Capítulo V – Sistemas de Objectos Distribuídos

From: Coulouris, Dollimore and Kindberg
Distributed Systems: Concepts and Design
Edition 3, © Addison-Wesley 2001

From: Wolfgang Emmerich
Engineering Distributed Objects
John Wiley & Sons, Ltd 2000

1 – O modelo de objectos

2 – Invocação remota de objectos
(Remote Method Invocation)

2.1 – semântica de invocação

2.2 – implementação do RMI

3 – Caso de estudo – Java RMI

Paula Prata,

Departamento de Informática
da UBI

http://www.di.ubi.pt/~pprata

Evolução dos sistemas cliente-servidor

. O modelo clássico (pedido-resposta) começou por ser implementado por instruções de baixo nível

Ex. Sockets <- demasiado "complicado" para o comum dos programadores ...

. Com o aparecimento das linguagens procedimentais, passou-se à utilização do modelo de

RPC (Remote Procedure Calling)

. O sucesso das linguagens OO motivou o aparecimento dos sistemas de Objectos Distribuídos

RMI – Remote Method Invocation

1 - O modelo de Objectos

```
Pessoa cliente;

...

String morada = cliente.obtemMorada ()
```

Esta variável (e programa) existem na máquina cliente

Num SD, o objecto cliente pode existir numa máquina remota (o servidor)

Ao ser enviada a mensagem obtemMorada ao objecto cliente, o sistema encarrega-se de obter os dados do cliente no servidor.

Referências para objecto locais

Vs referências para objectos remotos

. Num sistema OO os objectos são acedidos através das suas referências

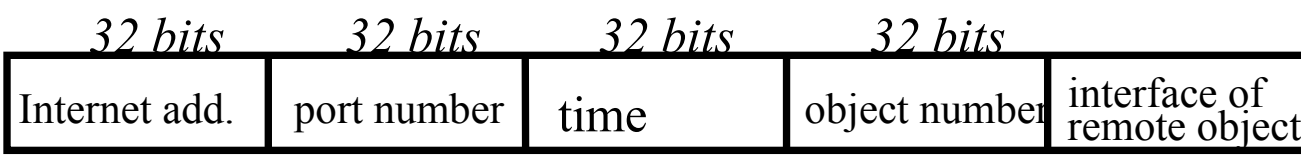
. Para invocar um método precisamos da referência do objecto, do nome do método e dos argumentos correspondentes.

. Referências, podem ser atribuídas a variáveis, passadas como argumentos e devolvidas como resultado de métodos

Referência remota

Identificador que pode ser usado no âmbito de um SD para se referir a um particular e único objecto remoto

(Acetato 19 cap.IV)



Referência remota ...

Referências remotas são idênticas a referências locais, nos seguintes aspectos:

1- o objecto que recebe a invocação do método é especificado como o objecto local

2 – referências remotas podem ser passadas como argumentos e resultados de invocação de métodos remotos

Passagem de parâmetros (entre processos diferentes)

cópia versus referência (tipos primitivos são passados por cópia)

Passagem de objectos remotos

- por referência (são objectos de acesso global)

na prática um referência remota "aponta" para um objecto local que funciona como um proxy do objecto remoto

Na passagem de um objecto remoto, é passado um proxy que é guardado na maquina receptora, e para o qual vão ser passadas as futuras invocações ao objecto

Passagem de objectos locais *(que implementem a interface java.io.Serializable)*

- por cópia (a referência não faz sentido na máquina remota)

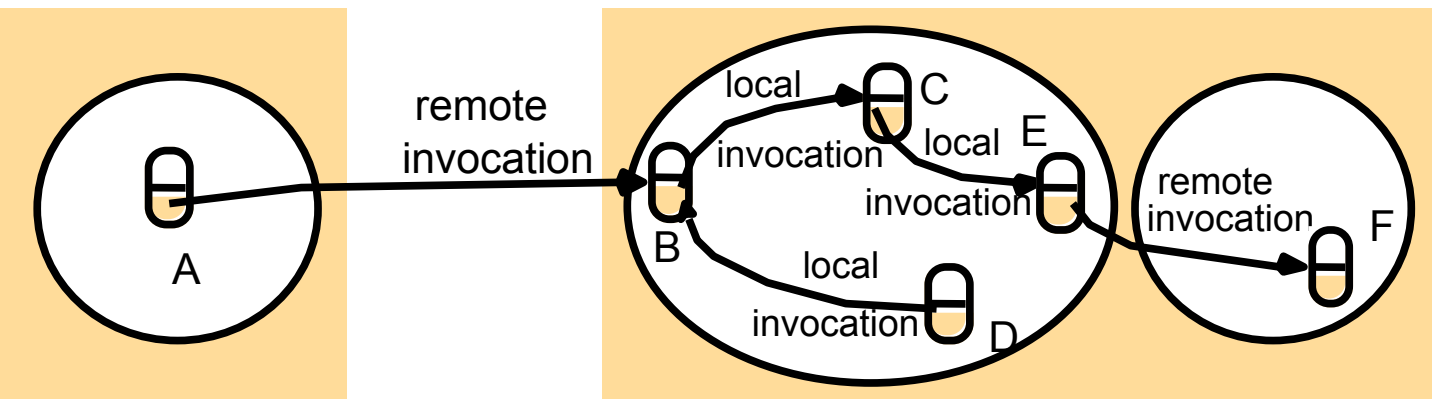
Do lado do receptor é criado um novo objecto que pode ser guardado localmente

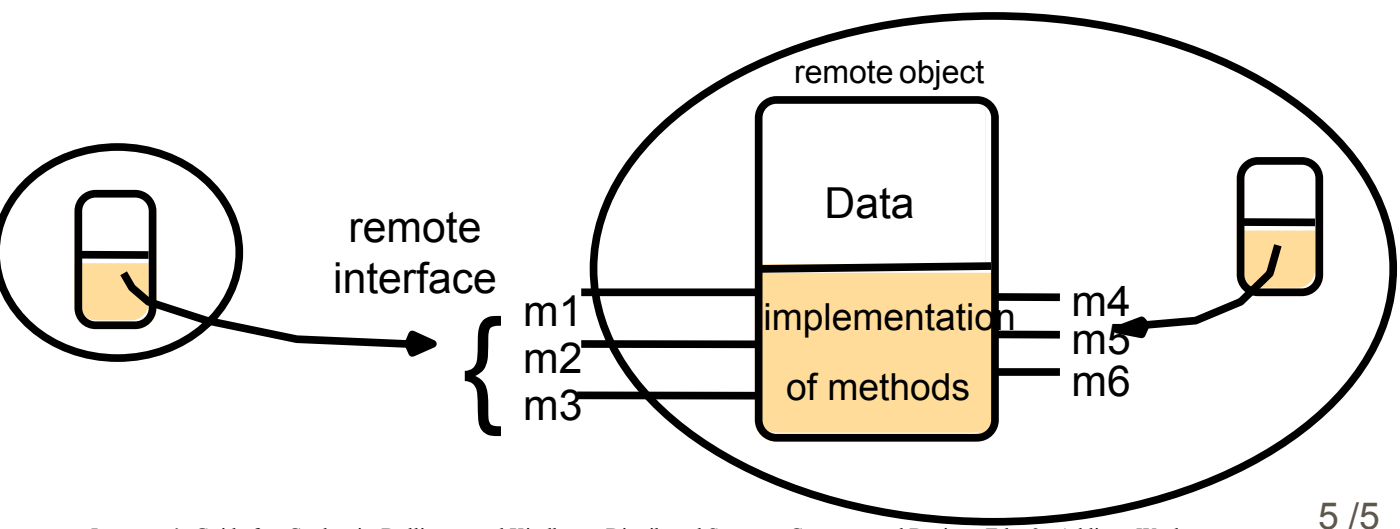
Interface

Especificação sintáctica de um conjunto de métodos. Uma classe que implemente a interface terá obrigatoriamente que implementar todos os métodos da interface

Interface remota,

Cada objecto remoto tem uma interface remota que especifica quais dos seus métodos podem ser invocados remotamente.





Método locais

A invocação de um método local resulta na execução do código correspondente no objecto receptor. No final o controlo de execução retorna ao objecto invocador.

A invocação de um método pode resultar em:

- o estado do método é alterado

- outras invocações, noutros objectos têm lugar

Método remotos

A invocação de um método remoto pode resultar na invocação de outros objectos tal como numa invocação local.

Eventualmente os objectos envolvidos podem existir em máquinas diferentes.

Quando uma invocação atravessa a barreira de um processo ou computador, tem lugar uma invocação de um procedimento remoto (Remote Method Invocation – RMI)

Para um objecto fazer uma invocação remota sobre um objecto tem que possuir a sua referência remota

Excepções locais

Quando ocorre uma situação de erro (recuperável) é gerada uma excepção (pelo sistema de execução ou pelo próprio código do programador)

É possível capturar uma excepção e transferir o controlo de execução para um bloco de código que tratará a condição de erro

Excepções remotas

Uma invocação remota além das excepções ocorridas no processo receptor, pode também gerar excepções devido a:
. erro na transmissão dos argumentos,
. o processo que contém o objecto remoto "falhou" (crashed)
. está tão ocupado que não consegue responder,
. o resultado da invocação perdeu-se.

Reciclagem (Garbage Collection - GC)

Meio de libertar o espaço ocupado por objectos que já não são necessários.

Java detecta automaticamente os objectos que já não são acessíveis

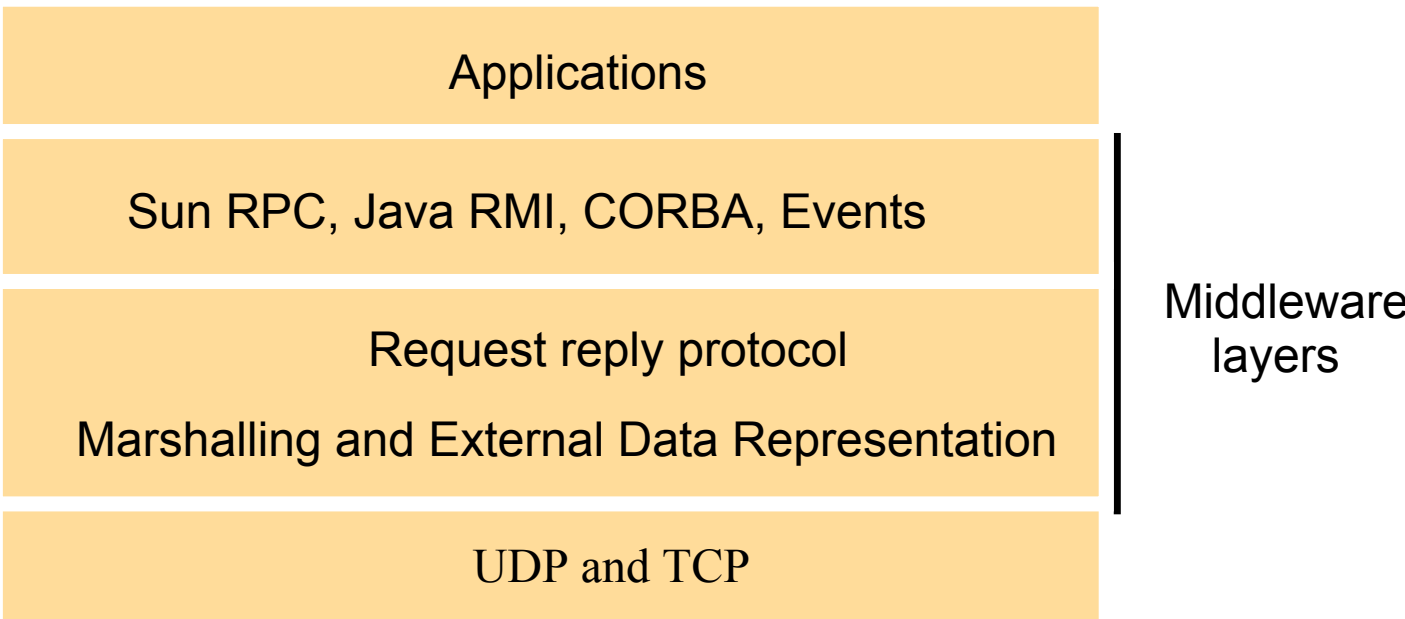
Em C++ o programador é o responsável por libertar esse espaço de memória

Reciclagem de Objectos Distribuídos

Extensão do GC tradicional

Geralmente baseada em contagem de referências

2 – Invocação remota de objectos



RPC/RMI Middleware de utilização geral

Common Object Request Broker Architecture (CORBA)

- Object Management Group (OMG)

Distributed Component Object Model (DCOM)

- Microsoft

Remote Procedure Call (RPC)

- Sun, DCE, …

Remote Method Invocation - RMI

- Java RMI (Sun)
- CORBA é um forma de RMI

Simple Object Access Protocol (SOAP)

- Microsoft, Sun, …

2 – Invocação remota de objectos

2.1 – Semântica da invocação

Fault tolerance measures			*Invocation semantics*
Retransmit request message	*Duplicate filtering*	*Re-execute procedure or retransmit reply*	
No	Not applicable	Not applicable	*Maybe*
Yes	No	Re-execute procedure	*At-least-once*
Yes	Yes	Retransmit reply	*At-most-once*

. Java RMI e CORBA fornecem "At-most-once"

O invocador ou recebe uma resposta e, nesse caso, sabe que o método foi executado uma única vez, ou recebe uma excepção o que significa que o método ou foi executado uma vez ou não foi executado.

. CORBA permite "maybe" para métodos que não devolvem resultado

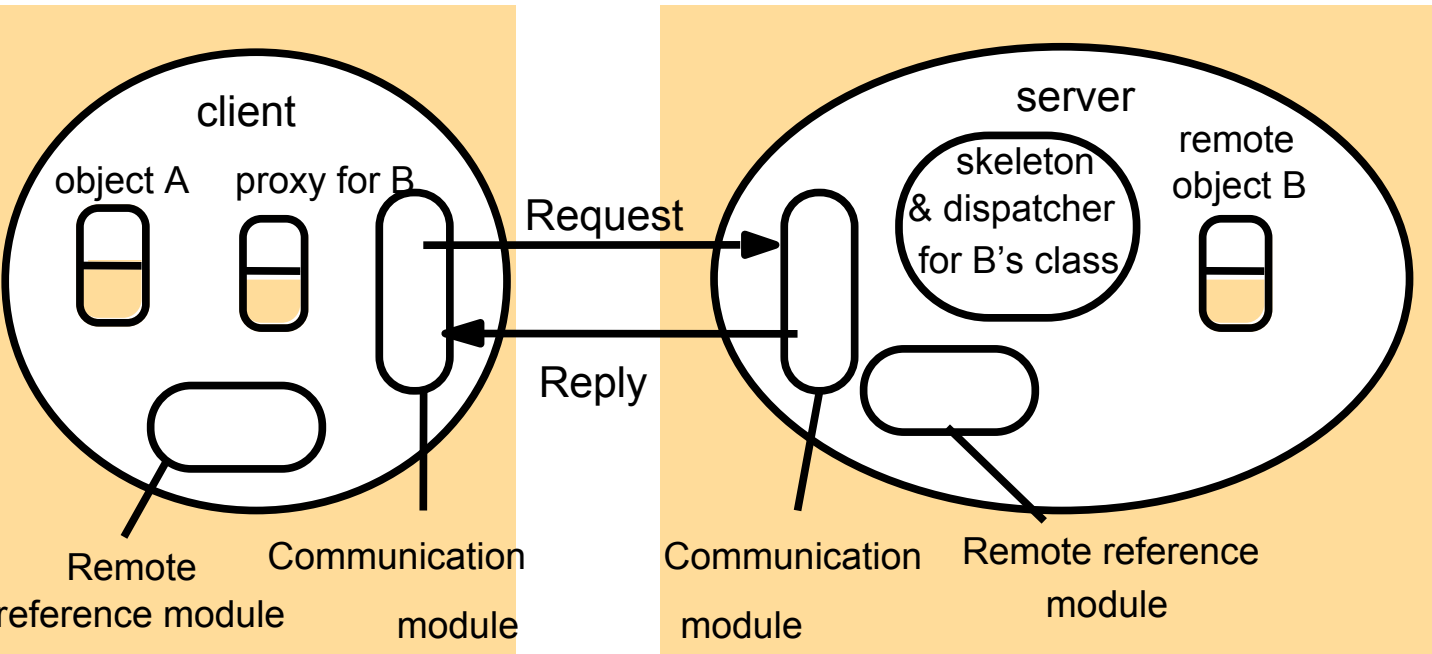
O invocador não sabe se o método foi ou não executado

. Sun RPC fornece "At-least-once"

O invocador ou recebe uma resposta, nesse caso, sabe que o método foi executado pelo menos uma vez, ou recebe uma excepção informando que não foi recebido resultado, neste caso, o servidor pode ter falhado. Em qualquer dos casos, execuções repetidas podem ter originado valores errados.

Instructor's Guide for Coulouris, Dollimore and Kindberg Distributed Systems: Concepts and Design Edn. 3 Addison-Wesley Publishers 2000

2.2 – Implementação do RMI



Um objecto A invoca um método no objecto remoto B

Módulo de Comunicação

Implementa o protocolo pedido-resposta

Módulo de Referência remota

Mantém uma tabela de referências remotas, que contém a correspondência entre referências locais e remotas.

A tabela do servidor conterá uma entrada para o objecto B

A tabela do cliente conterá uma entrada para o proxy de B

2.2 – Implementação do RMI ...

O software RMI

Camada de software entre a aplicação e os módulos de comunicação e referência

Proxy

Torna transparente a invocação do método remoto

Recebe a invocação, serializa os argumentos e envia a invocação através do módulo de comunicação;

Quando recebe o resultado, desserializa-o e envia-o ao cliente.

Dispatcher

Recebe o pedido do módulo de comunicação e encaminha-o para um skeleton.

Skeleton

Desserializa os argumentos, invoca o método no objecto local, recebe o resultado e encaminha-o no sentido inverso.

3 – Caso de estudo: Java RMI

Construção de uma aplicação cliente-servidor em Java RMI

Suponhamos um objecto servidor que implementa uma calculadora simples

1 – Definir a interface do objecto remoto

. *tem de ser definida como subinterface de* **java.rmi.remote**

. *todos os métodos têm de lançar (*throws) *a excepção* **java.rmi.remoteException**

2 – Implementar a interface remota

. *o objecto remoto deve ser subclasse de* **java.rmi.server.UnicastRemoteObject**

e implementar a interface remota (definida em 1)

Nota: qualquer classe usada como parâmetro ou resultado tem que ser serializável, i.e. implementar a interface java.io.Serializable

3 – Implementar o servidor

. *Criar uma instância do objecto e ligá-la ao serviço de nomes* - RMI Registry

4 – Desenvolver o cliente (programa ou applet) que usa a interface remota

5 – Gerar stubs (proxies) e skeletons

6 – Iniciar o "RMI registry"

7 – Criar um ficheiro com politica de segurança

8 – Executar o servidor e o cliente

1 –Interface do objecto remoto

```
public interface Calculator  extends java.rmi.Remote {
   public long add(long a, long b) throws java.rmi.RemoteException;
   public long sub(long a, long b) throws java.rmi.RemoteException;
   public long mul(long a, long b) throws java.rmi.RemoteException;
   public long div(long a, long b) throws java.rmi.RemoteException;
}
```

2 – Implementar a interface remota

```
public class CalculatorImpl extends
                java.rmi.server.UnicastRemoteObject
                                        implements Calculator {
 // Implementations must have an explicit constructor in order to declare
//the RemoteException exception
   public CalculatorImpl() throws java.rmi.RemoteException {
      super();
   }
   public long add(long a, long b) throws java.rmi.RemoteException {
      return a + b;
   }
   public long sub(long a, long b) throws java.rmi.RemoteException {
      return a - b;
   }
   public long mul(long a, long b) throws java.rmi.RemoteException {
      return a * b;
   }
   public long div(long a, long b) throws java.rmi.RemoteException {
      return a / b;
   }
}
```

3 – Implementar o servidor

i) Criar um gestor de segurança e instalá-lo (!)

ii) Criar uma instância do objecto remoto

iii) Registar o objecto remoto no serviço de nomes (!)

```
import java.rmi.*;
public class CalculatorServer {
    public CalculatorServer() {
// i)
        System.setSecurityManager ( new RMISecurityManager());
        try {
// ii)
          Calculator c = new CalculatorImpl();
// iii)
          Naming.rebind("rmi://localhost:1099/CalculatorService", c);
        } catch (Exception e) {
            System.out.println("Trouble: " + e);
        }
    }
  public static void main(String args[]) {
      new CalculatorServer();
  }
}
```

Notas: . Os servidores unicast são o tipo mais simples de servidor. Referências para ele são válidas apenas enquanto o processo que instanciou o objecto está a correr. Comunicação cliente/servidor usa o protocolo TCP.

4 – Implementar o cliente

i)Obter a referência do objecto a partir do serviço de nomes (RMI registry)

. É necessário saber o nome da máquina e do objecto remoto

ii) Invocar operações remotas (métodos no objecto remoto)

. Podem gerar uma java.rmi.RemoteException

```
import java.rmi.Naming;
import java.rmi.RemoteException;
public class CalculatorClient {
    public static void main(String[] args) {
        try {
          Calculator c = (Calculator)
              Naming.lookup("rmi://remoteHost/CalculatorService");
            System.out.println( c.sub(4, 3) );
            System.out.println( c.add(4, 5) );
            System.out.println( c.mul(3, 6) );
            System.out.println( c.div(9, 3) );
        }
        catch (RemoteException re) {
          System.out.println("RemoteException");
          System.out.println(re.getMessage());
      }
  }
}
```

5 – Gerar stubs e skeletons

É necessário ainda, - compilar o código: . javac *.java

- gerar os stubs (proxies) e skeletons para os objectos remotos:

. rmic CalculatorIMmpl

Stubs e Skeltons são gerados, usando o compilador de rmi, rmic. Vai gerar dois ficheiros CalculatorImpl_Skell.class CalculatorImpl_Stub.class

Se o stub do cliente é gerado na máquina do servidor, o cliente pode obtê-lo através da rede:

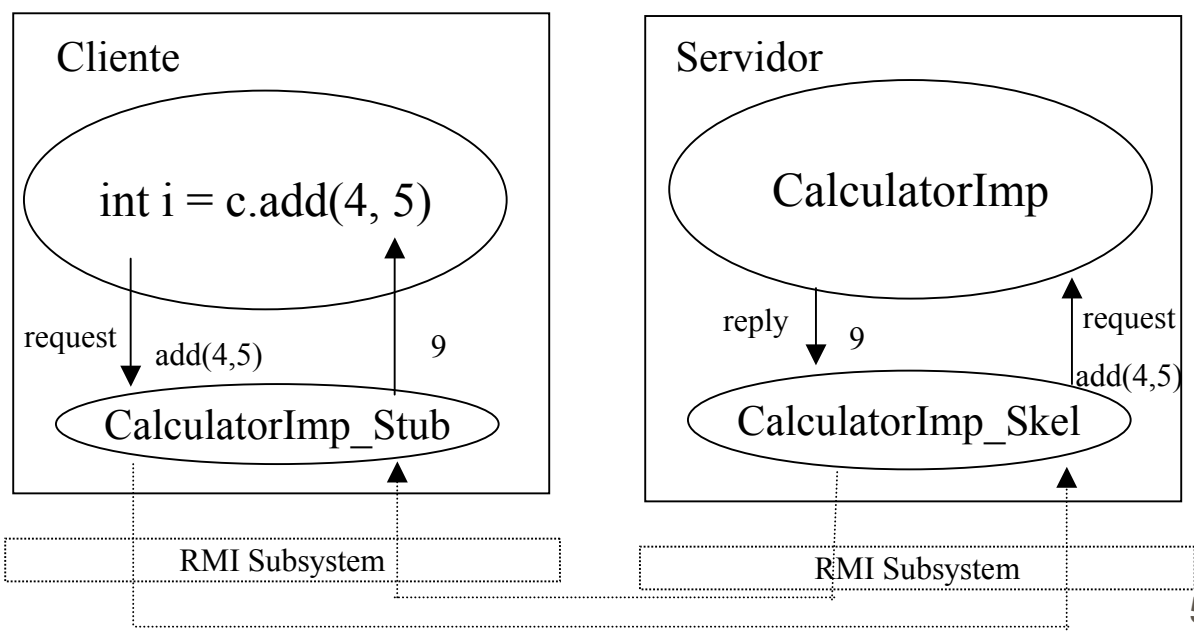
. java –Djava.rmi.server.codebase=file://...

. java –Djava.rmi.server.codebase=http://...

Na versão mais recente, não é necessária a classe Skeleton

A informação necessária é obtida por introspecção.

. rmic –v1.2 ...



6 – Iniciar o "RMI registry"

Um cliente precisa de um meio para obter a referência do objecto remoto.

Um serviço de nomes ("binder") mantém uma tabela com a correspondência entre a referência remota do objecto e um nome textual (URL-style).

O serviço de nomes é usado pelo servidor para registar os seus objectos remotos por um nome e pelo cliente para pesquisar se o servidor contém o objecto desejado.

O serviço de nomes do Java RMI é o RMI registry, baseado na interface java.rmi.registry.Registry que define as operações:

. void rebind (String nomeObjecto, Remote objecto);

// substitui a ligação para o nome especificado

. void unbind (String nomeObjecto)

// remove a referência do registo

. Remote lookup (String nomeObjecto)

// devolve a referência ligada ao nome especificado

. ...

6 – Iniciar o "RMI registry" ...

O nome de um objecto remoto inclui:

. O endereço da máquina que executa o RMIregistry, no qual o objecto remoto está registado

. O porto onde o RMIregistry está à escuta
(valor por omissão: 1099)

. O nome que o objecto remoto tem no RMIregistry

[rmi:] [//] [nomeMaquina] [:port] [/nomeObjecto]

Para podermos executar o servidor é necessário iniciar o RMIregistry:

. rmiregistry & (unix)

. start rmiregistry (windows)

Acesso a um registry:

A classe java.rmi.Naming permite efectuar operações sobre um registry remoto

7 – Criar um ficheiro com a politica de segurança

Em Java, através de um gestor de segurança (security manager) é possível controlar os privilégios do código a executar.

Um servidor só necessita de um security manager se houver transferência de objectos de uma máquina para outra

As permissões de um programa Java são especificadas num ficheiro de policy, por exemplo com:

java -Djava.security.policy =

file:d:\My_work\RMICalculator\Calculator.policy

Exemplo de um ficheiro de policy:

```
grant {
// allows anyone to listen on un-privileged port
permission java.net.SocketPermission "*:1024-65535", "listen,accept,connect";
};
```

Clientes necessitam de permissão de connect

```
permission java.net.SocketPermission “localHost:1024-65535", "connect";
```

Servidores necessitam de permissão de accept e de connect para contactar o serviço de nomes

```
permission java.net.SocketPermission “remoteHost:1099", "accept,connect";
```

Perigoso:

```
permission java.security.AllPermission;
```

8 – Executar o servidor e o cliente

. *java CalculatorServer*

. *java CalculatorClient*

Modelo de threading

A especificação do Java RMI não faz garantias sobre o modelo de threading ao servir invocações remotas:

. Uma única Thread pode ser usada para servir sucessivas invocações

. Pode ser usada uma Thread por invocação

Na prática, invocações remotas que cheguem concorrentemente a um dado objecto são despachadas para Threads diferentes

Implementações de objectos remotos devem fazer o controlo de concorrência