

RPC & RMI

Distributed Software Systems

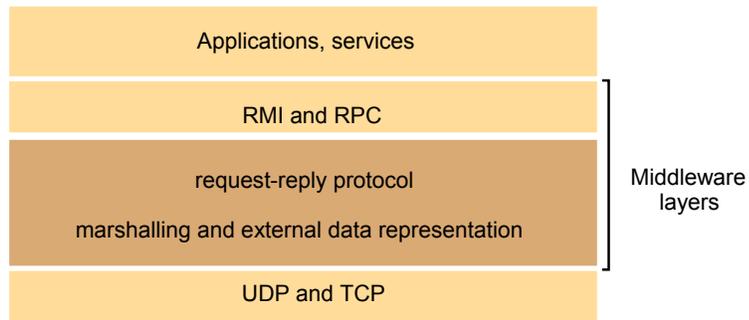
RMI 1

Motivation

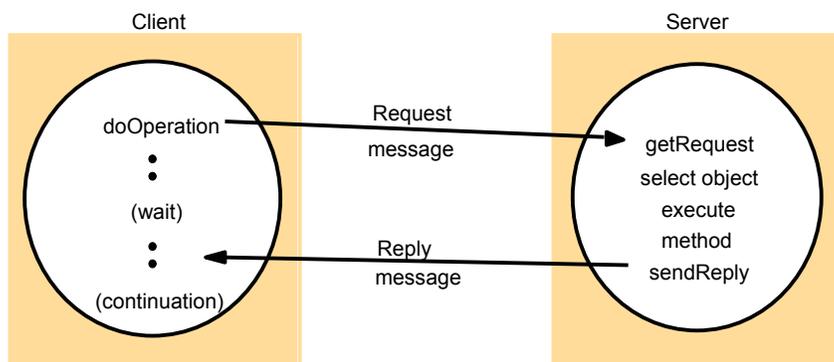
- **Sockets API** ≡ send & recv calls ≡ I/O
- **Remote Procedure Calls (RPC)**
 - Goal: to provide a procedural interface for distributed (i.e., remote) services
 - To make distributed nature of service transparent to the programmer
 - No longer considered a good thing
- **Remote Method Invocation (RMI)**
 - RPC + Object Orientation
 - Allows objects living in one process to invoke methods of an object living in another process

RMI 2

Middleware layers

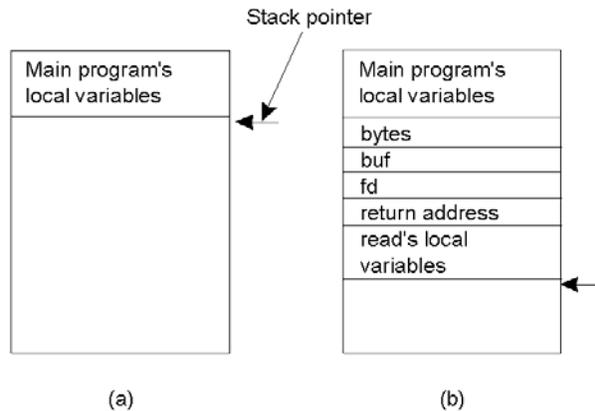


Request-reply communication



Conventional Procedure Call

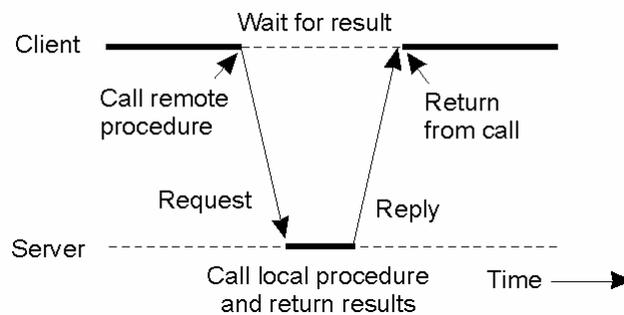
- a) Parameter passing in a local procedure call: the stack before the call to `read(fd,buf,bytes)`
- b) The stack while the called procedure is active



RMI 5

Remote Procedure Call

- Principle of RPC between a client and server program.



RMI 6

Remote Procedure Calls

- ❑ Remote procedure call (RPC) abstracts procedure calls between processes on networked systems.
- ❑ **Stubs** - client-side proxy for the actual procedure on the server.
- ❑ The **client-side stub** locates the server and *marshalls* the parameters.
- ❑ The **server-side stub** receives this message, unpacks the marshaled parameters, and performs the procedure on the server.

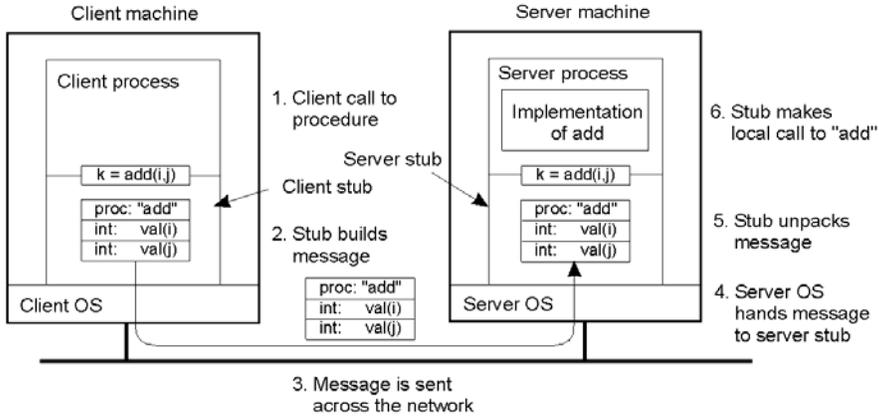
RMI 7

Steps of a Remote Procedure Call

1. Client procedure calls client stub in normal way
2. Client stub builds message, calls local OS
3. Client's OS sends message to remote OS
4. Remote OS gives message to server stub
5. Server stub unpacks parameters, calls server
6. Server does work, returns result to the stub
7. Server stub packs it in message, calls local OS
8. Server's OS sends message to client's OS
9. Client's OS gives message to client stub
10. Stub unpacks result, returns to client

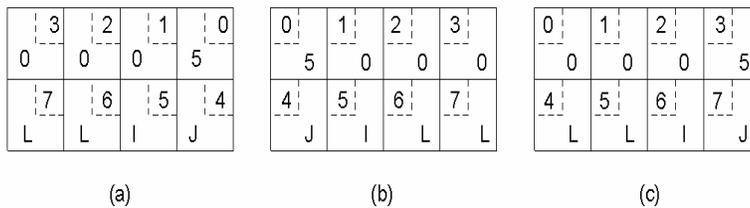
RMI 8

Passing Value Parameters (1)



RMI 9

Passing Value Parameters (2)



- Original message on the Pentium
- The message after receipt on the SPARC
- The message after being inverted. The little numbers in boxes indicate the address of each byte

RMI 10

Parameter Specification and Stub Generation

- a) A procedure
- b) The corresponding message.

```
foobar( char x; float y; int z[5] )  
{  
  ....  
}
```

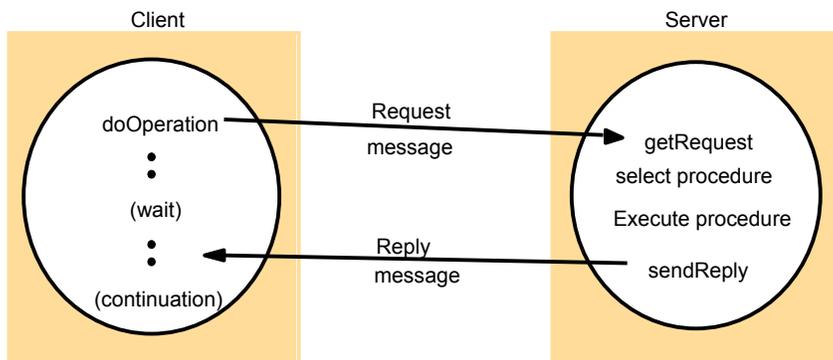
(a)

foobar's local variables	
	x
y	
5	
z[0]	
z[1]	
z[2]	
z[3]	
z[4]	

(b)

RMI 11

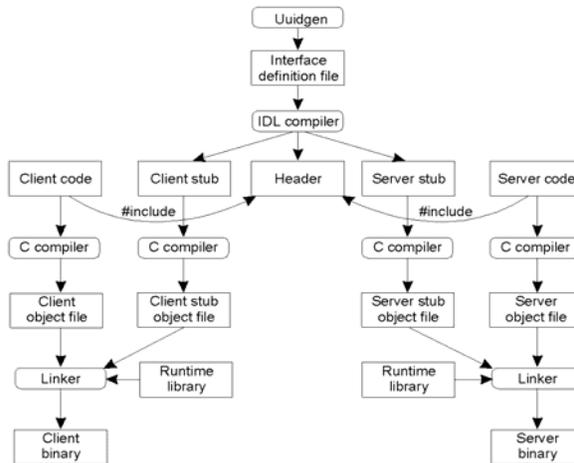
Request-reply communication



RMI 12

Writing a Client and a Server

The steps in writing a client and a server in DCE RPC
(SUN RPC is similar)



RMI 13

Files interface in Sun XDR

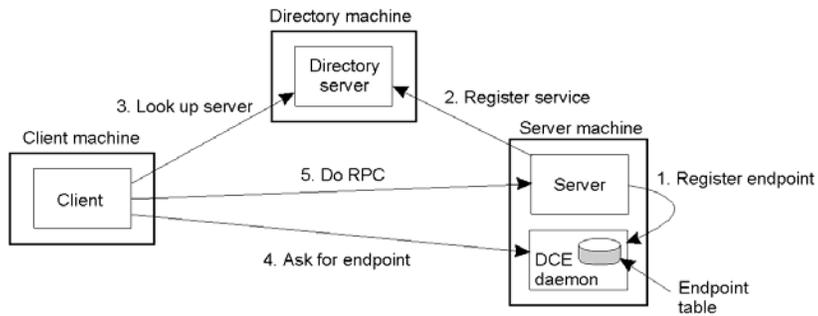
```
const MAX = 1000;
typedef int FileIdentifier;
typedef int FilePointer;
typedef int Length;
struct Data {
    int length;
    char buffer[MAX];
};
struct writeargs {
    FileIdentifier f;
    FilePointer position;
    Data data;
};
```

```
struct readargs {
    FileIdentifier f;
    FilePointer position;
    Length length;
};

program FILEREADWRITE {
    version VERSION {
        void WRITE(writeargs)=1;    1
        Data READ(readargs)=2;     2
    }=2;
} = 9999;
```

RMI 14

Binding a Client to a Server



RMI 15

RMI

- RMI = RPC + Object-orientation
 - Java RMI
 - CORBA
 - Middleware that is language-independent
 - Microsoft DCOM/COM+
 - SOAP
 - RMI on top of HTTP

RMI 16

Interfaces in distributed systems

- Programs organized as a set of **modules** that communicate with one another via procedure calls/method invocations
- Explicit **interfaces** defined for each module in order to control interactions between modules
- In distributed systems, modules can be in different processes
- A **remote interface** specifies the methods of an object that are available for invocation by objects in other processes defining the **types of the input and output arguments** of each of them

RMI 17

CORBA IDL example

```
// In file 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();  
};
```

RMI 18

Object model

- ❑ Object references
 - Objects accessed via object references
 - Object references can be assigned to variables, passed as arguments and returned as results
- ❑ Interfaces
 - Provides a signature of a set of methods (types of arguments, return values and exceptions) without specifying their implementations
- ❑ Actions (invocations)
- ❑ Exceptions
- ❑ Garbage Collection

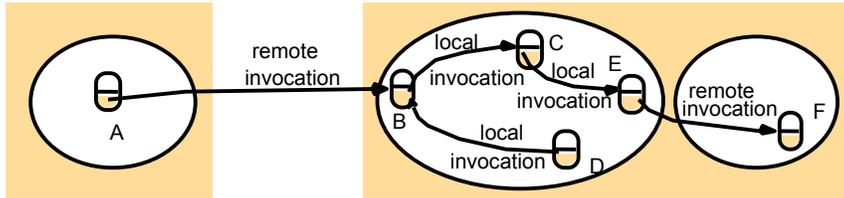
RMI 19

Distributed Objects

- ❑ Remote object references
 - An identifier that can be used throughout a distributed system to refer to a particular remote object
- ❑ Remote interfaces
 - CORBA provides an interface definition language (IDL) for specifying a remote interface
 - JAVA RMI: Java interface that extends *Remote* interface
- ❑ Actions: remote invocations
- ❑ Remote Exceptions may arise for reasons such as partial failure or message loss
- ❑ Distributed Garbage Collection: cooperation between local garbage collectors needed

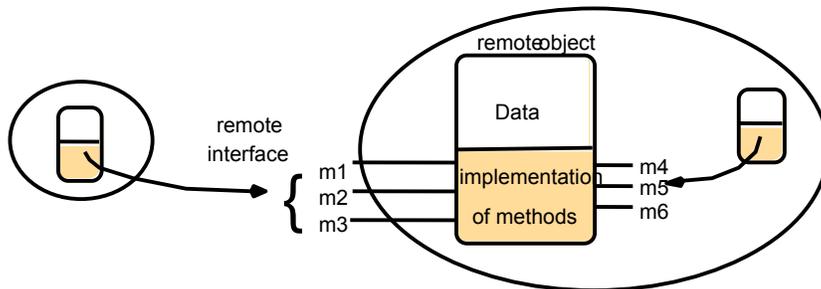
RMI 20

Remote and local method invocations



RMI 21

A remote object and its remote interface



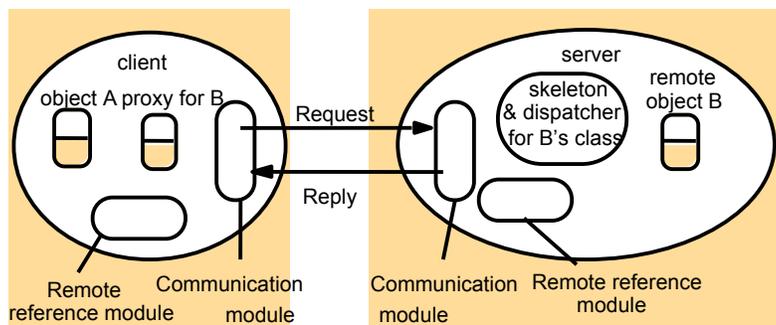
RMI 22

RMI Programming

- RMI software
 - Generated by IDL compiler
 - Proxy
 - Behaves like remote object to clients (invoker)
 - Marshals arguments, forwards message to remote object, unmarshals results, returns results to client
 - Skeleton
 - Server side stub;
 - Unmarshals arguments, invokes method, marshals results and sends to sending proxy's method
 - Dispatcher
 - Receives the request message from communication module, passes on the message to the appropriate method in the skeleton
- Server and Client programs

RMI 23

The role of proxy and skeleton in remote method invocation



RMI 24

RMI Programming

- Binder
 - Client programs need a means of obtaining a remote object reference
 - Binder is a service that maintains a mapping from textual names to remote object references
 - Servers need to register the services they are exporting with the binder
 - Java RMIregistry, CORBA Naming service
- Server threads
 - Several choices: thread per object, thread per invocation
 - Remote method invocations must allow for concurrent execution

RMI 25

RMI systems

- CORBA - language independent
- DCOM - Microsoft
- Java RMI
- SOAP (Simple Object Access Protocol)
 - HTTP is request-reply protocol
 - XML for data representation

RMI 26

Java RMI

□ Features

- Integrated with Java language + libraries
 - Security, write once run anywhere, multithreaded
 - Object orientation
- Can pass "behavior"
 - Mobile code
 - Not possible in CORBA, traditional RPC systems
- Distributed Garbage Collection
- *Remoteness of objects intentionally not transparent*

RMI 27

Remote Interfaces, Objects, and Methods

- Objects become remote by implementing a remote interface
 - A remote interface extends the interface `java.rmi.Remote`
 - Each method of the interface declares `java.rmi.RemoteException` in its throws clause in addition to any application-specific clauses

RMI 28

Creating distributed applications using RMI

1. Define the remote interfaces
2. Implement the remote objects
3. Implement the client (can be done anytime after remote interfaces have been defined)
4. Register the remote object in the name server registry
5. Generate the stub and client using *rmic*
6. Start the registry
7. Start the server
8. Run the client

RMI 29

Java Remote interfaces *Shape* and *ShapeList*

```
import java.rmi.*;
import java.util.Vector;
public interface Shape extends Remote {
    int getVersion() throws RemoteException;
    GraphicalObject getAllState() throws RemoteException;    1
}
public interface ShapeList extends Remote {
    Shape newShape(GraphicalObject g) throws RemoteException; 2
    Vector allShapes() throws RemoteException;
    int getVersion() throws RemoteException;
}
```

RMI 30

The Naming class of Java RMIregistry

void rebind (String name, Remote obj)

This method is used by a server to register the identifier of a remote object by name, as shown in Figure 15.13, line 3.

void bind (String name, Remote obj)

This method can alternatively be used by a server to register a remote object by name, but if the name is already bound to a remote object reference an exception is thrown.

void unbind (String name, Remote obj)

This method removes a binding.

Remote lookup (String name)

This method is used by clients to look up a remote object by name, as shown in Figure 15.15 line 1. A remote object reference is returned.

String [] list()

This method returns an array of Strings containing the names bound in the registry.

RMI 31

Java class ShapeListServer with main method

```
import java.rmi.*;
public class ShapeListServer{
    public static void main(String args[]){
        System.setSecurityManager(new RMISecurityManager());
        try{
            ShapeList aShapeList = new ShapeListServant();           1
            Naming.rebind("Shape List", aShapeList);                 2
            System.out.println("ShapeList server ready");
        }catch(Exception e) {
            System.out.println("ShapeList server main " + e.getMessage());}
    }
}
```

RMI 32

Java class *ShapeListServant* implements interface *ShapeList*

```
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;
import java.util.Vector;
public class ShapeListServant extends UnicastRemoteObject implements ShapeList {
    private Vector theList;           // contains the list of Shapes      1
    private int version;
    public ShapeListServant() throws RemoteException {...}
    public Shape newShape(GraphicalObject g) throws RemoteException {    2
        version++;
        Shape s = new ShapeServant( g, version);                          3
        theList.addElement(s);
        return s;
    }
    public Vector allShapes() throws RemoteException {...}
    public int getVersion() throws RemoteException { ... }
}
```

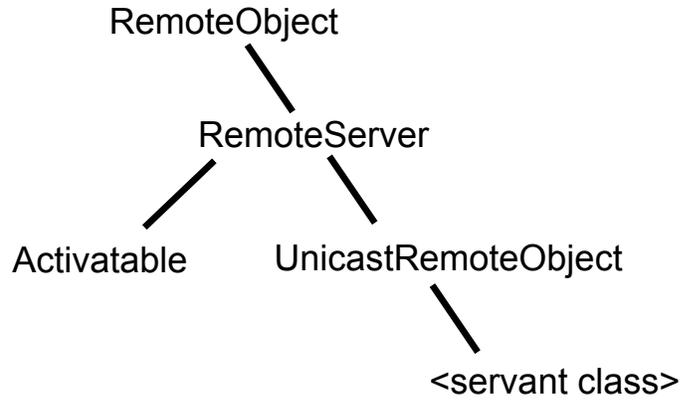
RMI 33

Java client of *ShapeList*

```
import java.rmi.*;
import java.rmi.server.*;
import java.util.Vector;
public class ShapeListClient{
    public static void main(String args[]){
        System.setSecurityManager(new RMISecurityManager());
        ShapeList aShapeList = null;
        try{
            aShapeList = (ShapeList) Naming.lookup("//bruno.ShapeList");    1
            Vector sList = aShapeList.allShapes();                          2
        } catch(RemoteException e) {System.out.println(e.getMessage());}
        } catch(Exception e) {System.out.println("Client: " + e.getMessage());}
    }
}
```

RMI 34

Classes supporting Java RMI



RMI 35

Advanced Techniques

- Passing behavior
 - See Java RMI tutorial track example
- Callbacks
- Activation

RMI 36