Slide 1

Centre International de Mathématiques Pures et Appliquées
International Center for Pure and Applied Mathematics

A CIMPA-UNSA-INRIA-UNESCO-VIETNAM SCHOOL ON

Objects and Network

Part : Distributed Objects Courses

Emmanuel Chailloux

`Emmanuel.Chailloux@lip6.fr`

`http://www-spi.lip6.fr/~emmanuel`

Slide 2

SUMMARY

Distributed Objects:

- RMI
- Distributed GC
- CORBA

Slide 3

Distributed Objects

Copying remote objects:

sockets + persistence $\Rightarrow$ transport of objects by copy (and creation of new instances)

Sharing remote objects:

References to a remote object from different network places to the **same** object
to invoke its methods and/or to modify its instance variables.

GOALS:

- To simplify the protocol.
  The object interface becomes the new protocol.
- To distribute applications.

Slide 4

Implementation difficulties

- Referential transparency
- Garbage Collector
- Typechecking for copied objects
- Remote exceptions

jdk 1.1 proposes a basic mechanism, called RMI (*Remote Method Invocation*) to manipulate remote objects.

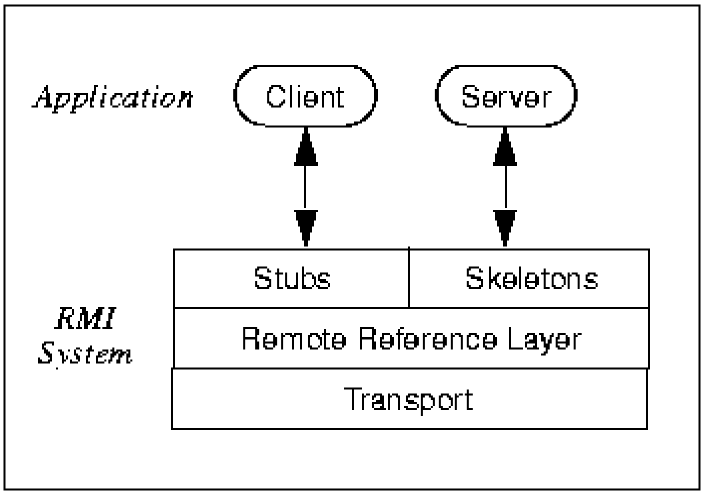
Slide 5

RMI

- remote objects : running in another Java abstract machine
- method call : to keep the same object model
  a remote method call has the **same** syntax!!!
- using remote object : Just to need to implement the `Remote` interface.
  Parameters and results are given by copy (`Serializable`)
- naming service : localization by URL
- communication : rmi protocol

Slide 6

Architecture



Slide 7

Server

The server part is more complicated for the following services :

- GC for distributed objects
- replication for remote objects
- activation for persistent objects

Slide 8

Packages

packages are :

- `java.rmi` : for client side
- `java.rmi.server` : for server side
- `java.rmi.registry` : to manage `Registry`
- `java.rmi.dgc` : for the distributed GC

The remote objects interface extends the `Remote` interface and adds some methods, which can throw `RemoteException`.

Slide 9

Packages description

in java.rmi

- `Remote` Interface : to identify remote objects (empty)
- `Naming` class : main `static` methods
  void `bind`(String, Remote)
  void `rebind`(String,Remote)
  String[] list(String)
  Remote `lookup`(String)
  void `ubind`(String)
  They can throw a lot of exceptions.
- `RMISecurityManager` class: default security
- a lot of exceptions : `RemoteException`,...

in java.rmi.server

- `UnicastRemoteObject` class : support for point-to-point active object references

Slide 10

Example : Remote Points

Interface:

```
import java.rmi.*;
public interface PointRMI extends Remote {

  void moveto (int a, int b) throws RemoteException;

  void rmoveto (int dx, int dy) throws RemoteException;

  void display() throws RemoteException;

  double distance() throws RemoteException;

}
```

Slide 11

Remote interfaces implementation

Interface implementation:

```
  import java.rmi.*;
  import java.rmi.server.UnicastRemoteObject;

public class PointD extends UnicastRemoteObject
                     implements PointRMI {

  int x,y;
  PointD(int a, int b) throws RemoteException {x=a;y=b;}
  PointD() throws RemoteException {x=0;y=0;}

  public void moveto (int a, int b)
    throws RemoteException
  { x=a; y=b;}

  public void rmoveto (int dx, int dy)
    throws RemoteException
  { x = x + dx; y = y + dy;}

  public void display()
   throws RemoteException
  { System.out.println("(" + x + "," + y + ")");}

  public double distance()
    throws RemoteException
  { return Math.sqrt(x*x+y*y);}
}
```

Slide 12

Compilation

- interface and classes : `javac` PointRMI.java PointD.java
- *stubs* and *skeletons* : `rmic` PointD

`rmic PointD` will create the files : `PointD_Stub.class` (for clients) and `PointD_Skel.class` (for servers).

Slide 13

Creation and registration

The points server will create `PointD` instances and will registry them (`rebind`) to the `rmiregistry` daemon which manages the rmi protocol.

```
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;

public class Creation {

  public static void main (String args[]) {
    System.setSecurityManager(new RMISecurityManager());
    try {
      PointD p0 = new PointD();
      PointD p1 = new PointD(3,4);
      Naming.rebind("//etretat.lip6.fr/point0",p0);
      Naming.rebind("//etretat.lip6.fr/point1",p1);
      System.out.println("Distributed Objects 'p0'" +
           " and 'p1' are registried");
    }
    catch (Exception e) { e.printStackTrace();
    }
  }
}
```

Slide 14

Server running

Naming service:

```
$ rmiregistry&
```

Server:

```
$  java Creation
Distributed Objects 'p0'
and 'p1' are registried
```

Slide 15

A client

```
import java.rmi.*;
public class ClientP {
  public static void main( String argv[]) {
    String url0="rmi://etretat.lip6.fr/point0";
    String url1="rmi://etretat.lip6.fr/point1";
 try {
   PointRMI p0 = (PointRMI)Naming.lookup(url0);
   PointRMI p1 = (PointRMI)Naming.lookup(url1);
   p0.display(); p1.display();
   p0.rmoveto(7,12);
   p1.rmoveto(5,6);
   p0.display(); p1.display();
   if (p0.distance() == p1.distance())
     System.out.println("same distance!!!");
   else
       System.out.println("different distance");
   }
 catch (Exception e) {
     System.err.println("exception : " +
                         e.getMessage());
     e.printStackTrace();
} } }
```

Slide 16

Client running

first execution:

```
(0,0)
(3,4)
(7,12)
(8,10)
different distance
```

second execution:

```
(7,12)
(8,10)
(14,24)
(13,16)
different distance
```

Slide 17

Concurrent calls

may or not may execute in a separate thread

- **same** client virtual machine :
  no guarantees to execute method invocation in different threads
- **different** client virtual machines :
  calls will execute in different threads

To use `synchronized` methods for the remote objects!!!

Slide 18

Concurrent Clients

```
etr: java ClientP C1 & \
     java ClientP C2 & java ClientP C3
[2] 32212
[3] 32213
C3.p0(112,192)
C1.p0(112,192)
C1.p1(83,100)
C1.p0(119,204)
C1.p1(88,106)
C3.p1(88,106)
C3.p0(126,216)
C3.p1(93,112)
C1 : different distance
[2]    Done                    java ClientP C1
C3 : different distance
C2.p0(126,216)
C2.p1(93,112)
C2.p0(133,228)
C2.p1(98,118)
C2 : different distance
[3]    Done                    java ClientP C2
```

Slide 19

Exceptions

Different cases:

- daemon doesn't run

```
exception : Connection refused to host;
...
```

- object doesn't registry

```
exception : point0
java.rmi.NotBoundException: point0
```

Slide 20

Communication port

Default port for the rmi service is : 1099,
but the rmiregistry can be running on another port :

```
rmiregistry 2000&
```

this new port must be used for the URL :

```
//youpou.lip6.fr:2000
```

Slide 21

Distributed GC

What is the problem?:

cleans the external reference to an object (in the server side)

- if a client doesn't use it
- need to be fault tolerant
  - when a client disappears (long time)
  - if a client disappears and appears again

Slide 22

Java solution

in `java.rmi.dgc` package :

A reference to a remote object is leased for a period of time by the client holding the reference.

- The lease period starts when the dirty call is received.
- It is the client's responsibility to renew the leases, by making additional dirty calls, on the remote references it holds before such leases expire.
- If the client does not renew the lease before it expires, the distributed garbage collector assumes that the remote object is no longer referenced by that client.

Slide 23

Interface DGC

```
package java.rmi.dgc;

import java.rmi.server.ObjID;

public interface DGC extends java.rmi.Remote {

        Lease dirty(ObjID[] ids, long sequenceNum, Lease lease)
                throws java.rmi.RemoteException;

        void clean(ObjID[] ids, long seqNum, VMID vmid,
boolean strong)
                throws java.rmi.RemoteException;
}
```

For each exported remote object in the local virtual machine, the garbage collector maintains a reference list – a list of clients that hold references to it.

The clean call removes the vmid from the reference list of each remote object indicated in *ids*.

Slide 24

CORBA

Common Object Request Broker Architecture : standard (OMG - 1990) allows :

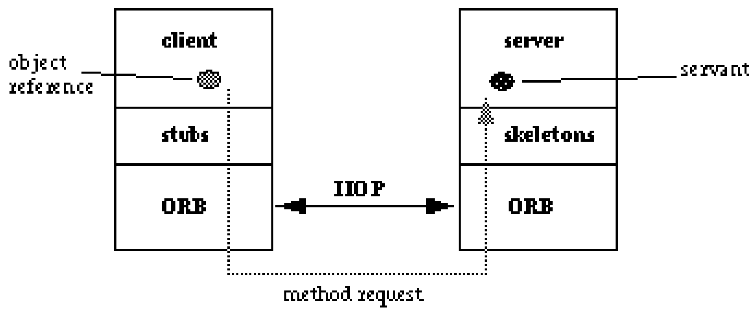
- to be independent from languages
- to have more complete registrying services
- to be independent from constructors
  a lot of ORBs (including free ORBs)

To be independent :

- from constructors, CORBA specifies IIOP (open protocol)
- from languages : CORBA specifies the IDL language.

Slide 25

Architecture



Slide 26

IDL

Interface Definition Language allows to describe visible *methods* and *fields* from objects.
Accept simple or multiple interface inheritance.

Point.idl:

```
interface Point {
  void moveto(in int a, in int b);
  void rmoveto(in int dx, in int dy);
  void display();
  double distance();
}
```

From this description, the IDL compiler (given with ORBs), generates *stub* and *skeletons* fro the server, and that for different languages (Smalltalk, C++, Java ...)

Slide 27

CORBA and Java

Java IDL: classes, libraries and tools to use CORBA object from Java, contains :

- an ORB
- a naming service (`tnameserv`)
- `idltojava` compiler

Java IDL conforms to IIOP 1.0 (Internet InterORB Protocol).

Slide 28

Java IDL : development process

1. To define the remote interface : Point.idl
2. To compile the remote interface, which generates Java version of the interface, and class code files for the *stubs* and *skeletons*
3. To implement the server, by using *skeletons* to implement remote interface, server code includes the ORB starting and waits for invocation.
4. To implement the client, by using *stubs* : to start its ORB
   look up the server using the name service, to obtain a reference to remote object and to call its method.
5. To start the applications
   to start the name service, then to start the server, then to run the clients

Slide 29

RMI and CORBA

same ideas (RMI technology was inspired by CORBA) :

- interfaces definition
- stubs and skeletons elaboration

CORBA is a standard (OMG = 500 companies), RMI not
CORBA gives a better registration services

Slide 30

RMI *vs* CORBA

	RMI	CORBA
version	JDK1.1.6	JDK 1.2 **Beta 4**
communication	Java machines	IIOP
services	minimal	better
performances	bad	variable
DGC	yes	no
cost	free licence	javatoidl ??
old programs	not	possible
usage	simpler	more complex