Invocar código em determinada máquina e obter apenas o resultado.
Tecnologias: RPC (linguagens procedurais) e RMI (linguagens procedurais adaptadas para orientação a objetos).

Paradigmas de invocação remota

- Protocolos de requisição-resposta;
- Remote procedure call (RPC);
- Remote Method Invocation (RMI);

Protocolos de requisição-resposta

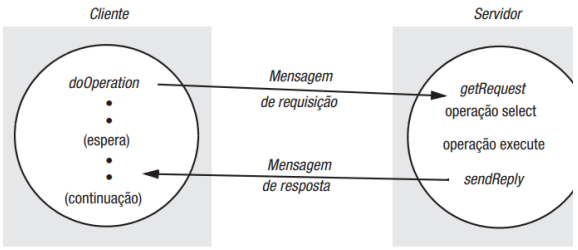

Figura 1. Comunicação por requisição-resposta.

O cliente executa uma operação que faz uma invocação remota e envia uma mensagem empacotada pela rede, ao chegar é desempacotada, executa o que tiver que executar e envia uma resposta ao cliente.

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.

Figura 2. Operações do protocolo de requisição-resposta.

Método tradicional de implementação do protocolo de requisição-resposta.

RPC

- Interfaces em sistemas distribuídos;
  Especificação de funções que podem ser invocadas remotamente
- Vantagens no uso de interface:
  - Oculta do programador detalhes de implementação, tais como a linguagem e plataforma utilizadas;
    Essa ocultação torna os detalhes de implementação responsabilidade apenas do servidor, o cliente apenas utiliza.
  - Possibilita mudanças na implementação desde que a interface original seja mantida compatível com a original.
    Permite que o desenvolvimento da função ou sistema remoto possa ser alterado ou melhorado sem ser necessário o cliente alterar seu código.

Limitações

- Não é possível o acesso direto às variáveis de módulo. Utilizam-se procedimentos get e set;
- Não é possível chamada por referência. Os parâmetros devem ser especificados como de entrada ou de saída;
  Não pode passar ponteiros por parâmetro dentro de uma chamada. O que se pode fazer é utilizar parâmetros de entrada (usados para enviar determinado dado ao servidor) ou saída (enviam resultados aos clientes).
- Não é possível passar endereços como argumentos.
  Sem ponteiros nos argumentos!!

Exemplo de entrada do 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( );
};
```

Figura 3. Exemplo de IDL do CORBA.

Semânticas de RPC

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*

Figura 4. Semânticas de chamada

3 semânticas relacionadas a tolerância a falhas do envio de em requisições:

- **Talvez:** Um procedimento que foi invocado pode ser executado ou não uma vez.
- **Pelo menos uma vez:** Recebe o resultado de uma falha de invocação pelo menos uma vez.
- **No máximo uma vez:** Recebe uma vez ou nada.

Transparência em RPC

Tem como objetivo garantir que a invocação de dispositivos locais e remotos sejam o mais semelhante possível.

- Busca tornar iguais as chamadas locais e remotas a procedimentos.
- Oculta detalhes do empacotamento e da troca de mensagens;
  O desenvolvedor não precisa se preocupar com detalhes.
- RPC é mais vulnerável a falhas do que as locais;
  Invocações remotas são mais vulneráveis pois possuem

mais pontos de falha que as locais. Um ponto de falha seria a rede: pode ser perdida a mensagem ou chegar atrasada.

- A latência é muito maior;
  Na invocação remota a mensagem precisa viajar pela rede para chegar ao destino e receber a resposta, isso gera uma latência muito maior.
- Diferença na passagem de parâmetros.
  Não pode-se usar ponteiros.

Implementação de RPC

- Um procedimento stub para cada procedimento da interface;
  Todo processo tem um **stub**: um código adicional associado ao servidor e ao cliente responsável por fazer o processo de empacotamento e desempacotamento dos dados.
- O stub empacota os argumentos e o id do procedimento e envia ao servidor;
- Ao receber a mensagem o despachante no servidor envia ao stub no servidor;
  Ao receber a mensagem o componente interno denominado despachante vai receber essa requisição e baseado no id do procedimento vai encaminhar a requisição para o stub do servidor.
- O stub no servidor desempacota os argumentos e repassa ao procedimento de serviço;

- Os stubs e o despachante são gerados automaticamente pelo compilador.

Implementação de RPC

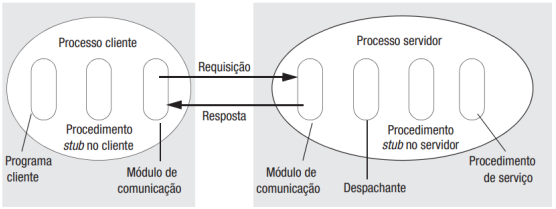

Figura 5. Função dos procedimentos stub no cliente e no servidor na RPC.

No lado do **Cliente** tem-se a programação que o cliente vai usar, associado a ela tem-se o stub (módulo que será adicionado ao código do cliente no momento da compilação) e o módulo de comunicação (responsável por pegar as requisições do cliente ao servidor e encaminhar ao servidor).

No lado do **Servidor** tem-se o módulo de comunicação também, ele recebe a requisição e encaminha ao módulo despachante, o

despachante tem a informação de cada procedimento e redireciona para o stub do servidor. Quando chega ao stub, ele desempacota a requisição e realiza o procedimento de serviço (onde está implementada a função). Por fim, o retorno da requisição é repassado novamente ao stub que empacota e envia o despachante, e módulo de comunicação envia ao cliente.

Assim que a mensagem chega no cliente, o stub de desempacota e mensagem é entregue no formato adequado ao cliente.

Exemplo usando rpcgen

Objetivo: executar procedimento remoto que soma e subtrai dois números;

IDF: arquivo de definição de interface

Criar arquivo IDF: calcula.x

```
struct operandos {
   int x;
   int y;
};
program PROG {
   version VERSAO {
      int ADD(operandos) = 1;
      int SUB(operandos) = 2;
   } = 100;
} = 55555555;
```

Figura 6. Criação do arquivo IDF.

- Execute o comando a seguir:
    - `rpcgen -a calcula.x`

Alguns arquivos são gerados a partir desse comando.

Arquivos gerados:

Calcula.h	Arquivo com as definições que deverão estar inclusas nos códigos cliente e servidor
Calcula_client.c	Arquivo contendo o esqueleto do programa principal do lado cliente
Calcula_clnt.c	Arquivo contendo o stub do cliente
Calcula_xdr.c	Contém as funções xdr necessárias para a conversão dos parâmetros a serem passados entre host
Calcula_svc.c	Contém o programa principal do lado servidor.
Calcula_server.c	Contém o esqueleto das rotinas a serem chamadas no lado servidor
Makefile.calcula	Deve ser renomeado para Makefile. Contém as diretivas de compilação para a ferramenta *make*.

Figura 7. Arquivos gerados pelo rpcgen.

Calcula.h: contém todas definições e estrutura criada.

Calcula_client.c: Arquivo no qual o desenvolvedor deve implementar seu código e fazer procedimentos remotos.

Calcula_clnt.c: stub do cliente, contém as informações de empacotamento e desempacotamento.

Calcula_xdr.c: Contém código para empacotamento e desempacotamento.

Calcula_svc.c: Programa principal do servidor.

Calcula_server.c: Contém especificação do código que vai ser criado do lado do servidor.

Makefile.calcula: Arquivo para realizar compilação do código de maneira automatizada.

Antes de executar

- Compile usando o Makefile;
- Instale o pacote rpcbind;
  - apt-get install rpcbind
- Pronto!

Invocação de método remoto

- Extensão do RPC para o mundo de objetos distribuídos;
- Semelhanças
  - Suportam programação com interfaces;
    Em ambos deve-se definir uma interface especificando que métodos podem ser invocados remotamente.
  - São construídos sobre protocolos requisição-resposta;
    A ideia é a mesma nos dois métodos.
  - Nível semelhante de transparência;
- Diferenças
  - RMI conta com toda a expressividade de orientação;
    A orientação a objetos ajuda no trabalho do desenvolvedor.
  - RMI tem uma riqueza maior na passagem de parâmetros que o RPC;
    No RMI pode-se passar objetos por parâmetros.

Questões de projeto para o RMI

- Modelo de objeto;
  Os objetos irão interagir através de invocação de métodos.
- Referências de objeto;
  Forma de acessar outros objetos.
- Interfaces;
  Usadas para definir as assinaturas dos métodos (métodos, argumentos e tipos), sem nenhuma implementação em si.
- Ações;
  Se refere a iniciar a invocação de um método em outro objeto.
- Exceções;
  Modo de tratar condições de erros.
- Coleta de lixo;
  Modo de liberar espaço de memória de objetos não referenciados.
- Objetos distribuídos;
  Possibilidade de espalhar instâncias de objetos em diversas máquinas.
- Referência de objetos remotos;
  Invocação de objetos localizados em uma outra máquina.

- Interface remota;
  Garante que os objetos que vão oferecer métodos para invocação remota tenham uma interface remota.

Modelo de objeto distribuído

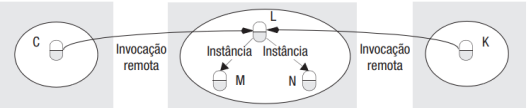

Figura 8. Instanciação de objetos remotos.

Um objeto A faz invocação remota a B, da mesma forma de uma invocação local.

Objeto remoto e sua interface montada

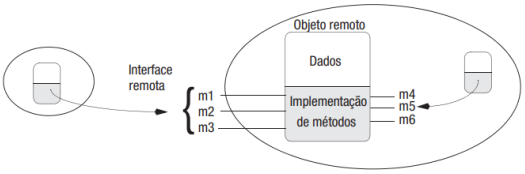

Figura 9. Um objeto remoto e sua interface remota.

Ao definir um objeto remoto deve-se definir quais métodos podem ser invocados remotamente e quais apenas localmente.

Implementação de RMI

- Módulos de comunicação;
  Responsável pelo envio e recebimento de respostas.
- Módulos de referência remota;
  Componente do RMI responsável por fazer a transformação de referências de objetos locais e objetos remotos e também pela criação de referência de objeto remoto.
- Serventes;
  Fornece os métodos remotos. Ele trata todas as requisições remotas que chegam ao servidor e executa o método.
- Softwares RMI:
  - Proxy;
    Elemento do cliente que recebe as requisições remotas e empacota-as.
  - Despachante;
    Responsável pelo redirecionamento da requisição de invocação remota para o esqueleto adequado.
  - Esqueleto.
    Realiza o desempacotamento do lado do servidor para o servente.

Implementação de RMI

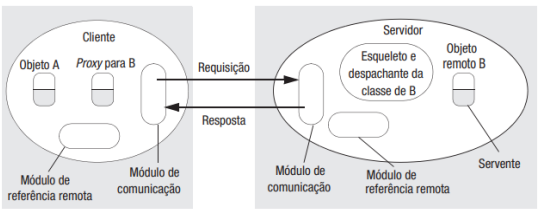

Figura 10. A função do proxy e do esqueleto na invocação de método remoto.

Implementação de RMI

- O vinculador (binder): JAVA RMI = registry;
  elemento com função de relacionar cada objeto local e disponibilizar um endereço remoto.
- Programas e servidores: usada para criar serventes;
- Métodos de fábrica: método usado para criar novos objetos remotos;
  métodos com função de criar objetos remotos que terão seus próprios métodos invocados remotamente.
- Ativação dinâmica: Permite a invocação remota de objetos que não foram registrados;
  em tempo de execução os objetos são localizados e invocados remotamente.
- Esqueletos dinâmicos: permite ao servidor conter objetos remotos cujas interfaces não eram conhecidas no momento da compilação.
  define em tempo de execução quais métodos exatamente podem ser invocados remotamente.

Exemplo: JAVA RMI

- Um servidor de operações;
- Passos:
  - Definir a interface do objeto remoto;
  - Implementação da classe do objeto remoto;
  - Compilação dos fontes com javac;
  - Criação dos stub e skeleton com rmic;
  - Aplicação servidora que instancia e registra o objeto remoto no serviço de registros;
  - Aplicação cliente que consulta o serviço de registros.

Definição de Interface

```
import java.rmi.*;

public interface InterfaceCalcServer extends
Remote {
   public double soma(double a, double b) throws
RemoteException;
   public double subtrai(double a, double b) throws
RemoteException;
   public double multiplica(double a, double b)
throws RemoteException;
  public double divide(double a, double b) throws
RemoteException;
}
```

A interface implementa outra interface (Remote) que garante que os objetos possam ser invocados remotamente.

Implementação do objeto remoto

```
import java.rmi.*;
import java.rmi.server.*;
public class CalcServer extends UnicastRemoteObject implements InterfaceCalcServer
{
   public CalcServer() throws RemoteException   {
      System.out.println("Novo Servidor instanciado...");
   }
   public double soma(double a, double b) throws RemoteException   {
      return a+b;
   }
  public double subtrai(double a, double b) throws RemoteException   {
      return a-b;
   }
 public double multiplica(double a, double b) throws RemoteException {
      return a*b;
   }
 public double divide(double a, double b) throws RemoteException   {
      return a/b;
   }
}
```

UnicastRemoteObjeto: permite invocações remotas no modo unicast.

Criação da aplicação servidora

```
import java.rmi.*;
import java.rmi.server.*;
public class CalcServer extends UnicastRemoteObject implements InterfaceCalcServer
{
   public CalcServer() throws RemoteException   {
      System.out.println("Novo Servidor instanciado...");
   }
   public double soma(double a, double b) throws RemoteException   {
      return a+b;
   }
  public double subtrai(double a, double b) throws RemoteException   {
      return a-b;
   }
 public double multiplica(double a, double b) throws RemoteException {
      return a*b;
   }
 public double divide(double a, double b) throws RemoteException   {
      return a/b;
   }
}
```

getRegistry: retorna uma referência para o servidor de serviços.
rebind: associa determinado objeto ao registry.

Criação aplicação cliente

```
public class Cliente {
public Cliente() {
   System.out.println("Iniciando o Cliente...");
   try  {
      Registry registry = LocateRegistry.getRegistry("servidor");
      msi = (InterfaceCalcServer) registry.lookup("CalcServer_1");
}   catch (Exception e)  {
         System.out.println("Falhou a inicialização do
Cliente.\n"+e);
         System.exit(0);
      }
}
```

Busca referência do servidor de serviços e busca o objeto remoto o qual se quer referenciar.

Aplicação Cliente

```
public double area(double a, double b) throws
RemoteException      {
      return msi.multiplica(a,b);
}
public static void main (String[] argv){
    Cliente c = new Cliente();
     try {
           System.out.println("Area: " + c.area(20.0,40.0));
     } catch (Exception e)   {
           System.out.println("Exceção durante chamadas
remotas:" +e);
        }
}
  private InterfaceCalcServer msi;
}
```

Invoca-se os métodos remotos tal como se fossem locais (transparência).

Executar o programa exemplo

- rmiregistry no servidor de registros (executar na mesma pasta onde estão os .class dos objetos remotos);
- javac IniciaServidor.java
- javac Cliente.java
- Java IniciaServidor
- Java Cliente

- Java –Djava.rmi.server.hostname=<IP_SERVIDOR> IniciaServidor (Se problema com endereço do servidor)
- OBS: Atenção para a configuração correta do nome e IP da máquina nos arquivos /etc/hostname e /etc/hosts