Introduction to Java Remote Method Invocation (RMI)

CS 446/646, ECE 452, SE 464

Thiago Tonelli Bartolomei
June 2007
Sources used in this presentation

- The Java RMI Tutorial at Sun
- Wikipedia (for history)
- A Java RMI server framework
- Eclipse RMI Plugin
  - (References in the end)
Outline

1) Main idea and some history
2) Java RMI Architecture
3) Details / Examples
4) References / Further study
1) Main Idea

• Provide a mechanism for Client-Server communication in which:
  – The server is abstracted, i.e., the client does not necessarily know if the server is local or remote.
  – Programmers must not deal with communication protocols, serialization and connection management in general.
Some History

• Remote Procedure Calls (RPCs) were first described in RFC 707, in 1976.
  – Interface Description Languages, Marshalling...

• Object-oriented analogous
  – Java Remote Method Invocation
  – Microsoft .NET Remoting

• Newer technologies
  – EJB (based on RMI)
  – SOAP
  – Webservices
2) Java RMI Architecture

1 – Service is exported to port X and its stub is bound to registry with a name, e.g., “MyService”.

2 – A consumer uses a locator to get a representation of the registry present on the server host.

3 – The consumer uses the registry to get a stub of the service, using as key the service's bound name, “MyService”.

4 – The consumer uses the Stub as if it was the original Service. All communication (method calls) to the service is redirected through port X to the server's Service object.
3) Details / Examples

- Development Process
  1) Define the Remote Interface
  2) Implement the Interface (server)
  3) Implement export code (server)
  4) Implement location code (client)

- Deployment Process
  1) Start RMIRegistry process in server
  2) Start server application
  3) Start client application
The Remote Interface

- A Java Interface extending `java.rmi.Remote`
- Methods must throw `java.rmi.RemoteException`
- Arguments and return type must implement `java.io.Serializable`. (more details later)

```java
package simple;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface IService extends Remote {
    public String concat(String a, String b) throws RemoteException;
}
```
Implementing the Interface (Server)

- Provide a default empty constructor...
- And Just implement the methods!

```java
package simple;

import java.rmi.RemoteException;

public class Service implements IService {
    public String concat(String a, String b) throws RemoteException {
        return a + b;
    }
}
```
Exporting the Service (server)

- Make sure there is a security manager (more details later)
- Create the service object
- Export the service and get a stub
- Bind the stub to the registry with a identifier name
  - (Display code in Eclipse)
Locating the Service (client)

- Make sure there is a security manager
- Locate the registry
- Get a service stub from the registry, using the same identifier used by the server export
- Use the stub for the communication
  - (Display Code in Eclipse)
Compiling and Deploying

- As of Java 6, simply compile with javac
- Start RMI Registry in the server with `rmiregistry &`
- Start a JVM with server
  - `-Djava.security.policy=URL` (security policy location)
  - `-Djava.rmi.server.codebase=URL` (stubs location)
- Start a JVM with client (similar JVM options)
Some “Rules/Hints”

• Registry
  - When starting rmiregistry, CLASSPATH can NOT have any of the application classes

• Security Manager
  - security.policy can be at $HOME/.java.policy
  - If defining policy with -Djava.security.policy:
    • Can use =file if file is in the current directory
    • Can use =URL (with http://, file://, etc)

• Codebase
  - If URL is a file://, must end with /
  - Always use /, never use \ (even in Windows)
Some “Rules/Hints”

- **Hosts**
  - Registry methods (such as `getRegistry()`) have `localhost` as default. This is OK for the server, but the client must know where the server is located!
Some “Rules/Hints”

• Ports
  − Registry default is 1099. But you should remember to use the ports defined for your group (see in the course's homepage).
  − Sending 0 to `exportObject(service, 0)` causes port to be dynamically defined by the OS. However, you should use only your port range!
  − JVM is smart with ports, you can export several objects to the same port without conflict (even the registry)!
Some “Rules/Hints”

- Threads
  - Client thread will block until the method returns.
  - RMI does not guarantee that 2 concurrent requests to the server run in different threads. Therefore, you must create code to spawn worker threads and guarantee thread safety.
  - In the server, a “reaper thread” is created and will hang while there are exported objects referenced. When exporting objects, they get registered if there is a reference to them.
    - Service s = new Service(); export(s); VS.
    - export(new Service());
Some “Rules/Hints”

• Object Identity
  – Stubs are local but represent a remote object
    • Methods called in stubs are executed in the remote machine, in the context of the remote object.
  – Parameters and return objects are:
    • Passed by value, if it is a primitive type or any serializable object.
    • Passed by reference, if it is a Stub!
A more advanced topic

• Make things more modular/reusable with design patterns
  - Factory: create the service local or remote, removing this code from the client
  - Adapter: clients don't have to handle RemoteException – but you have basically to maintain 2 similar interfaces!
4) References / Further Study

- The Java RMI Tutorial
  - http://java.sun.com/docs/books/tutorial/rmi/

- Eclipse RMI Plugin
  - http://www.genady.net/rmi/

- A Java RMI server framework