Work groups meeting

INF5040 (Open Distributed Systems)

Sabita Maharjan
sabita@ifi.uio.no

Department of Informatics
University of Oslo
August 31, 2009

Meeting Plan

- 31.08.09
  - Introduction
  - Lecture: Java Distributed Computing-RMI
- 04.09.09
  - Lecture: Java Distributed Computing-EJB
  - Lecture: JEE Application Servers-JBoss
- 07.09.09
  - Lecture on “Java Distribution Patterns”
  - Assignment 1/Group formation

Meeting Plan

- 07.09.09-05.10.09
  - Working on programming exercise
  - TA will be available during meeting time for possible questions regarding assignment 1
- 05.10.09
  - Submission of Assignment 1
  - Assignment 2/ Group formation

Meeting Plan

- 05.10.09- --.11.09
  - Working on programming exercise
  - TA will be available during meeting time for possible questions regarding assignment 2

Assignment 1

- Implementation of a simple Java distributed application (will be described 07.09.2009)
- Technologies and Tools: J2EE Specifications (Servlet, JSP, EJB, JDBC, …), Jboss, Distribution patterns
- Deadline: Monday, 05.10.2009

Assignment 2

- Will be announced on 05.10.2009.

Regarding both assignments

- 2 or 3 students: one group
- Both programming assignments must be approved to be eligible to take the final exam
- For those who took this subject last fall;
  - If you were eligible to take the exam then, i.e., you got both the programming assignment and your presentation approved, you do NOT have to do them again this semester!.

Java Distributed Computing-RMI

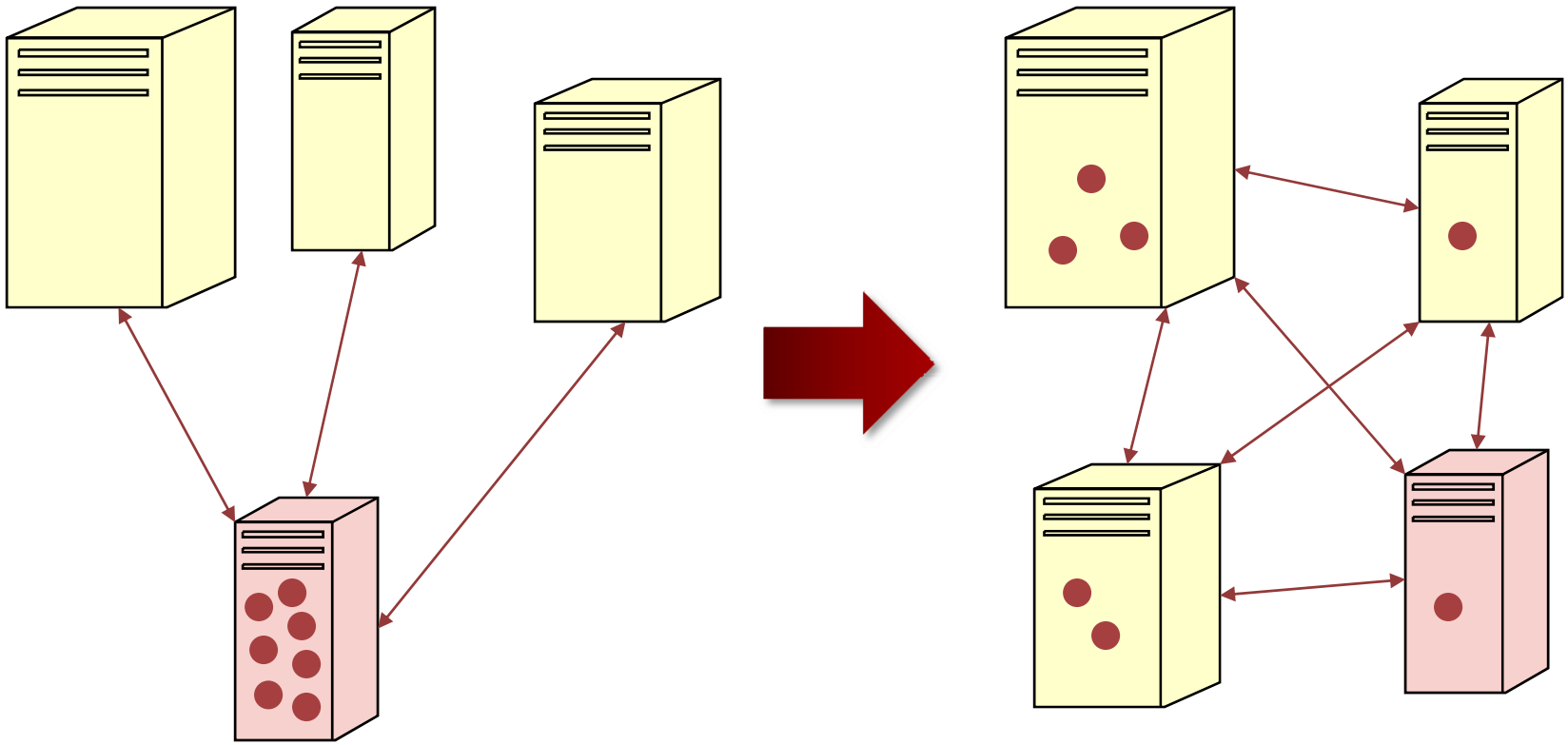
Distributed Objects

- Definition:

software modules that are designed to work **together**, but reside either in **multiple computers** connected via a network or in **different processes** inside the same computer. One object sends a **message** to another object in a **remote machine** or process to perform some task. The results are sent back to the calling object.

Distributed Objects

Why to distribute objects?

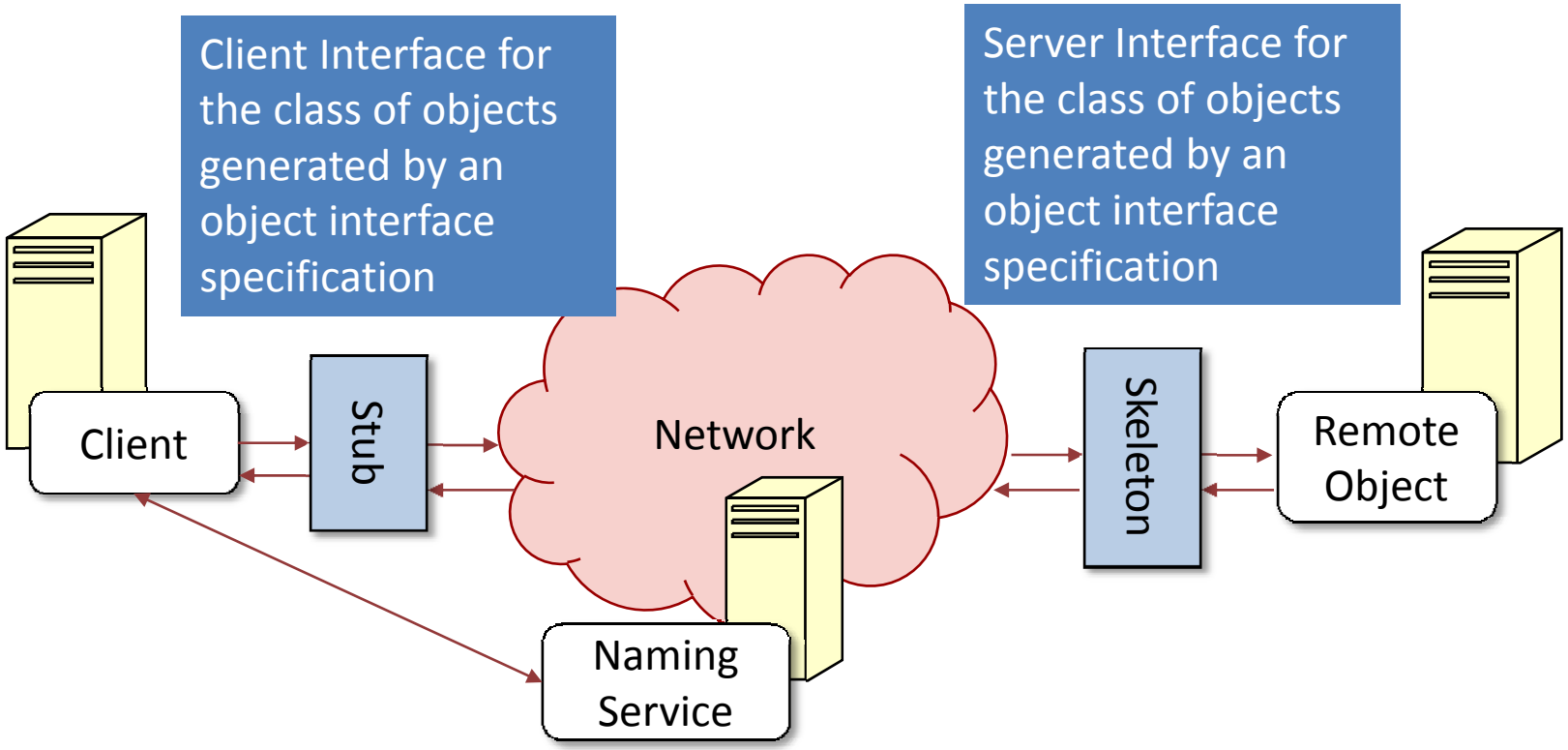


Distributed Objects

Why to distribute objects?

- Objective: To let any object reside anywhere in the network, and allow an application to interact with these objects exactly the same way as they do with a local object.
- Reduces design overhead, and makes a large system easier to maintain in the long run.

Architecture for distributed object systems



Implementations

- CORBA
  - A standard defined by the Object Management Group (**OMG**)
- DCOM
  - Microsoft solution for distribution: an extension of the Component Object Model (COM) that allows COM components to communicate across network boundaries.
- .NET Remoting
  - Part of **.Net Framework** to interact with one another across application domains
- RMI
  - **Sun** scheme for distributed objects

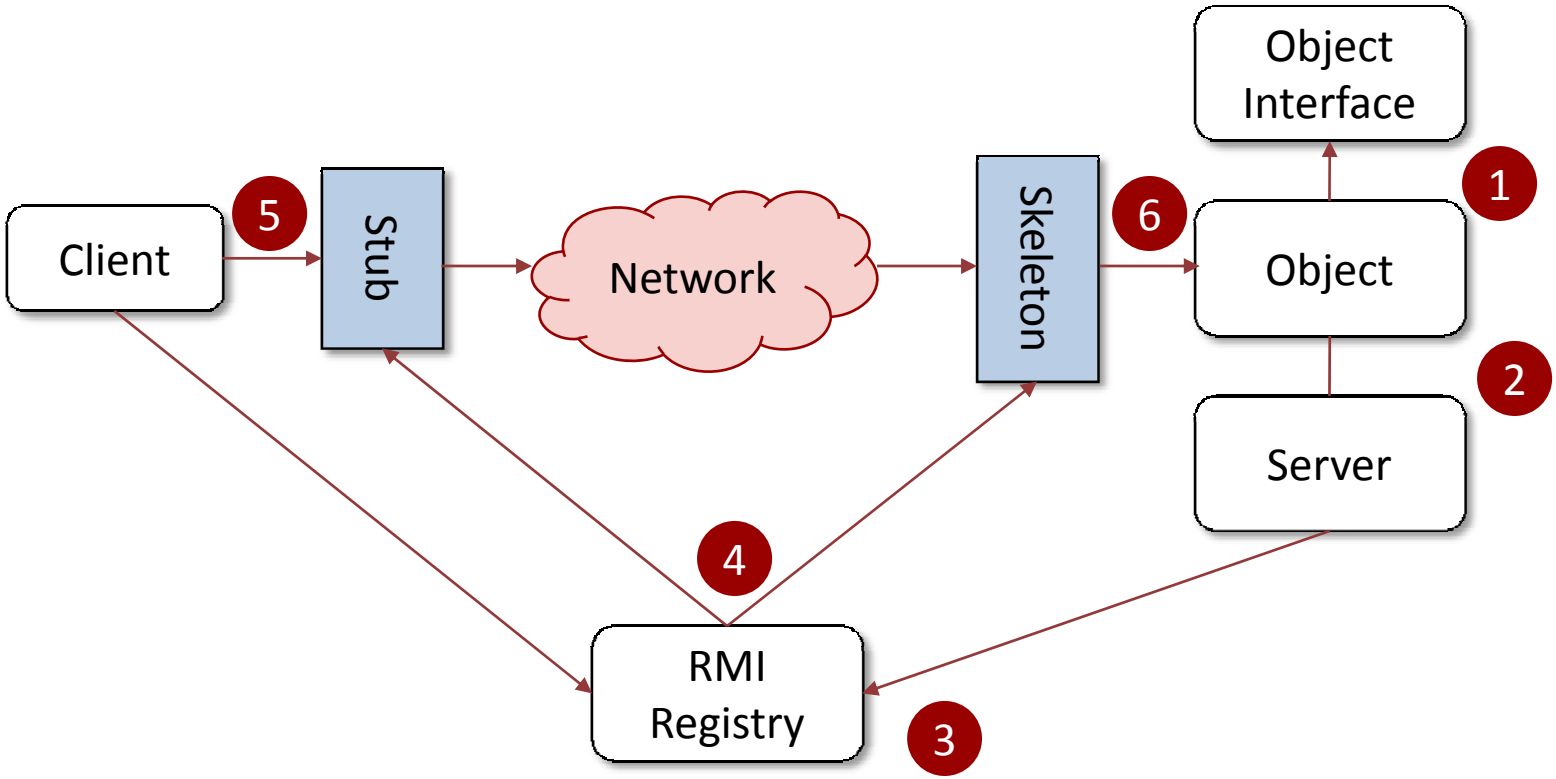
CORBA (Common Object Request Broker Adapter)

- A generic framework for building systems involving distributed objects
- Enables software components written in different computer languages and running on multiple computers to work together (**platform and language independent**)
  - The stub and skeleton need not be compiled in the same programming language
- Any agent in a CORBA system can act as both a client and a server of remote objects
- CORBA 1.1 was released in 1991

RMI (Remote Method Invocation)

- Specification by Sun for making remote method invocations on Java objects from Java clients
- Specific to Java
  - not language independent
  - provides features because it is a Java-only system
- Java Standard Edition (J2SE) comes with implementation of RMI
- Part of Java since 1.1 (1997)

RMI Scenario



RMI Deployment

1. Create remote object interface
2. Implement the interface on the server side
3. Create client stub and server skeleton for the object
    1. The interface and the server implementation are compiled using *javac* compiler
    2. The linkage is generated from the client through the RMI registry to the object implementation
4. Register remote object and client application
5. Launch server program and client program

Example: Bank Account

- Create Remote Interface:

The interface for the remote object should be written as extending the *java.rmi.Remote* interface

```
public interface BankAccount extends java.rmi.Remote {
    public Integer getBalance() throws RemoteException;
    public void makeWithdrawal(Integer) throws RemoteException;
    public void makeDeposit(Integer) throws RemoteException;
}
```

All methods in the interface must be declared as throwing the *java.rmi.RemoteException*.

The *RemoteException* is the base class for many of the exceptions that RMI defines for remote operations

Example: Bank Account

- Implement the Interface

```
public class BankAccountImpl extends UnicastRemoteObject implements
BankAccount {

    private Integer balance;

    public Integer getBalance() throws RemoteException {
            return balance;
    }
    public void makeDeposit(Integer amount)  throws RemoteException {
            balance = balance.add(amount);
    }
    public void makeWithdrawal(Integer amount) throws RemoteException {
            balance = balance.subtract(amount);
    }
}
```

The server implements the object's interface and extends the *java.rmi.server.UnicastRemoteObject* class

Stubs and Skeletons

- Generate client stub and server skeleton for the object.
  - using the RMI compiler: **rmic**
- Takes an implementation class file and generates a pair of stub and skeleton class files
  - Appends the postfixes "_Stub" and "_Skel" to the implementation class name

The RMI Registry

A simple mapping of text names to server objects (Listens for requests on a default port is 1099)

- An RMI registry can be started on a host by using *rmiregistry* command
- Object implementations can be registered using *java.rmi.Naming* class for interacting with registry
  - Servers
    - Register objects
    - *Naming.bind() or Naming.rebind()*
  - Clients
    - Lookup objects on the *Naming* interface
    - *Naming.lookup()*

Example: Bank Account

- Register Server Object

BankServer.java

```
…..
public static void main (String[] argv) {
            try {
                Naming.rebind(“Bank”,new BankAccountImpl ());
                System.out.println (“Bank Server is ready.");
            } catch (Exception e) {
                System.out.println (”Bank Server failed: " + e);
            }
}
……
```

Example: Bank Account

- Register client Application

BankClient.java

```
….
public static void main (String[] argv) {
        try {
            BankAccount bank= (BankAccount ) Naming.lookup
            ("//"+args[0]+"/Bank");
            System.out.println (bank . getBalance ());
        } catch (Exception e) {
            System.out.println ("BankClient exception: " + e);
        }
}
…..
```

Deployment Example: Bank Account

- Generating Stubs and Skeletons

```
rmic BankAccountImpl
```

- On the server host:
  1. Launch Naming service

```
rmiregistry
```

  2. Launch Server program

```
java BankServer
```

- On the client host(s):

  Launch the client program

```
java BankClient <server name–e.g.
                              kolme.ifi.uio.no>
```

RMI References

- Tutorials:
  - http://java.sun.com/docs/books/tutorial/rmi/index.html
  - http://oreilly.com/catalog/javadc/chapter/ch03.html

- Email preferences regarding group composition (if any)
  - To: sabita@ifi.uio.no