Desarrollo de Aplicaciones en Red

José Rafael Rojano Cáceres
http://www.uv.mx/rrojano

RMI

Remote Method Invocation

Introduction

- Java RMI let's work calling remote methods.
- Underneath it works with the Socket Java API.
- With RMI it's possible to access an object like if it where local.
- David Curtis, principal of OMG, describe RMI as a technology for programming while CORBA is a technology for integration.
- RMI is only oriented a Java and Corba to any language.

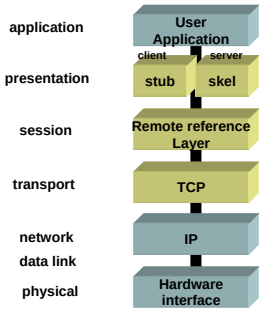
Considerations

- *Object models for Corba and RMI are different, it is possible to use Corba with Java but the object needs to be translated.*
- *Any JVM can use RMI*
- *RMI is simple and easy if only needs to be used Java, Corba will necessary if heterogeneous system are used.*

Considerations

- *Object in java doesn't call methods directly in stead of call their interface*
- *It's possible also to interact with remote object without previous knowledge of them*

RMI architecture



- Presentation, session and transport are the most important layer in RMI
- Presentation, has the stubs and skels that provides a common interface
- Session is conformed by the Remote Reference Layer which manages the creation of object and their calls, for that use the protocol JRMP (*Java Remote Method Protocol*) it's different to IIOP (*Internet Inter ORB Protocol*) from Corba
- The transport layer use by default TCP to send the object

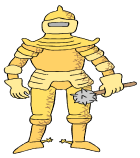
Publish object with RMI

- *In order to communicate RMI defines two basic entities, on the one hand the client, and the other hand the server*
- *Server has the task to create remote objects*
- *Clients have the task to get remote references and invoke it*
- *RMI defines for this task a stub (for client) and a skeleton (for server)*

7

Stub

- *The stub is a proxy or local adapter that let the client communicates with remote objects*



8

Skeleton

- *A skeleton is the complement for the stub, and it is in charge to pass all the request from clients to remote object, and later to return the results.*

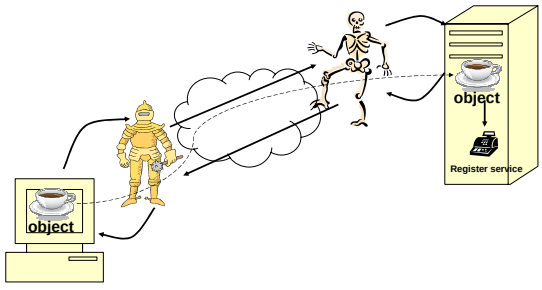
9

Communication (1)

- *The communication process is based in serialization which is not made explicitly in stead it's made by RMI class*
  - **sun.rmi.server.MarshalOutputStream**
- *Any primitive object and objects derivate from the class Serializable are passed by value*
- *When an object is passed by reference, it'll be passing stub or skeletons as pointers from the remote object (increases messaging on the network)*

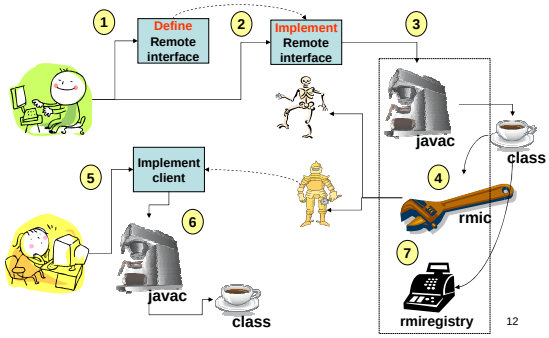
10

Conceptual communication



11

Working with RMI



12

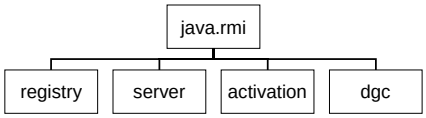
RMI registry (1)

- ***To publish the object can be used three mechanism***
  - **DNS**
  - **JNDI (Java Naming and Directory Service)**
  - **RMI**
- ***In the last case RMI provides a tool called rmiregistry***
- ***rmiregistry provides a service listening by default at the port 1099***
- ***Through this service object can communicates***

RMI registry (2)

- ***Process of registry and discovering***
  - **Start rmiregistry (by default 1099 port)**
  - **The remote object is created and registered with a name**
    - This creates a skeleton
    - A sockets waits for the request
  - **The local object call the remote interface through the invocation with the name used in the registry**
  - **The stub send a reference with the IP, port and ID of the object**
    - The stub open a stream and serializes the object and wait that the layer makes their work
    - The transport layer creates a socket and send the stream
  - **At the remote server, the transport layer does a similar process passing the information to the skeleton**
  - **The skeleton opens a stream and deserialize el object**
    - The skeleton makes the invocation at the server and again the same step that in the client are achieved
  - **The answers is returned to the client**

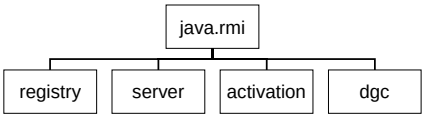
RMI packages



registry: A registry is a remote object that maps names to remote objects. A server registers its remote objects with the registry so that they can be looked up. When an object wants to invoke a method on a remote object, it must first lookup the remote object using its name. The registry returns to the calling object a reference to the remote object, using which a remote method can be invoked.

server: Provides classes and interfaces for supporting the server side of RMI. A group of classes are used by the stubs and skeletons generated by the rmic stub compiler. Another group of classes implements the RMI Transport protocol and HTTP tunneling.

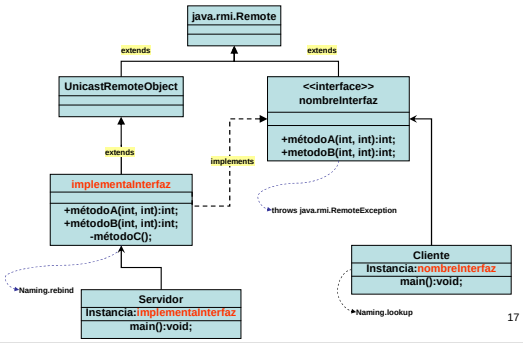
RMI packages



activation: Provides support for RMI Object Activation. A remote object's reference can be made "persistent" and later activated into a "live" object using the RMI activation mechanism.

dgc: Provides classes and interface for RMI distributed garbage-collection (DGC). When the RMI server returns an object to its client (caller of the remote method), it tracks the remote object's usage in the client. When there are no more references to the remote object on the client, or if the reference's "lease" expires and not renewed, the server garbage-collects the remote object.

The Application



General description

- ***The Remote interface serves to identify interfaces whose methods may be invoked from a non-local virtual machine.***
- ***The Naming class provides methods for storing and obtaining references to remote objects in a remote object registry.***
  - **bind**
  - **rebind**
  - **lookup**

rebind method

- ***Rebind let's "to re-link" the specified name with a new object***
  - **public static void rebind(String name, Remote obj) throws RemoteException, MalformedURLException**
- ***You can use like:***

```
implementaInterfaz objeto = new implementaInterfaz();
rebind("rmi://host/serviceName",objeto);
```

- ***The string name takes the form of a URL:***

rmi://host:port/serviceName

lookup method

- ***Returns a reference, a stub, for the remote object associated with the specified name.***
  - **public static Remote lookup(String name) throws NotBoundException, MalformedURLException, RemoteException**
- ***You can use like:***

```
nombreInterfaz objeto =(nombreInterfaz) Naming.lookup
        ("rmi://localhost/nameService");
int resultado = objeto.MétodoA(1,2);
```


RemoteException

- ***A RemoteException is the common superclass for a number of communication-related exceptions that may occur during the execution of a remote method call.***
- ***Each method of a remote interface, an interface that extends java.rmi.Remote, must list RemoteException in its throws clause.***

1

Defining the interface

```
import java.rmi.*;
public interface Calculadora extends Remote
{
  // Se declaran todos los métodos accesibles
  public double multi(double n1, double n2) throws
      remoteException;
  public double suma(double n1, double n2) throws
      remoteException;
  ...
}
```

primitives or implements Serializable or Remote

2

Implementing interface

```
import java.rmi.*;
import java.rmi.server.*;
public class CalculadoraImplementada extends UnicastRemoteObject
       implements Calculadora
{
   // Se declaran todos los métodos accesibles
   public double multi(double n1, double n2) throws
   remoteException
   {   // se implementa el método multi, que multiplica dos
   números
   }
   public double suma(double n1, double n2) throws remoteException
   {   // se implementa el método suma, que suma dos números
   }
   ...
}
```


5

Using remote interface

```
import java.rmi.*;
public class CalculadoraCliente
{
   public static void main (String arg[])
   {
       Calculadora calc = new BuscarRegistro("cliente",
   "objeto");
       calc.multi(23.2452324, 123.123121231);
   }

}
```

This should be the method **lookup** from RMI

Example

25

Reference

- ***William Grosso, Java RMI, O'Reilly, 2001***
- ***JavaTM 2 Platform Std. Ed. v1.4.2, Sun***

26