novo objeto pode ser instanciado.
  3. Outras invocações podem ocorrer nos métodos de outros objetos.
- uma ação é um encadeamento de invocações de métodos, cada uma com seu respectivo retorno.

5.4 Invocação a Método Remoto (RMI)

5.4.1 Questões de Projeto

O Modelo de Objeto (convencional)

- **Exceções**: proporcionam uma maneira clara de tratar com condições de erro, sem complicar o código.
- **Coleta de lixo:** é necessário fornecer uma maneira de liberar o espaço em memória ocupado pelos objetos quando eles não são mais necessários.
  - Quando uma linguagem não suporta coleta de lixo, o programador tem de se preocupar com a liberação do espaço alocado para os objetos. Isso pode ser uma fonte de erros significativa.

5.4 Invocação a Método Remoto (RMI)

5.4.1 Questões de Projeto

O Modelo de Objeto Distribuído

Extensões feitas no modelo de objeto para torná-lo aplicável aos objetos distribuídos. Cada processo contém um conjunto de objetos, alguns dos quais podem receber invocações locais e remotas, enquanto os outros objetos podem receber somente invocações locais. Características:

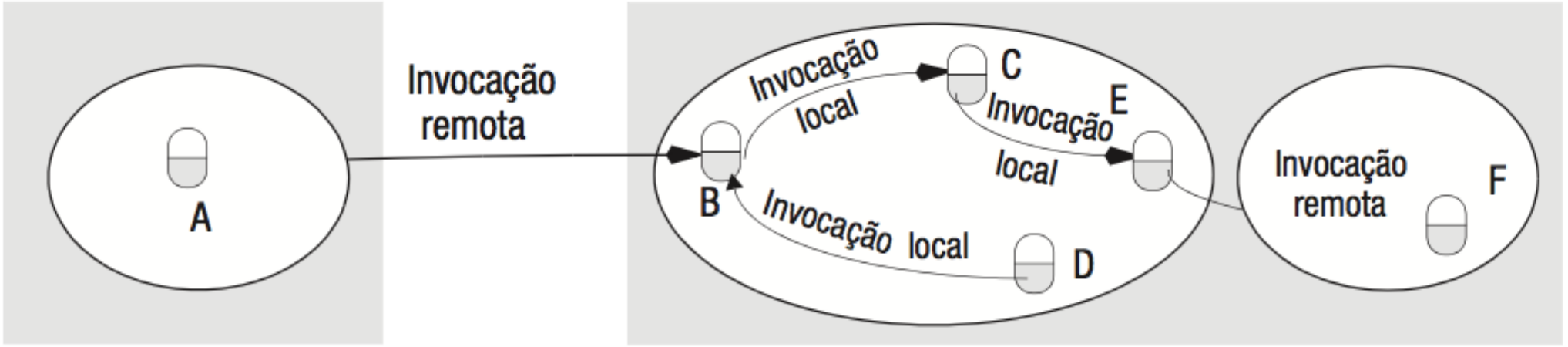
- **Referências de objeto remoto**
- **Interfaces remotas**
- **Ações em sistema de objeto distribuído**
- **Exceções**
- **Coleta de lixo em um sistema de objeto distribuído**

5.4 Invocação a Método Remoto (RMI)

5.4.1 Questões de Projeto

O Modelo de Objeto Distribuído

Invocação a métodos locais e remotos



- ***Objetos remotos:*** objetos que podem receber invocações remotas (B e F).
- ***Referência de objeto remoto****:* outros objetos podem invocar os métodos de um objeto remoto se tiverem acesso a sua *referência de objeto remoto*. Por exemplo, uma referência de objeto remoto de B deve estar disponível para A.
- ***Interface remota****:* todo objeto remoto tem uma interface remota especificando qual de seus métodos pode ser invocado de forma remota. Por exemplo, os objetos B e F devem ter interfaces remotas.

5.4 Invocação a Método Remoto (RMI)

5.4.1 Questões de Projeto

O Modelo de Objeto Distribuído

- **Referências de objeto remoto (I)**: estendem a noção de referência de objeto para permitir que um objeto que possa receber uma RMI tenha uma referência de objeto remoto.
- É um identificador que pode ser usado por todo um sistema distribuído para se referir a um objeto remoto único em particular, sendo, portanto, diferente da representação de referência de objeto local.

5.4 Invocação a Método Remoto (RMI)

5.4.1 Questões de Projeto

O Modelo de Objeto Distribuído

- **Referências de objeto remoto (II)**: As referências de objeto remoto são análogas, em função, às locais, pois:

  1. O objeto remoto que vai receber uma invocação a método remoto é especificado pelo invocador como uma referência de objeto remoto.

  2. As referências de objeto remoto podem ser passadas como argumentos e resultados de invocações a métodos remotos.

5.4 Invocação a Método Remoto (RMI)

5.4.1 Questões de Projeto

O Modelo de Objeto Distribuído

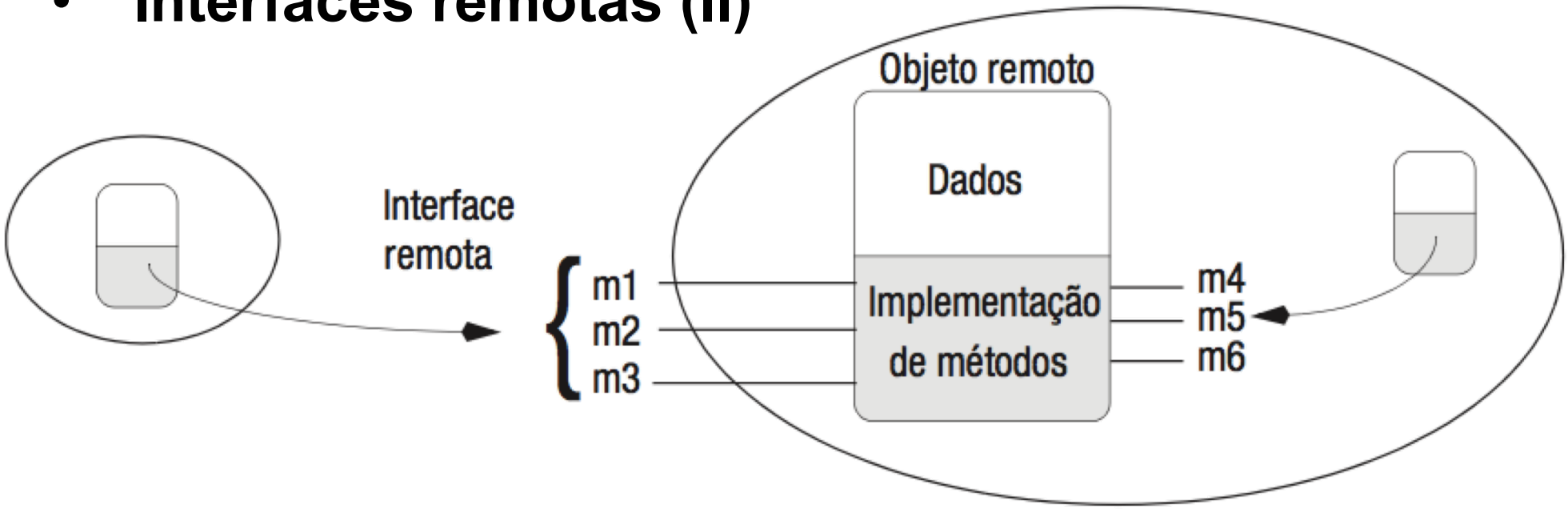
- **Interfaces remotas (I)**: a classe de um objeto remoto implementa os métodos de sua interface remota.
- Objetos em outros processos só podem invocar os métodos pertencentes à interface remota.
- Os objetos locais podem invocar os métodos da interface remota, assim como outros métodos locais implementados por um objeto remoto.

5.4 Invocação a Método Remoto (RMI)

5.4.1 Questões de Projeto

O Modelo de Objeto Distribuído

- **Interfaces remotas (II)**



Um objeto remoto e sua interface remota

- Os métodos m1, m2 e m3 podem ser acessados localmente e remotamente
- Os métodos m4, m5 e m6 podem ser acessados apenas localmente

5.4 Invocação a Método Remoto (RMI)

5.4.1 Questões de Projeto

O Modelo de Objeto Distribuído

- **Ações em um sistema de objeto distribuído (I)**: os objetos envolvidos em um encadeamento de invocações relacionadas podem estar localizados em diferentes processos ou em diferentes computadores.
- Quando uma invocação cruza o limite de um processo ou computador, a RMI é usada e a referência remota do objeto deve estar disponível para o invocador. Exemplo: o objeto A precisa conter uma referência de objeto remoto para o objeto B. Objeto A poderia obter uma referência remota para o objeto F a partir do objeto B.

5.4 Invocação a Método Remoto (RMI)

5.4.1 Questões de Projeto

O Modelo de Objeto Distribuído

- **Ações em um sistema de objeto distribuído (II)**
- instanciação remota de objetos: Os aplicativos distribuídos podem fornecer objetos remotos com métodos para instanciar objetos que podem ser acessados por uma RMI.

Exemplo: o objeto L contém um método para criar objetos remotos e as invocações remotas de C e K podem levar à instanciação dos objetos M e N, respectivamente.

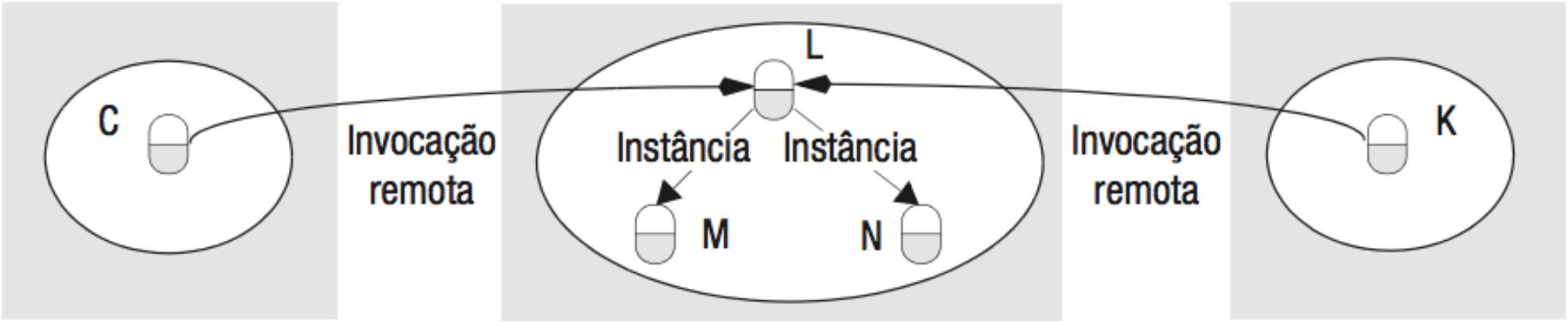
5.4 Invocação a Método Remoto (RMI)

5.4.1 Questões de Projeto

O Modelo de Objeto Distribuído

- **Ações em um sistema de objeto distribuído (III)**
- instanciação remota de objetos:

Exemplo: o objeto L contém um método para criar objetos remotos e as invocações remotas de C e K podem levar à instanciação dos objetos M e N, respectivamente.



Instanciação de objetos remotos

5.4 Invocação a Método Remoto (RMI)

5.4.1 Questões de Projeto

O Modelo de Objeto Distribuído

- **Coleta de lixo em um sistema de objeto distribuído:** se a linguagem suporta coleta de lixo, então qualquer sistema RMI associado deve permitir a coleta de lixo de objetos remotos.
- A coleta de lixo distribuída geralmente é obtida pela cooperação entre o coletor de lixo local existente e um módulo adicionado que realiza uma forma de coleta de lixo distribuída, normalmente baseada em contagem de referência.
- Se a coleta de lixo não estiver disponível, os objetos remotos que não são mais necessários deverão ser excluídos.

5.4 Invocação a Método Remoto (RMI)

5.4.1 Questões de Projeto

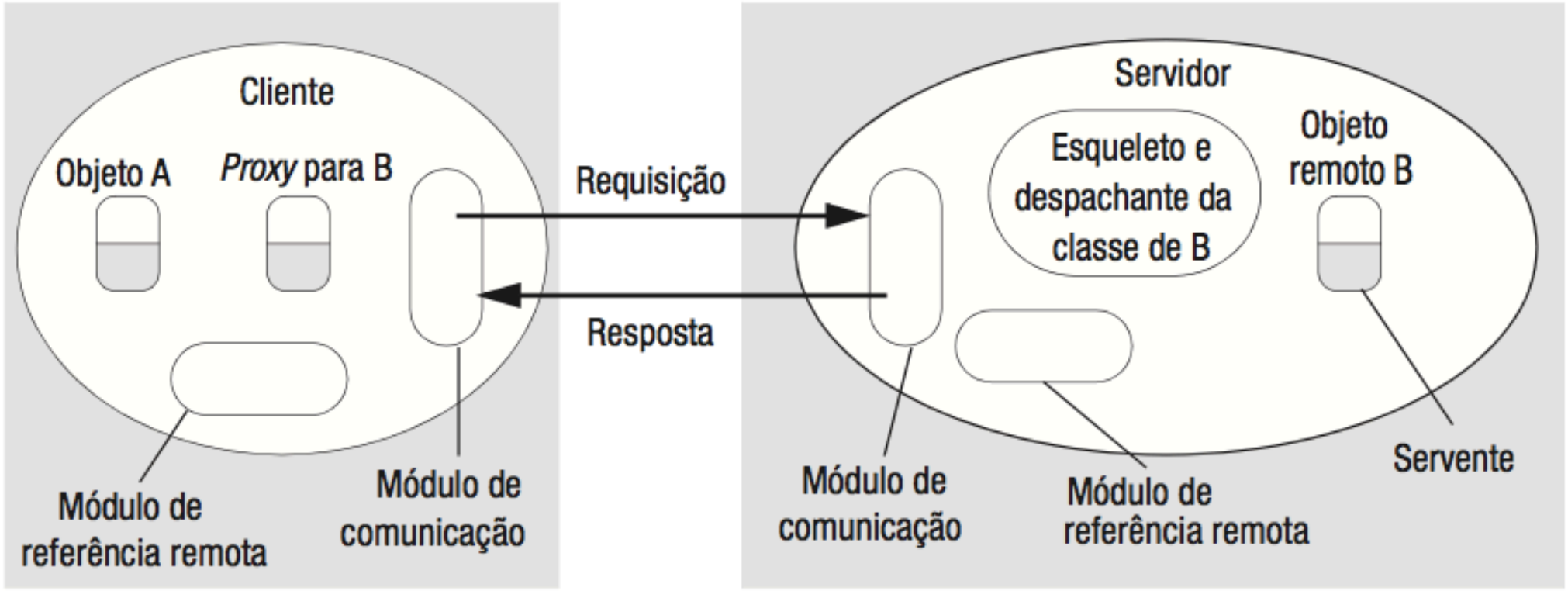
O Modelo de Objeto Distribuído

- **Exceções**: uma invocação remota pode falhar por motivos relacionados ao fato de o objeto invocado estar em um processo ou computador diferente do invocador.
- A invocação a método remoto deve ser capaz de lançar exceções, como expiração de tempo limite (timeout), causados pelo envio de mensagens, assim como as ocorridas durante a execução do método invocado, como tentativa de ler além do final de um arquivo ou de acessar um arquivo sem as permissões corretas.

5.4 Invocação a Método Remoto (RMI)

5.4.2 Implementação de RMI

Vários objetos e módulos separados estão envolvidos na realização de uma invocação a método remoto

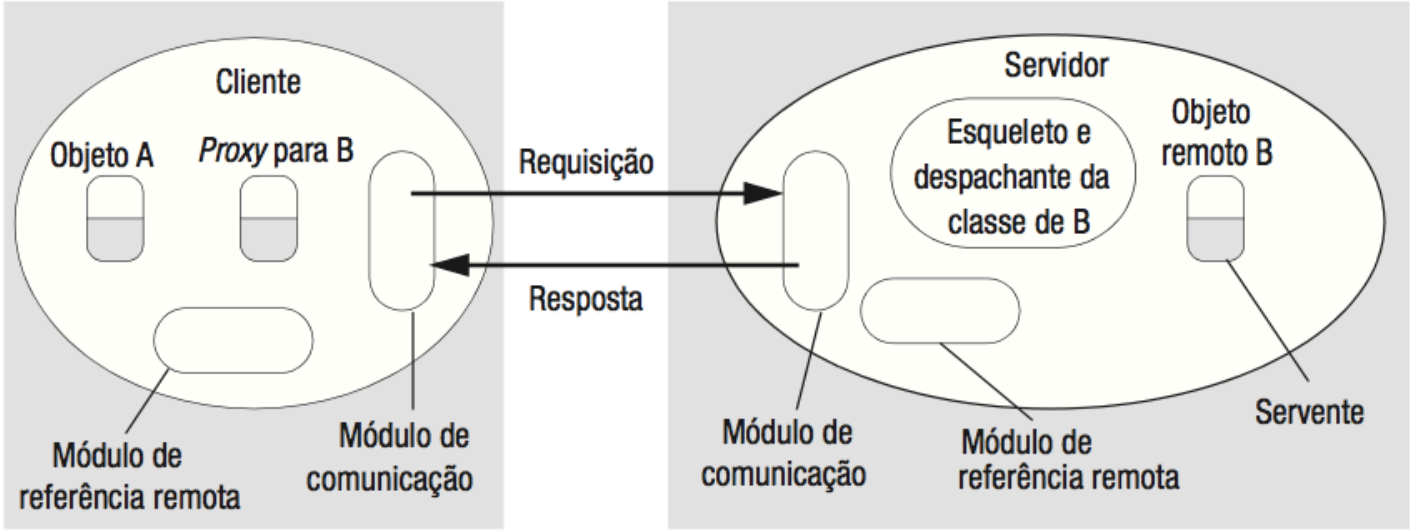


Um objeto em nível de aplicativo A invoca um método em um objeto remoto B, para o qual mantém uma referência de objeto remoto.

5.4 Invocação a Método Remoto (RMI)

5.4.2 Implementação de RMI

- Módulo de Comunicação
- Módulo de referência remota
- Serventes
- O software RMI
  - Proxy
  - Despachante
  - Esqueleto (*skeleton*)
  - Clientes e servidores



5.4 Invocação a Método Remoto (RMI)

5.4.2 Implementação de RMI

Módulo de comunicação (I)

- Cuida do protocolo de comunicação de mensagens entre cliente e servidor (dois módulos)
- Implementa o protocolo Requisição-Resposta (request-reply)
- Provê a identificação das requisições
- Semântica de chamadas (ex.: at-most-once)
- Retransmissão e eliminação de duplicatas
- No servidor é responsável por selecionar o despachante da classe do objeto
  - Passa a referência local do objeto obtido através do módulo de referência remota

5.4 Invocação a Método Remoto (RMI)

5.4.2 Implementação de RMI

Módulo de comunicação (II)

- Define os tipos de mensagens utilizados
- Utiliza os campos methodId, requestId e a referência de objeto remota do pacote de mensagem

messageType	*int (0=Request, 1=Reply)*
requestId	*int*
remoteReference	*RemoteRef*
OperationId	*int ou operação*
arguments	*// vetor de bytes*

5.4 Invocação a Método Remoto (RMI)

5.4.2 Implementação de RMI

Módulo de Referência Remota (I)

- Responsável por fazer a correspondência entre referências de objetos remotos e referências locais
- No lado do servidor:
    - tabela com as referências a objetos remotos que residem no processo local
        - ponteiro para o skeleton correspondente
- No lado cliente:
    - tabela com as referências a objetos remotos que residem em outros processos e que são utilizadas por clientes locais
        - ponteiro para o proxy local do objeto remoto

5.4 Invocação a Método Remoto (RMI)

5.4.2 Implementação de RMI

Módulo de Referência Remota (II)

- Responsável por:
  - Criar uma referência remota quando um objeto remoto é passado como parâmetro ou resultado pela primeira vez
  - No lado cliente, se a referência remota não estiver na tabela, deve-se adicioná-la e criar um proxy correspondente
- Usado durante o *marshalling* e *unmarshalling* (empacotamento e desempacotamento) de referências remotas

5.4 Invocação a Método Remoto (RMI)

5.4.2 Implementação de RMI

Serventes

- Instância de uma implementação de objeto
- Tratam as requisições remotas
- Hospedados em processos servidores

5.4 Invocação a Método Remoto (RMI)

5.4.2 Implementação de RMI

O Software RMI - Proxy

- Objeto local que representa o objeto remoto para o cliente. Um proxy para cada objeto remoto
- “Implementa” os métodos definidos na interface do objeto remoto
- Cada implementação de método no proxy:
    - Marshalling de requisições
    - Unmarshalling de respostas
    - Envio e recebimento através do módulo de comunicação
- Torna a localização e o acesso ao objeto remoto transparentes para o cliente

5.4 Invocação a Método Remoto (RMI)

5.4.2 Implementação de RMI

O Software RMI - Despachante

- um servidor tem um despachante e um esqueleto para cada classe que representa um objeto remoto (exemplo: o servidor tem um despachante e um esqueleto para a classe do objeto remoto B).
- O despachante recebe a mensagem de requisição do módulo de comunicação e utiliza o OperationId para selecionar o método apropriado no esqueleto, repassando a mensagem de requisição.
- O despachante e o proxy usam o mesmo OperationId para os métodos da interface remota.

5.4 Invocação a Método Remoto (RMI)

5.4.2 Implementação de RMI

O Software RMI - Esqueleto

- um servidor tem um despachante e um esqueleto para cada classe que representa um objeto remoto (exemplo: o servidor tem um despachante e um esqueleto para a classe do objeto remoto B).
- O despachante recebe a mensagem de requisição do módulo de comunicação e utiliza o OperationId para selecionar o método apropriado no esqueleto, repassando a mensagem de requisição.
- O despachante e o proxy usam o mesmo OperationId para os métodos da interface remota.

5.4 Invocação a Método Remoto (RMI)

5.4.2 Implementação de RMI

O Software RMI – Clientes e Servidores

- O programa servidor contém as classes para os despachantes e esqueletos, junto às implementações das classes de todos os serventes que suporta.
- O programa servidor contém uma seção de inicialização responsável por criar e inicializar pelo menos um dos serventes que fazem parte do servidor. Mais serventes podem ser criados em resposta às requisições dos clientes.
- O programa cliente conterá as classes dos proxies de todos os objetos remotos que ativará. Ele pode usar um vinculador para pesquisar referências de objeto remoto.

5.5 Estudo de Caso: RMI Java

RMI Java

- Fornecer suporte a objetos distribuídos na linguagem Java
- O cliente sabe quando faz uma requisição a um objeto remoto:
    - Deve lidar com RemoteExceptions
- Objeto remoto sabe que seus métodos podem ser
- acessados remotamente:
    - Deve implementar a interface Remote
- Na definição de uma interface remota os métodos devem lançar a exceção RemoteException

5.5 Estudo de Caso: RMI Java

RMI Java

- Tanto objetos comuns quanto objetos remotos podem ser passados como parâmetros ou resultados
    - Objetos remotos implementam a interface Remote
- Objetos comuns (não remotos) são passados por valor e o código de suas classes podem baixados entre diferentes processos
- Objetos remotos ao serem definidos como parâmetros ou resultados são substituídos por suas referências remota

5.5 Estudo de Caso: RMI Java

RMI Java – exemplo - quadro compartilhado

- Trata-se de um programa distribuído que permite a um grupo de usuários compartilhar uma vista comum de uma superfície de desenho contendo objetos gráficos, como retângulos, linhas e círculos, cada um dos quais desenhado por um dos usuários.
- O servidor mantém o estado corrente de um desenho, fornecendo uma operação para os clientes informarem-no sobre a figura mais recente que um de seus usuários desenhou e mantendo um registro de todas as figuras que tiver recebido.
- O servidor também fornece operações que permitem aos clientes recuperarem as figuras mais recentes desenhadas por outros usuários, e outras funções.

5.5 Estudo de Caso: RMI Java

RMI Java – exemplo - quadro compartilhado

Interfaces remotas Shape e ShapeList

```
import java.rmi.*;
import java.util.Vector;
public interface Shape extends Remote {
        int getVersion( ) throws RemoteException;
        GraphicalObject getAllState( ) throws RemoteException;                1
}

public interface ShapeList extends Remote {
        Shape newShape(GraphicalObject g) throws RemoteException;      2
        Vector allShapes( ) throws RemoteException;
        int getVersion( ) throws RemoteException;
}
```

5.5 Estudo de Caso: RMI Java

RMI Java – exemplo - quadro compartilhado

- Estas interfaces são utilizadas para criar aplicação distribuída chamada shared whiteboard
- Permitir o compartilhamento de objetos gráficos tais como retângulos, linhas e círculos fornecidos pelos usuários (clientes)
- O servidor mantém estes objetos gráficos que podem ser acessados por todos os usuários
  - Permite verificar a versão atual para verificar se novos objetos foram adicionados

5.5 Estudo de Caso: RMI Java

RMI Java - RMIRegistry

- O binder do Java RMI
  - Utilizado para publicar objetos remotos do Java RMI
- Qualquer processo que hospeda objetos remotos deve possuir uma instância do RMIregistry
- Mantém uma tabela com o nome do objeto remoto e sua referência
- Seus métodos são utilizados através da classe *Naming*

5.5 Estudo de Caso: RMI Java

RMI Java - RMIRegistry

- O binder do Java RMI
  - Utilizado para publicar objetos remotos do Java RMI
- Qualquer processo que hospeda objetos remotos deve possuir uma instância do RMIregistry
- Mantém uma tabela com o nome do objeto remoto e sua referência
- Seus métodos são utilizados através da classe *Naming*

5.5 Estudo de Caso: RMI Java

RMI Java - A classe Naming de RMIregistry Java

void rebind (String name, Remote obj)
Este método é usado por um servidor para registrar o identificador de um objeto remoto pelo nome, conforme mostrado na Figura 5.18, linha 3.

void bind (String name, Remote obj)
Este método pode ser usado como alternativa por um servidor para registrar um objeto remoto pelo nome, mas se o nome já estiver vinculado a uma referência de objeto remoto, será disparada uma exceção.

void unbind (String name, Remote obj)
Este método remove vínculos.

Remote lookup(String name)
Este método é usado pelos clientes para procurar um objeto remoto pelo nome, conforme mostrado na Figura 5.20, linha 1. Retorna uma referência de objeto remoto.

String [ ] list( )
Este método retorna um vetor de objetos String contendo os nomes vinculados no registro.

5.5 Estudo de Caso: RMI Java

RMI Java – Implementação do Servidor

```
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;
public class ShapeListServer{
    public static void main(String args[ ]){
        System.setSecurityManager(new RMISecurityManager( ));
        try{
            ShapeList aShapeList = new ShapeListServant( );                          1
            ShapeList stub =                                                         2
                    (ShapeList) UnicastRemoteObject.exportObject(aShapeList,0);  3
             Naming.rebind("//bruno.ShapeList", stub );                              4
             System.out.println("ShapeList server ready");
        }catch(Exception e) {
             System.out.println("ShapeList server main " + e.getMessage( ));}
    }
}
```

5.5 Estudo de Caso: RMI Java

RMI Java – Implementação do Servidor

- Na linha 1, o servidor cria uma instância de *ShapeListServant*.
- As linhas 2 e 3 utilizam o método *exportObject* (definido em *UnicastRemoteObject*) para tornar esse objeto disponível para o *runtime* da RMI, tornando-o, com isso, disponível para receber invocações.
- A linha 4 vincula o objeto remoto a um nome no RMIregistry. Note que o valor vinculado ao nome é uma referência de objeto remoto e seu tipo é o tipo de sua interface remota – *ShapeList*.

5.5 Estudo de Caso: RMI Java

RMI Java – Implementação do Servente

```
import java.util.Vector;
public class ShapeListServant implements ShapeList {
    private Vector theList; // contém a lista de elementos Shapes
    private int version;
    public ShapeListServant( ){...}
    public Shape newShape(GraphicalObject g) {          1
        version++;
        Shape s = new ShapeServant( g, version);        2
        theList.addElement(s);
        return s;
    }
    public Vector allShapes( ){...}
    public int getVersion( ) { ... } }
```

A classe *ShapeListServant* Java que implementa a interface *ShapeList*.

5.5 Estudo de Caso: RMI Java

RMI Java – Implementação do Servente

1. O método de *newShape* que poderia ser chamado de método de fábrica, pois ele permite que o cliente solicite a criação de um servente. Ele usa o construtor de *ShapeServant*, o qual cria um novo servente contendo o objeto *GraphicalObject* e o número de versão passados como argumentos. O tipo do valor de retorno de *newShape* é *Shape* – a interface implementada pelo novo servente.
2. Antes de retornar, o método *newShape* adiciona a nova figura em seu vetor, que contém a lista de figuras.

5.5 Estudo de Caso: RMI Java

RMI Java – Implementação do Cliente

```
import java.rmi.*;
import java.rmi.server.*;
import java.util.Vector;
public class ShapeListClient{
    public static void main(String args[ ]){
        System.setSecurityManager(new RMISecurityManager( ));
        ShapeList aShapeList = null;
        try{
             aShapeList = (ShapeList) Naming.lookup("//bruno.ShapeList");     1
            Vector sList = aShapeList.allShapes( );                           2
        } catch(RemoteException e) {System.out.println(e.getMessage( ));
        }catch(Exception e) {System.out.println("Client: " + e.getMessage( ));}
    }
}
```

Cliente Java para *ShapeList*.

5.5 Estudo de Caso: RMI Java

RMI Java – Implementação do Cliente

1. O cliente configura um gerenciador de segurança e, depois, pesquisa uma referência de objeto remoto usando a operação *lookup* do RMIregistry.
2. O cliente invoca o método *allShapes* no objeto remoto e recebe um vetor de referências de objeto remoto para todas as figuras correntemente armazenadas no servidor.

Se o cliente implementasse uma tela para o quadro branco, ele usaria o método *getAllState* do servidor na interface *Shape* para recuperar cada um dos objetos gráficos do vetor e os exibir na janela.

5.5 Estudo de Caso: RMI Java

RMI Java – Classes que suportam o RMI Java