

CmpE 473 Internet Programming

Pinar Yolum
pinar.yolum@boun.edu.tr

Department of
Computer Engineering
Boğaziçi University

RMI

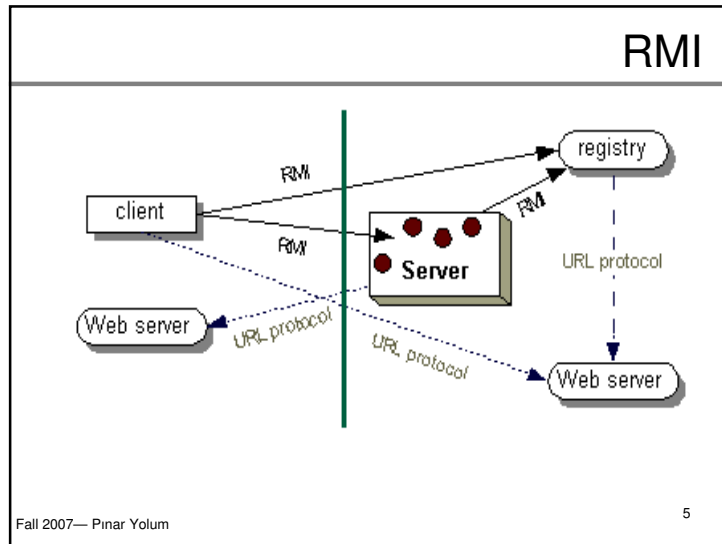
*Examples from Advanced Java: Internet
Applications, Art Gittleman*

RMI

- Remote Method Invocation
 - Allows an object to call methods of a remote object
 - Both objects should run in JVM
 - Uses sockets-based transport for communication by default
- Three steps
 - Locate the object
 - Communicate with the object
 - Load bytecodes of transferred objects

RMI Example

- Get fortunes from a fortune server
 - Fortune server registers a fortune object with registry
 - Client accesses the registry and finds a fortune object
 - Client calls the method that returns “fortune”
- Client must know which methods are available



- ## Remote Object
- Remote interface
 - Contains methods that can be called remotely by the client
 - Remote objects
 - Whose methods can be called by objects on other machines
 - Implement remote interface
 - Extends java.rmi.Remote
 - All methods in the interface must throw java.rmi.RemoteException + other specific exceptions
- Fall 2007— Pinar Yolum 6

- ## Passing Objects
- Local object
 - Passed by value
 - The object must be serializable
 - Default serialization
 - Copies all fields (except static or transients)
 - Can be overridden
 - Remote object
 - Passed by reference
 - Stub: Representative (or proxy) for the remote object
 - Stub implements the same set of remote interfaces as the remote object
 - Stub can be cast to any of the remote interfaces implemented by the remote object
- Fall 2007— Pinar Yolum 7

Sketch of the RMI Server (1)

Client

Fortune Server

```

import java.rmi.Remote;
import java.rmi.RemoteException;
public interface Fortune extends Remote {
    public Object getFortune (String when) throws
        RemoteException;
}
  
```

Because clients will call its methods

Fall 2007— Pinar Yolum 8

Sketch of the RMI Server (2)

- Fortune Server implements Fortune interface
- Only methods specified in the interface can be called by the client
- Extends `UnicastRemoteObject`
- `UnicastRemoteObject`
 - `java.rmi.server.UnicastRemoteObject`
 - Implements `Remote` and `Serializable`
 - Implementations for `equals`, `toString`, `hashCode`
- `exportObject(Remote obj, int port)`
 - Make RMI runtime aware of the object
 - Supports point-to-point remote communication
 - Load a stub class
 - Construct an instance of the stub

Fall 2007— Pinar Yolum

9

RMI Server (1)

```
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;
import java.util.Vector;
public class FortuneServer extends UnicastRemoteObject
    implements Fortune {
    public static final int SIZE = 3;
    private Vector now = new Vector(SIZE);
    private Vector later = new Vector(SIZE);
    public FortuneServer()throws RemoteException {
        now.addElement("A friend is near");
        now.addElement("Expect a call");
        now.addElement("Someone misses you");
        later.addElement("Wealth awaits -- if you desire it.");
        later.addElement("Climb the hill of effort for high grades.");
        later.addElement("The door to success is open to you."); }
    private Vector find(String when) {
        if (when.equals(Fortune.NOW)) return now;
        else return later;
    }
}
```

Fall 2007— Pinar Yolum

10

RMI Server (2)

```
public synchronized String getFortune(String when)
    throws RemoteException {
    int number = (int)(3*Math.random());
    Vector fortunes = find(when);
    return (String)fortunes.elementAt(number);
}

public static void main(String[] args) {
    System.setSecurityManager(new RMISecurityManager());
    try {
        Fortune fortune = new FortuneServer();
        String url = "rmi://" + args[0] + "/Seer"; Naming.rebind(url,fortune);
        System.out.println("Starting Fortune server");
    } catch(Exception e) { e.printStackTrace(); }
}}
```

Fall 2007— Pinar Yolum

11

RMI Client

```
import java.rmi.*;
public class FortuneClient {
    public static void main(String[] args) {
        System.setSecurityManager(new
            RMISecurityManager());
        try {
            String url = "rmi://" + args[0] + "/Seer";
            Fortune fortuneTeller = (Fortune)Naming.lookup(url);
            String fortune = fortuneTeller.getFortune(Fortune.NOW);
            System.out.println("Today's fortune is: " + fortune);
            fortune=fortuneTeller.getFortune(Fortune.LATER);
            System.out.println("Future fortune is: " + fortune);
        } catch(Exception e) { e.printStackTrace(); }
    }
}}
```

Fall 2007— Pinar Yolum

12

Stub-to-Remote

- When the client calls a method on the stub
 - Stub connects the remote JVM
 - Writes (marshals) parameters to the remote JVM
 - Reads (unmarshals) the result (return value, warning, exception)
 - Gives the result to the client

Fall 2007— Pinar Yolum

13

Security Managers

- Checks accesses to
 - System resources (e.g., local file system)
 - Privileged operations
 - Based on policy implemented
 - Methods
 - Are of type `checkXXX`; (`checkAccess`; `checkConnect`)
 - Ex: `checkWrite(String filename)`
 - Ex: `checkConnect(String host, int port)`
 - Throw an exception if there is a problem
- All RMI programs must install one
 - `java.rmi.RMI SecurityManager`
 - Extends `java.lang.SecurityManager`

Fall 2007— Pinar Yolum

14

Permissions

- Granted to codes for access
- Create a permission

```
perm = new java.io.FilePermission("/tmp/abc", "read");
```
- Grant a permission

```
grant codeBase "file:/home/sysadmin/" {
    permission java.io.FilePermission "/tmp/abc", "read";
};
```
- Stored in a `.policy` file
- Difficult to construct by hand (policytool)
- Check out the default policy
`$JAVA_HOME/lib/security/java.policy`
- `java -Djava.security.policy=policy-file MyClass`

Fall 2007— Pinar Yolum

15

Registry

- `Naming.rebind(name, engine);`
 - Adds it to the registry (association between name and object)
 - Name: String formatted as a URL
 - Default port: 1099
- Clients can talk to the server only over the registry
- Keeps track of registered servers
- Wakes up the server if there is an incoming request

Fall 2007— Pinar Yolum

16

Compiling

- Build a jar file of the interfaces
- Build the server side files (And Make them accessible over a Web server)
 - Compile
 - `rmic -d FortuneServer`
- Build the client side files
 - Compile
 - Copy `FortuneServer_Stub.class` to the client directory

Fall 2007— Pinar Yolum

17

Running

- Unset CLASSPATH (no java classes should be reachable)
- Run RMI registry first
 - `start rmiregistry` (optional port number)
- Set classpath
 - It should see both `Fortune.jar` and remote object implementation
- Start the server
 - `start java`
 - Djava.rmi.server.hostname=localhost
 - Djava.security.policy=java.policy FortuneServer localhost

Fall 2007— Pinar Yolum

18

Running

- Start the client
 - Set CLASSPATH to include `Fortuna.jar`
 - `java -Djava.security.policy=java.policy FortuneClient localhost`

Fall 2007— Pinar Yolum

19

Constructing a Stub

- Naming creates a stub and stores it in the registry
- Identify the root class (`getClass()`)
 - If the remote object implements an interface that extends `Remote`, then root class is remote object's class
 - Otherwise, the highest superclass that implements an interface that extends `Remote` is the root class
- Name of stub=`concat(binary_name_root_class, "_Stub")`
- Classloader of root class will load the stub to JVM
- Stub must extend `RemoteStub` and must have a constructor whose one parameter is a `RemoteRef`
 - `RemoteRef ref=engine.getRef();`
- Construct the class with `RemoteRef`
 - `ComputeEngine_Stub stub=new ComputeEngine_Stub(ref);`

Fall 2007— Pinar Yolum

20