CSE 70: Remote Method Invocation

Ingolf Krueger

Department of Computer Science & Engineering
University of California, San Diego
La Jolla, CA 92093-0114, USA

California Institute for Telecommunications and Information Technologies
La Jolla, CA 92093-0405, USA
Learning Goals for Today
Learning Goals

• Understand the key elements of a Remote Method Invocation (RMI) system:
  – Remote Object/Interface [service provider]
  – Registry [white/yellow pages]
  – Client [finds service provider, calls upon service]

• Be able to write basic RMI systems yourself
Java Remote Method Invocation – Client/Server for you and me
package chat.server;

public class ChatServer {
    String server_id;

    public ChatServer(String server_id) {
        this.server_id = server_id;
    }

    ...
}

ChatServer.java
package chat.client;
import chat.server.*;

public ChatClient {
    public ChatClient(ChatServer server, String client_id) {
        server.sign_on(client_id);
    }
    ...
}

ChatSystem Snippet

... public static void main(String[] args) {
    ChatServer server =
        new ChatServer();
    ChatClient client1 =
        new ChatClient(server, "ikrueger");
    ChatClient client2 =
        new ChatClient(server, "mamenari");

    if (server.is_present("ikrueger") {
        System.out.println("found him!");
    }
}...
Client/Server on Same VM

![Diagram of Client/Server on Same VM](image)

- **Computer**
  - **Java VM**
    - **Heap**
      - `client1: ChatClient`
      - `server: ChatServer`
      - `client2: ChatClient`
    - **Stack**
New Situation: We are distributing the client and server over multiple VMs!
Client/Server on Different Computers & VMs

Computer 3

JVM 3

server: ChatServer

Computer 1

JVM 1

client1: ChatClient

Computer 2

JVM 2

client2: ChatClient
Networking Challenges

Computer 1

JVM 1

client1: ChatClient

Network

Computer 3

JVM 3

server: ChatServer
Network Programming Challenges?

• Do we have to worry about:
  – creating sockets?
  – data marshaling/unmarshaling?
  – network errors?
  – having the right internet addresses for the server?
  – create threads to execute concurrent client calls?

No! RMI does all of that for us!
RMI Application – Players (idealized)
Server Registers with Registry
Client Locates Registry and Looks Up Server
Client Receives Stub (implements Server Interface)
Client Makes Local (!) Call

- Computer 1
  - JVM 1
    - client1: ChatClient
    - stub: ChatServer
  - 4. local call

- Computer 3
  - JVM 3
    - ChatServer
    - server: ChatServerImpl
  - registry: Registry
    - ChatServer

© Ingolf Krueger, 2010
Stub Makes Remote Call

Computer 1
  JVM 1
    client1: ChatClient
    stub: ChatServer

Computer 3
  JVM 3
    ChatServer
    server: ChatServerImpl

Computer 3
  JVM 4
    registry: Registry
    ChatServer

5. remote call
Server Returns Result to Stub
Stub Returns Result to Client
End-to-End View
Writing RMI Applications

1. Code the server (a “remote” object at runtime)
   1. specify the server’s interface
   2. implement the interface
      • have the server register itself with the registry

2. Code the client
   1. Lookup the server in the registry
   2. Obtain the server’s interface and instantiate it as a local object (stub)
   3. Make calls upon the stub (as if server was local)

3. Start the registry

4. Start the server – it registers with the registry

5. Start the client(s) – they look up the server in the registry
Distributed Chat
Client/Server Application
package simplechat.server;

import java.rmi.*;

public interface ChatServer extends Remote {
    public void sign_on(String client_id)
        throws RemoteException;
    public void publish(String message)
        throws RemoteException;
    public boolean isPresent(String client_id)
        throws RemoteException;
}

Remote Server Implementation
package simplechat.server;

import java.rmi.*;
import java.rmi.registry.*;
import java.rmi.server.UnicastRemoteObject;
import java.util.*;

public class ChatServerImp extends UnicastRemoteObject implements ChatServer {

    private String server_id;
    private List<String> clients;
    private List<String> log;

    // Constructor and methods implementation...
}
ChatServerImp(String server_id) throws RemoteException {
    super();
    this.server_id = server_id;
    clients = new LinkedList<String>();
    log = new LinkedList<String>();
}

public synchronized void publish(String message) {
    log.add("published:"+message);
}
public synchronized void sign_on(String client_id) {
    clients.add(client_id);
}

public boolean isPresent(String client) {
    return clients.contains(client);
}
public static void main(String[] args) {
    try {
        Registry registry = LocateRegistry.getRegistry();
        registry.rebind("chat server",
                new ChatServerImp("chat server"));
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}
Client Implementation
package simplechat.client;

import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

import simplechat.server.*;

public class ChatClientImp {
    private ChatServer server;

    public ChatClientImp {
        private ChatServer server;
    }
}
public ChatClientImp(String client_id,
                     String server_id) {
    try {
        Registry registry =
            LocateRegistry.getRegistry();
        server            =
            (ChatServer)
            registry.lookup(server_id);
        server.sign_on(client_id);
    }
}
assert(server.isPresent(client_id));
server.publish("client " +
    client_id+" is on");
} catch (Exception e) {
    e.printStackTrace();
}
}
public static void main(String args[]) {
    ChatClientImp client1 = 
        new ChatClientImp("ikrueger", "chat server");
    ChatClientImp client2 =
        new ChatClientImp("bdemchak", "chat server");
}
}
Compile & Run!
Compile and Run

- Compile the server in Eclipse
- Compile the client in Eclipse
- at the console:
  
  \texttt{rmiregistry} &
  
  this installs a registry at the default port 1099
- launch server (from within Eclipse)
- launch client (from within Eclipse)
- Enjoy!

- Many subtleties due to security model in Java
  - Discussed in lab today (5-6pm)
What have you learned today?
Learning Goals

• Be able to write Java code that has packages, interfaces and exceptions.

• Understand the key elements of a Remote Method Invocation (RMI) system:
  – Remote Object/Interface [service provider]
  – Registry [white/yellow pages]
  – Client [finds service provider, calls upon service]

• Be able to write basic RMI systems yourself
Self-study exercise:
Implement basic RMI system where both client and server are remote objects