Comunicação entre Processos – parte 2

Raquel Mini - raquelmini@pucminas.br

Representação Externa de Dados

As informações armazenadas nos programas em execução são representadas como estruturas de dados

As informações presentes nas mensagens são sequências puras de *bytes*

Nem todos os computadores armazenam tipos de dados primitivos na mesma ordem

A representação interna de números em ponto flutuante também difere entre as arquiteturas e processadores

Representação Externa de Dados

Existem duas variantes para a ordenação de inteiros:

- Ordem *big-endian*: o byte mais significativo aparece na primeira posição
- Ordem *little-endian*: o byte mais significativo aparece por último

Código usado para representar caracteres

- ASCII com um byte por caractere
- Unicode permite a representação de textos em muitos idiomas diferentes e usa dois bytes por caractere

Representação Externa de Dados

As estruturas de dados devem ser convertidas em uma sequência de bytes antes da transmissão e reconstruídas na sua chegada

- Os dados são convertidos para uma representação externa de dados
- Empacotamento (*marshalling*)
  - Procedimento de pegar um conjunto de itens de dados e montá-los em uma forma conveniente para transmissão em uma mensagem
- Desempacotamento (*unmarshalling*)
  - Procedimento inverso de desmontá-los na chegada para produzir um conjunto de itens de dados equivalente no destino

Empacotamento

Preparação de um conjunto de dados para transmissão em mensagens

```
nome="José", conta=152, saldo=25.2

msg:= new Msg;
msg.addField.asString (nome);
msg.addField.asInteger (conta);
msg.addField.asFloat (saldo);
msg.send (Q);
```

Desempacotamento

Recuperação de um conjunto de dados enviados em mensagens

```
msg:= new Msg;
msg.receive (P);
nome= msg.getField.asString ();
conta= msg.getField.asInteger ();
saldo= msg.getField.asFloat ();
```

Empacotamento e Desempacotamento

Manualmente: tedioso

Automaticamente a partir de uma especificação da estrutura de dados, requer:

- Linguagem para especificação
- Compilador para esta linguagem

Representação Externa de Dados e Empacotamento

Estratégias para representação externa de dados e empacotamento:

- Representação Comum de Dados (CDR) do CORBA
- Serialização de objetos Java
- XML (*Extensible Markup Language*) que define um formato textual para representar dados estruturados

CDR do CORBA

Pode representar todos os tipos de dados que são usados como argumentos e valores de retorno em invocações a métodos remotos no CORBA

- 15 tipos primitivos
- Variedade de tipos construídos

Cada argumento ou resultado em uma invocação remota é representado por uma sequência de bytes na mensagem de invocação ou resultado

CDR do CORBA

Tipos construídos

- Os valores primitivos que compreendem cada tipo construído são adicionados a uma sequência de *bytes*, em uma ordem específica

Tipo	*Representação*
sequence	comprimento (*unsigned long*) seguido de seus elementos, em ordem
string	comprimento (*unsigned long*) seguido pelos caracteres que o compõem (um caractere pode ocupar mais de um *byte*)
array	elementos de vetor, fornecidos em ordem (nenhum comprimento especificado, pois é fixo)
struct	na ordem da declaração dos componentes
enumerated	*unsigned long* (os valores são especificados pela ordem declarada)
union	identificador de tipo seguido do membro selecionado

CDR do CORBA – Exemplo

```
struct Person{
        string name;
        string place;
        unsigned long year;
};
```

Mensagem no CDR do CORBA

Índice na sequência de bytes	← 4 bytes →	Observações sobre a representação
0–3	5	Comprimento do string
4–7	"Smit"	'Smith'
8–11	"h___"	
12–15	6	Comprimento do string
16–19	"Lond"	'London'
20–23	"on__"	
24–27	1984	unsigned long

Serialização de Objetos Java

No Java RMI, tanto objetos como valores de dados primitivos podem ser passados como argumentos e resultados de invocações de método

Serialização se refere à atividade de simplificar um objeto, ou um conjunto de objetos conectados, em uma forma sequencial conveniente para ser armazenada em disco ou transmitida em uma mensagem

Desserialização consiste em restaurar o estado de um objeto ou conjunto de objetos a partir de sua forma serializada

Serialização de Objetos Java – Exemplo

```
public class Person implements Serializable {
        private String name;
        private String place;
        private int year;
        public Person(String aName, String aPlace, int aYear) {
                name = aName;
                place = aPlace;
                year = aYear;
        }
        // seguido dos métodos para acessar as variáveis de instância
}
```

```
Person p = new Person("Smith", "London", 1984);
```

Indicação da forma serializada Java

Valores serializados				*Explicação*
Person	Número da versão de 8 bytes		h0	*Nome da classe, número da versão*
3	int year	java.lang.String name	java.lang.String place	*Número, tipo e nome das variáveis de instância*
1984	5 Smith	6 London	h1	*Valores das variáveis de instância*

Na realidade, a forma serializada inclui marcas adicionais de tipos; h0 e h1 são identificadores.

XML

XML é uma linguagem de marcação que foi definida pelo WWW Consortium (W3C) para uso da Web

Foi projetada para elaborar documentos estruturados para a Web

Os itens de dados XML são rotulados com *strings* de marcação (*tags*)

As *tags* são usadas para descrever a estrutura lógica dos dados e para associar pares atributo-valor às estruturas lógicas

XML

Permite que clientes se comuniquem com serviços Web e para definir as interfaces e outras propriedades desses mesmos serviços

É usada também no arquivamento e na recuperação de sistemas

Especificação de interfaces com usuário

Codificação de arquivos de configuração em sistemas operacionais

É extensível

- Usuários podem definir suas próprias *tags*

XML – Exemplo

```
Person p = new Person("Smith", "London", 1984);
```

Definição XML

```
<person id="123456789">
        <name>Smith</name>
        <place>London</place>
        <year>1984</year>
        <!-- a comment -->
</person >
```

XML – Exemplo

```
Person p = new Person("Smith", "London", 1984);
```

Um esquema XML

```
<xsd:scheme xmlns:xsd = URL das definições de esquema XML >
    <xsd:element name= "person" type ="personType"/>
        <xsd:complexType name="personType">
            <xsd:sequence>
                <xsd:element name = "name"  type="xs:string"/>
                <xsd:element name = "place"  type="xs:string"/>
                <xsd:element name = "year"  type="xs:positiveInteger"/>
            </xsd:sequence>
            <xsd:attribute name= "id"   type = "xs:positiveInteger"/>
        </xsd:complexType>
</xsd:schema>
```

Exercícios

4. Qual é a forma serializada utilizando a serialização Java para seguinte código?

```
class Couple implements Serializable{
private Person one;
private Person two;
public Couple(Person a, Person b) {
        one = a;
        two = b;
}
}
```

Exercícios

5. Por que dados binários não podem ser representados diretamente em XML? A XML pode usar a codificação base64? Explique o funcionamento desta codificação.

Comunicação entre Processos

Troca de mensagens é pouco natural

Utilização de abstrações de mais alto nível para "esconder" a troca de mensagens

Abstrações de mais alto nível:

- Protocolos de requisição-resposta: HTTP
- Chamada de procedimento remoto (RPC – *Remote Procedure Call*): XDR (*External Data Representation*) da Sun
- Invocação de método remoto (RMI – *Remote Method Invocation*): Java RMI

Comunicação entre Processos

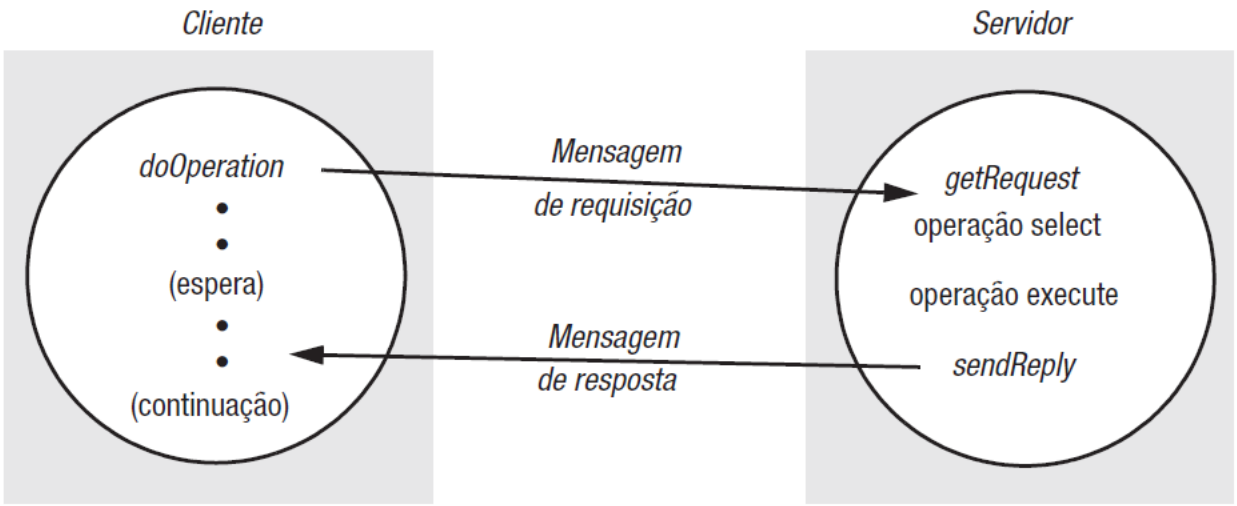
Abstrações de mais alto nível:

- **Protocolos de requisição-resposta**
- Chamada de procedimento remoto
- Invocação de método remoto

Protocolos de Requisição-resposta

Projetada para suportar funções de trocas de mensagens em interações cliente e servidor típicas

API Java para datagramas UDP



Protocolos de Requisição-resposta

Operações do protocolo de requisição-resposta

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

Envia uma mensagem de requisição para o servidor remoto e retorna 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.

Protocolos de Requisição-resposta

Estrutura da mensagem de requisição-resposta

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

Protocolos de Requisição-resposta

Se as três primitivas `doOperation`, `getRequest` e `sendReply` forem implementadas sobre o UDP, elas sofrerão falhas

- Utilização de tempo limite (*timeouts*)
- Descarte de mensagens de requisição duplicadas
- Mensagem de resposta perdidas
- Histórico das mensagens transmitidas

Protocolos de Requisição-resposta

A escolha do protocolo TCP simplifica a implementação de protocolos de requisição-resposta

- Comunicação confiável
- Controle de fluxo

Protocolos de Requisição-resposta

HTTP (*HyperText Transfer Protocol*) é um exemplo de protocolo de requisição-resposta usado pelos navegadores Web para fazer pedidos para servidores Web e receber suas respostas

- Especifica as mensagens envolvidas em uma troca de requisição-resposta, os métodos, os argumentos, os resultados e as regras para representá-los nas mensagens
- É implementado sobre o TCP
- Utiliza conexões persistentes
  - Permanecem abertas durante uma sequência de trocas de requisição-resposta entre cliente e servidor

HTTP

As requisições e respostas são empacotadas nas mensagens como *strings* de texto ASCII

- Recursos podem ser representados como sequência de bytes e podem ser compactados

Cada requisição especifica o nome de um método a ser aplicado em um recurso no servidor e o URL desse recurso

A resposta fornece o status da requisição

As mensagens de requisição-resposta podem conter dados de recurso, conteúdo de um formulário ou a saída de um programa executado no servidor Web

HTTP

Cliente HTTP: *browser*

Servidor HTTP: Servidor *web* (espera conexões na porta 80)

Um cliente HTTP abre uma conexão e envia uma **mensagem de requisição** ao servidor HTTP

O servidor HTTP retorna uma **mensagem de resposta,** geralmente contendo o recurso que foi requisitado

Formato das Mensagens

1. <linha inicial> diferente para msg de requisição e de resposta
2. Header 1: value
   Header 2: value
   Header 3: value

   Linhas de cabeçalho

   <linha em branco>
3. <corpo da mensagem>

Linha Inicial

Mensagem de requisição

```
<método> <caminho recurso requisitado> <versão do HTTP>
GET  /path/to/file/index.html  HTTP/1.1
```

Mensagem de resposta (“status line”)

<versão do HTTP> <código do status da resposta>

```
HTTP/1.1 200 OK
```

ou

```
HTTP/1.1 404 Not Found
```

HTTP – Métodos de Requisição

Method	Description
GET	Request to read a Web page
HEAD	Request to read a Web page's header
PUT	Request to store a Web page
POST	Append to a named resource (e.g., a Web page)
DELETE	Remove the Web page
TRACE	Echo the incoming request
CONNECT	Reserved for future use
OPTIONS	Query certain options

HTTP – Mensagens de Resposta

Code	Meaning	Examples
1xx	Information	100 = server agrees to handle client's request
2xx	Success	200 = request succeeded; 204 = no content present
3xx	Redirection	301 = page moved; 304 = cached page still valid
4xx	Client error	403 = forbidden page; 404 = page not found
5xx	Server error	500 = internal server error; 503 = try again later

Linhas de Cabeçalho

<header-name>: value <CRLF>

```
Header1:some-long-value-1a,
        some-long-value-1b,
        some-long-value-1c
```

HTTP 1.0 define 16 cabeçalhos

HTTP 1.1 define 46 cabeçalhos

HTTP – Cabeçalho de Mensagem

Header	Type	Contents
User-Agent	Request	Information about the browser and its platform
Accept	Request	The type of pages the client can handle
Accept-Charset	Request	The character sets that are acceptable to the client
Accept-Encoding	Request	The page encodings the client can handle
Accept-Language	Request	The natural languages the client can handle
Host	Request	The server's DNS name
Authorization	Request	A list of the client's credentials
Cookie	Request	Sends a previously set cookie back to the server
Date	Both	Date and time the message was sent
Upgrade	Both	The protocol the sender wants to switch to
Server	Response	Information about the server
Content-Encoding	Response	How the content is encoded (e.g., gzip)
Content-Language	Response	The natural language used in the page
Content-Length	Response	The page's length in bytes
Content-Type	Response	The page's MIME type
Last-Modified	Response	Time and date the page was last changed
Location	Response	A command to the client to send its request elsewhere
Accept-Ranges	Response	The server will accept byte range requests
Set-Cookie	Response	The server wants the client to save a cookie

Corpo da Mensagem

Mensagem de requisição: local onde os dados fornecidos pelo usuário ou arquivos para *upload* são enviados para o servidor

Mensagem de resposta: é o local onde o recurso requisitado se encontra

Quando uma msg HTTP possui um corpo, geralmente existem cabeçalhos para descreverem a mensagem

- `Content-Type: tipo MIME da página`
- `Content-Length: comprimento da página em bytes`

Métodos de Requisição

POST

- utilizado para enviar dados ao servidor para serem processados de alguma forma
- utiliza o corpo da mensagem e por isto utiliza cabeçalhos adicionais como `Content-Type` e `Content-Length`
- A URL requisitada não é um arquivo a ser recuperado, mas um programa para tratar os dados que estão sendo enviados
- A mensagem de resposta é normalmente uma saída de um programa e não um arquivo estático

Métodos de Requisição

O método GET também pode ser utilizado para enviar pequenas quantidades de dados ao servidor. Os dados são codificados na URL após o sinal de “?”.

HEAD: solicita ao servidor a resposta sem o corpo da mensagem. É útil para verificar características (ex. disponibilidade) de um recurso sem realmente baixá-lo

Cliente HTTP 1.1

HTTP 1.0: a conexão TCP era finalizada após cada requisição/resposta. Cada recurso a ser recuperado requisitava sua própria conexão.

Cliente HTTP 1.1

HTTP 1.1 e posteriores utilizam **Conexões Persistentes**

- O cliente abre a conexão e envia muitas requisições em série (*pipeling*) e recebe as respostas na mesma ordem que as requisições foram enviadas
- É possível enviar e receber várias requisições dentro uma mesma conexão TCP
- Vantagem: diminui o *overhead* causado pelo estabelecimento e finalização das conexões TCP

Comunicação entre Processos

Abstrações de mais alto nível:

- Protocolos de requisição-resposta
- **Chamada de procedimento remoto**
- Invocação de método remoto

Chamada de Procedimento Remoto

Tornar a programação de sistemas distribuídos semelhante à programação convencional

Obter transparência de distribuição de alto nível

Estender 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

Foi apresentado pela primeira vez por Birrel & Nelson, 1984

Chamada de Procedimento Remoto

Questões de projeto para RPC

- Programação com interfaces
- Semântica de chamada RPC
- Transparência

Programação com Interfaces

Para controlar as possíveis interações entre os módulos, é definida uma interface explícita para cada módulo

A interface especifica os procedimentos e as variáveis que podem ser acessados a partir de outros métodos

Os módulos são implementados de forma a ocultar todas as informações sobre eles, exceto o que está disponível por meio de sua interface

Contanto que sua interface permaneça a mesma, a implementação pode ser alterada sem afetar os usuários do módulo

Programação com Interfaces

Limitações do sistema distribuído

- Não é possível um módulo ser executado em um processo e acessar as variáveis de um módulo em outro processo
- As passagens de parâmetro são por valor
- Os endereços de um processo não são válidos em outro processo remoto
  - Os endereços não podem ser passados como argumentos nem retornados como resultado de chamadas para módulos remotos

Programação com Interfaces

Linguagens de definição de interface (IDL) são projetadas para permitir que procedimentos implementados em diferentes linguagens invoquem uns aos outros

- 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

Programação com Interfaces

Exemplo de IDL do CORBA

```
Person p = new Person("Smith", "London", 1984);
```

```
// 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( );
};
```

Semântica das Chamadas

Normalmente, o cliente retransmite uma mensagem após um certo *time-out*

Como então determinar quantas vezes o procedimento remoto é executado ?

- Uma vez: quando tudo funciona corretamente
- Zero vezes: quando a mensagem do cliente não chega ao servidor, apesar das várias retransmissões
- Mais de uma vez: quando a resposta do servidor se perde

Semântica de Chamadas

Tipos de semânticas em chamadas RPC:

- *Exactly Once* ("exatamente uma vez"):
  - Desejada, mas difícil de conseguir
- *At Least Once* ("pelo menos uma vez"):
  - Basta que cliente tente até conseguir
  - Procedimento remoto pode ser executado mais de uma vez
- *At Most Once* ("no máximo uma vez")
  - Procedimento remoto pode ser executado zero ou uma vez

Semântica de Chamadas

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*

Tipos de Operações

Operações Idempotentes:

- Podem ser executadas qualquer número de vezes, pois não tem efeito colateral
- Exemplo: consultas em geral (hora, saldo etc)
- Semântica "*at least once*"

Operações não Idempotentes:

- Possuem efeito colateral.
- Exemplo: atualizações (transferência bancária, gravação de um registro etc)
- Semântica "*exactly once*"

Transparência

Tornar as chamadas de procedimentos remotos o mais parecidas possível com as chamadas de procedimentos locais

As chamadas de procedimento remoto são mais vulneráveis às falhas do que as locais, pois envolvem uma rede, outro computador e outro processo

A latência de um procedimento remoto é muito maior do que a de um local

Comunicação entre Processos

Abstrações de mais alto nível:

- Protocolos de requisição-resposta
- Chamada de procedimento remoto
- **Invocação de método remoto**

Invocação de Método Remoto

Extensão da RPC para o mundo dos objetos distribuídos

Um objeto chamador pode invocar um método em um objeto potencialmente remoto

Invocação de Método Remoto

Semelhanças entre RMI e RPC

- Suportam programação com interfaces
- São construídas sobre protocolos de requisição-resposta
- Oferecem diversas semânticas de chamada (ex. pelo menos uma vez, no máximo uma vez)
- Oferecem um nível de transparência semelhante
  - Chamadas locais e remotas com a mesma sintaxe

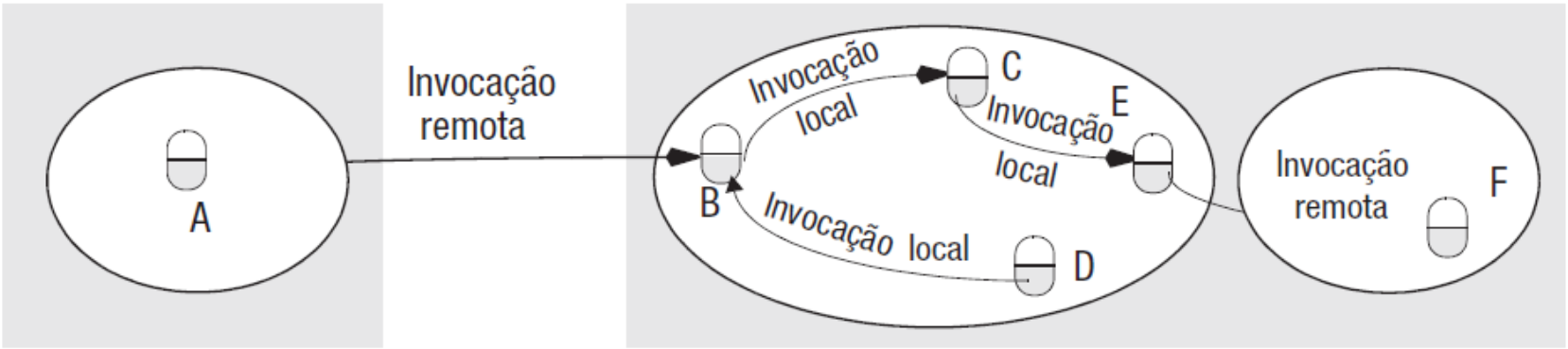
Invocação de Método Remoto

Diferenças entre RMI e RPC

- Na RMI o programador pode usar todo o poder expressivo da programação orientada a objetos
- Todos os objetos têm referências exclusivas e tais referências podem ser passadas como parâmetros
  - Semântica de passagem de parâmetros mais rica que na RPC

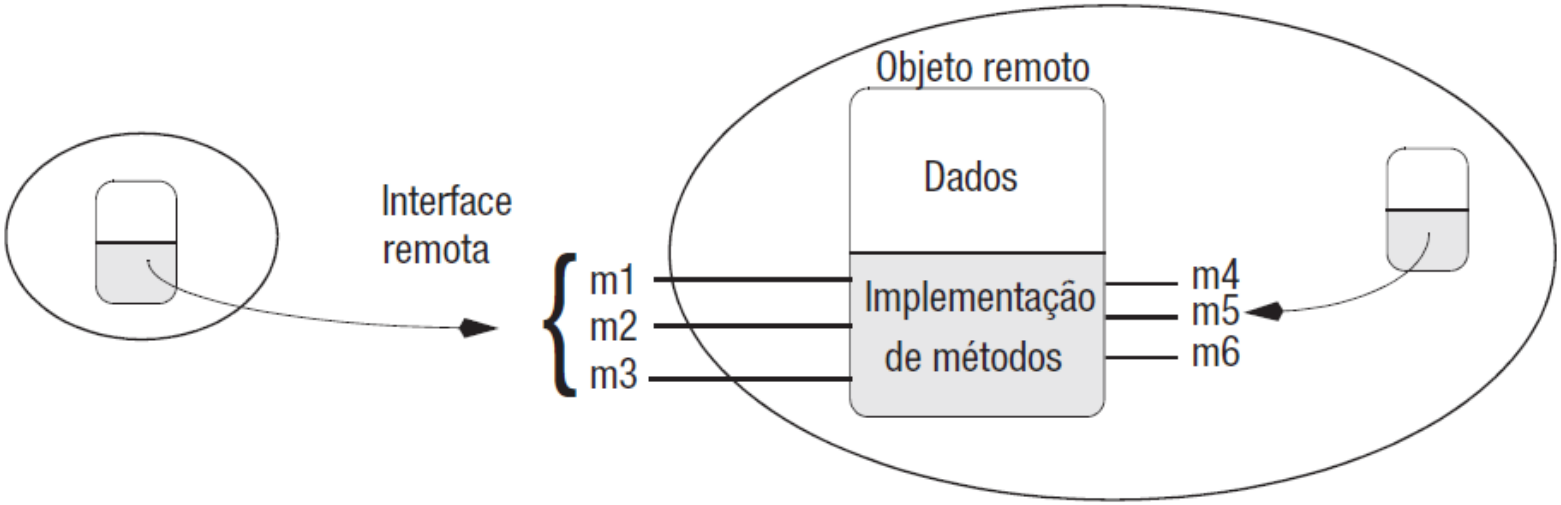
Invocação de Método Remoto

Invocação a métodos locais e remotos



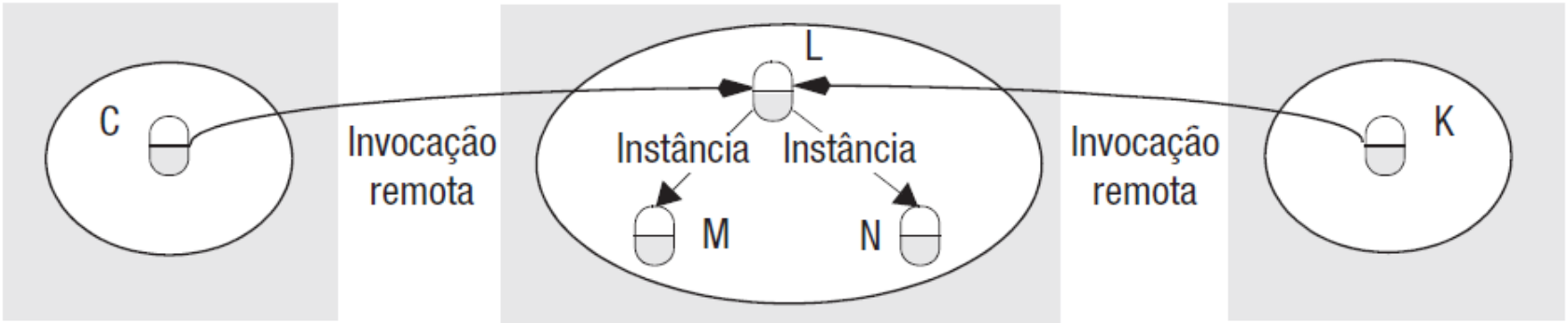
Invocação de Método Remoto

Um objeto remoto e sua interface remota



Invocação de Método Remoto

Instanciação de objetos remotos



Tarefa 3 – postar no Canvas até 14/03/2021

1. Verifique se as seguintes operações são idempotentes:

   i) pressionar o botão “subir” (elevador);

   ii) escrever dados em um arquivo;

   iii) anexar dados em um arquivo.

   É uma condição necessária para a idempotência o fato de a operação não estar associada a nenhum estado?

Tarefa 3 – postar no Canvas até 14/03/2021

2. Uma interface *Election* fornece dois métodos remotos:
   - **vote**: este método possui dois parâmetros por meio dos quais o cliente fornece o nome de um candidato (um `string`) e o "número do votante" (um valor inteiro usado para garantir que cada usuário vote apenas uma vez). Os números dos votantes são alocados esparsamente a partir do intervalo de inteiros para torná-los difíceis de adivinhar.
   - **result**: este método possui dois parâmetros com os quais o servidor fornece para o cliente o nome de um candidato e o número de votos desse candidato.

Tarefa 3 – postar no Canvas até 14/03/2021

Defina a interface do serviço *Election* na IDL CORBA e na RMI Java. Note que a IDL CORBA fornece o tipo long para inteiros de 32 bits. Compare os métodos nas duas linguagens, para especificar argumentos de entrada e saída.

Tarefa 3 – postar no Canvas até 14/03/2021

3. O serviço *Election* deve garantir que um voto seja registrado quando o usuário achar que depositou o voto. Discuta o efeito da semântica talvez no serviço *Election*. A semântica pelo menos uma vez seria aceitável para o serviço *Election* ou você recomendaria a semântica no máximo uma vez?

Tarefa 3 – postar no Canvas até 14/03/2021

4. Um protocolo de requisição-resposta é implementado em um serviço de comunicação com falhas por omissão para fornecer semântica de invocação pelo menos uma vez. No primeiro caso, o desenvolvedor presume um sistema assíncrono distribuído. No segundo caso, o desenvolvedor presume que o tempo máximo para a comunicação e a execução de um método remoto é T. De que maneira esta última suposição simplifica a implementação?