Distributed Systems

Lesson 3
Introduction to RMI in Java

University of New York in Tirana
Master of Science in Computer Science
Prof. Dr. Marenglen Biba

Lesson 3 – Lab Session

Distributed Objects

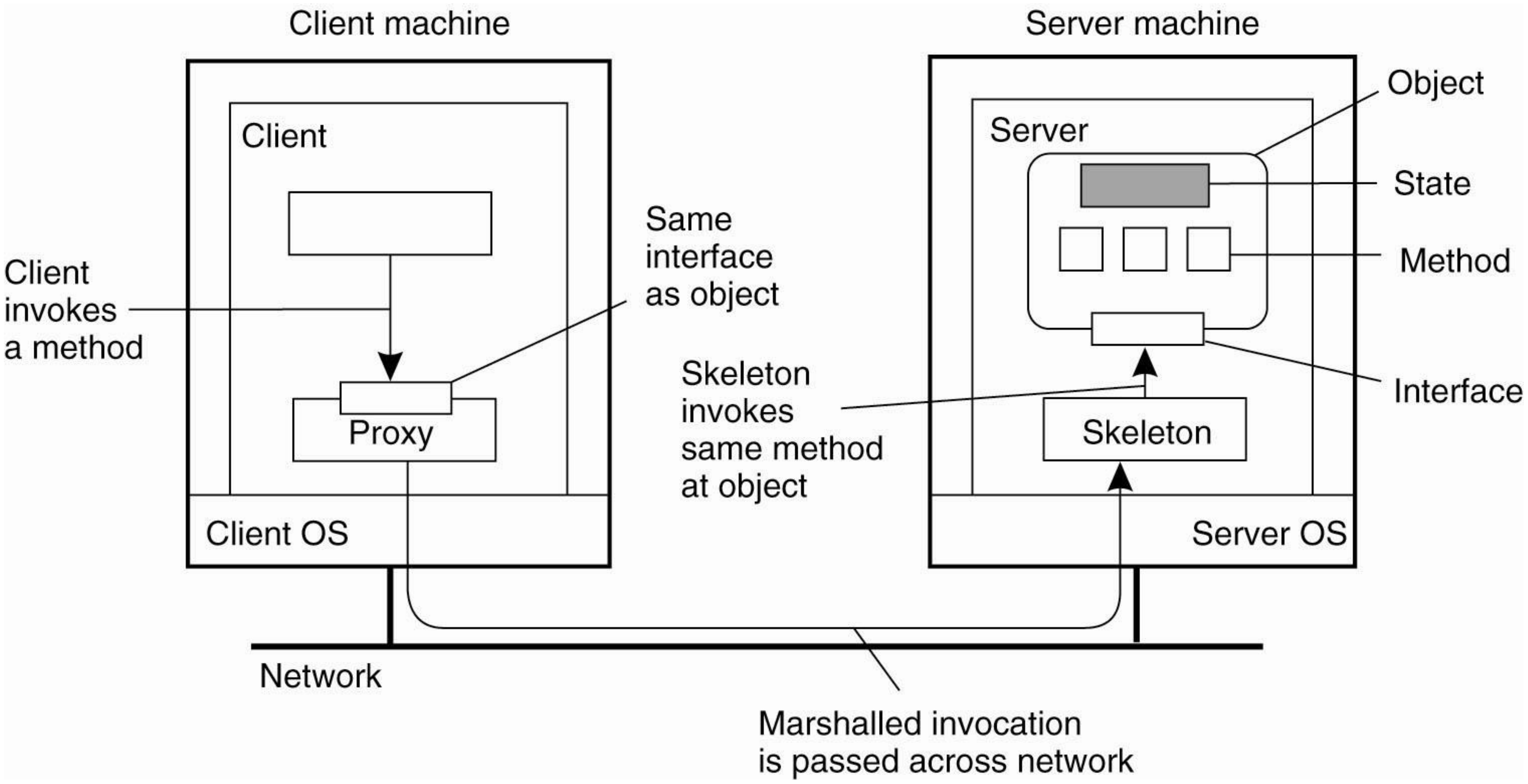


Figure 10-1. Common organization of a remote object with client-side proxy.

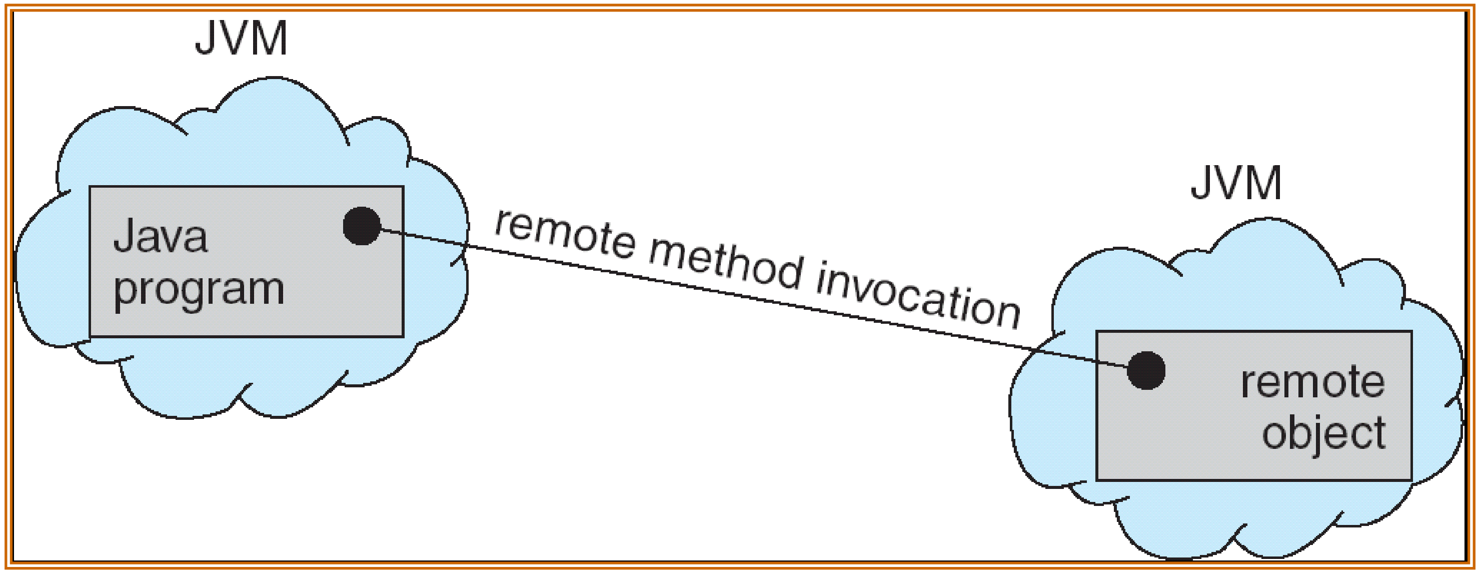
Communication

- RPC – Remote Procedure Call
- RMI – Remote Method Invocation

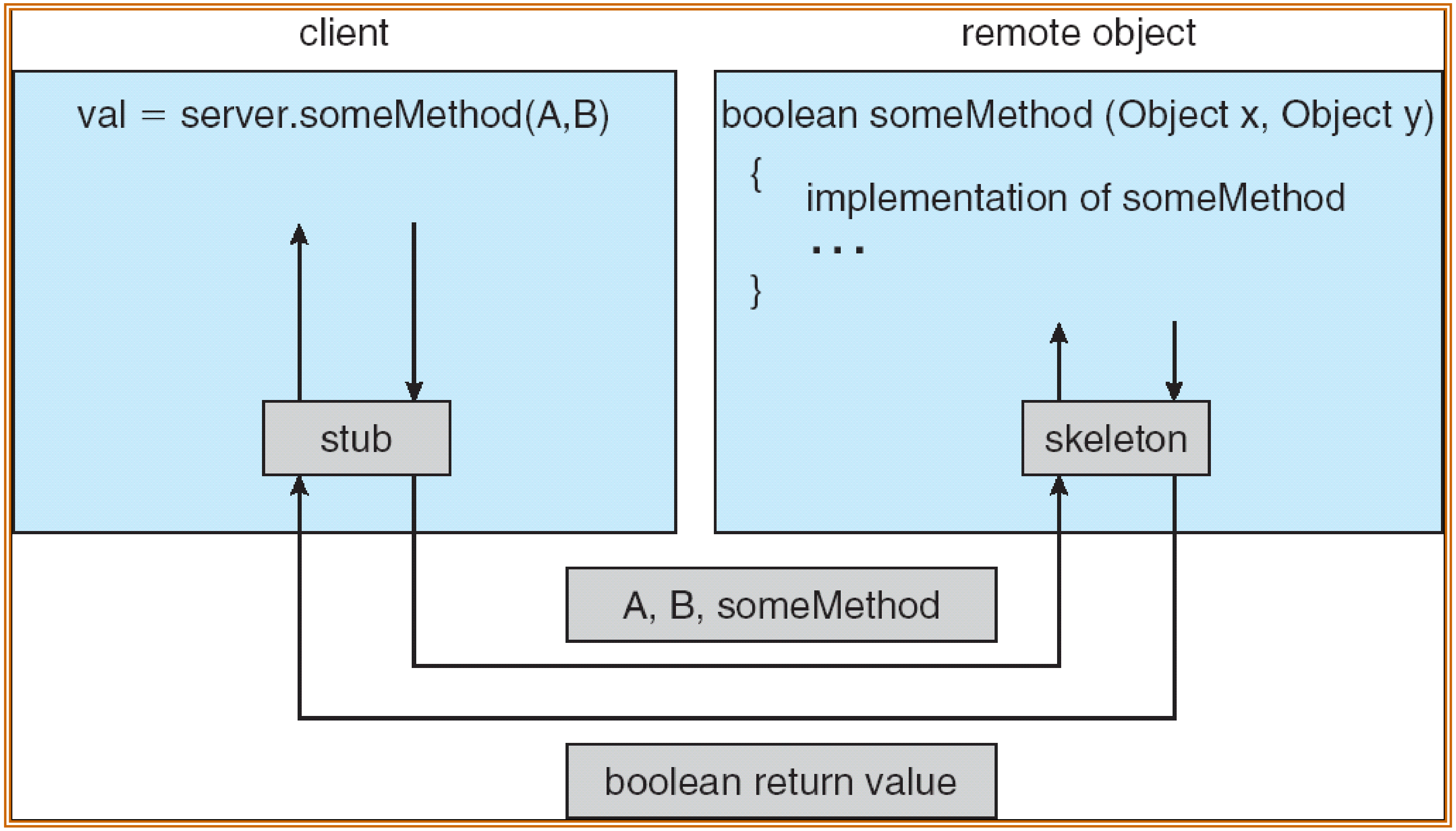
- RMI, is very similar to an RPC when it comes to issues such as marshaling and parameter passing.
- An essential difference between an RMI and an RPC is that RMIs generally support system-wide object references

Remote Method Invocation

- Remote Method Invocation (RMI) is a Java mechanism similar to RPCs.
- RMI allows a Java program on one machine to invoke a method on a remote object.

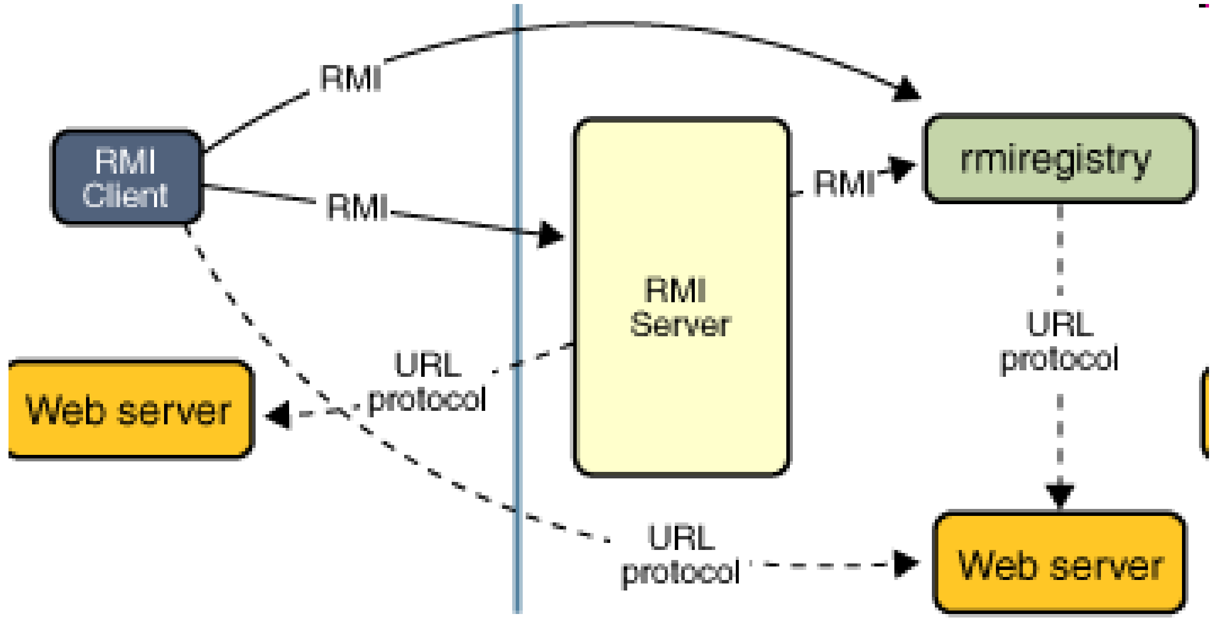


Marshalling Parameters


client
remote object
val = server.someMethod(A,B)
boolean someMethod (Object x, Object y)
{
implementation of someMethod
. . .
}
stub
skeleton
A, B, someMethod
boolean return value

RMI

- RMI applications often comprise two separate programs, a server and a client.
- A typical server program:
  - creates some remote objects,
  - makes references to these objects accessible
  - waits for clients to invoke methods on these objects.
- A typical client program:
  - obtains a remote reference to one or more remote objects on a server
  - then invokes methods on them.
- RMI provides the mechanism by which the server and the client communicate and pass information back and forth.
- Such an application is sometimes referred to as a *distributed object application*.



- The illustration depicts an RMI distributed application that uses the RMI registry to obtain a reference to a remote object.
- The server calls the registry to associate (or bind) a name with a remote object.
- The client looks up the remote object by its name in the server's registry and then invokes a method on it.
- The illustration also shows that the RMI system uses an existing web server to load class definitions, from server to client and from client to server, for objects when needed.

Remote Interfaces, Objects, and Methods

- Like any other Java application, a distributed application built by using Java RMI is made up of interfaces and classes.

- The interfaces declare methods.

- The classes implement the methods declared in the interfaces and, perhaps, declare additional methods as well.

- Objects with methods that can be invoked across Java virtual machines are called *remote objects*.

Java.rmi.Remote

- An object becomes remote by implementing a *remote interface*, which has the following characteristics:

- 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 exceptions.

Creating Distributed Applications by Using RMI

- Using RMI to develop a distributed application involves these general steps:

1. Designing and implementing the components of your distributed application.
2. Compiling sources.
3. Making classes network accessible.
4. Starting the application.

Designing and Implementing the Application Components

- First, determine your application architecture, including which components are local objects and which components are remotely accessible.
- This step includes:

1. **Defining the remote interfaces.**
2. **Implementing the remote objects.**
3. **Implementing the clients.**

Defining the remote interfaces

- A remote interface specifies the methods that can be invoked remotely by a client.
- Clients program refer to remote interfaces, not to the implementation classes of those interfaces.
- The design of such interfaces includes the determination of the types of objects that will be used as the parameters and return values for these methods.

Declaration of a remote interface

```
package compute;
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Compute extends Remote {
<T> T executeTask(Task<T> t) throws
   RemoteException; }
```

Implementing the remote objects

- Remote objects must implement one or more remote interfaces.
- The remote object class may include implementations of other interfaces and methods that are available only locally.
- If any local classes are to be used for parameters or return values of any of these methods, they must be implemented as well.

Implementing the remote objects

- In general, a class that implements a remote interface should at least do the following:
  - Declare the remote interfaces being implemented
  - Define the constructor for each remote object
  - Provide an implementation for each remote method in the remote interfaces

RMI Server

- An RMI server program needs to create the initial remote objects and *export* them to the RMI runtime, which makes them available to receive incoming remote invocations.
- This procedure should do the following:

1. Create and export one or more remote objects
2. Register at least one remote object with the RMI registry (or with another naming service, such as a service accessible through the JNDI - Java Naming and Directory Interface).

Implementing the clients

- Clients that use remote objects can be implemented at any time after the remote interfaces are defined, even after the remote objects have been deployed.

Practical Session: RMI

- Refer to the Lab manual given in class for step-by-step instructions on how to develop your RMI application.

End of Lesson 3

- Readings
  - Distributed Systems, Chapter 10
    - Sections 10.3.3 and 10.3.4

- Lab Manual on RMI given in class

- For further study, online tutorial at:
  - http://download.oracle.com/javase/tutorial/rmi/index.html