

Outline

- Background
- RMI Architecture
- A TimeServer
- Passing Objects
- Passing Behavior

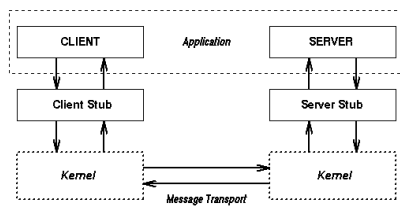
2/19/01

ICSS235 - Java RMI

1

Remote Procedure Calls

- Introduced in 1984



2/19/01

ICSS235 - Java RMI

2

Remote Procedure Calls

- Language neutral
- Based on the semantics of procedure calls

2/19/01

ICSS235 - Java RMI

3

Java Remote Method Invocation

- Java RMI provides a framework that can be used to build distributed systems in Java.
- RMI uses the Java object model to support object oriented operations between hosts.
- Objects can be passed as both parameters and return values.
- Behavior can be passed across a network.

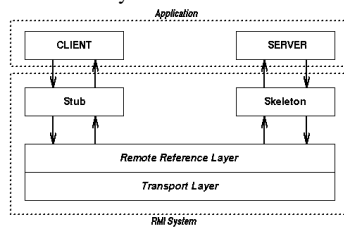
2/19/01

ICSS235 - Java RMI

4

RMI Architecture

- Very similar to RPC
- Consists of three layers



2/19/01

5

RMI Architecture

- Stubs/Skeletons
 - Stubs act as a proxy for the server
 - Forward requests to server-side skeleton
 - Skeleton makes call and returns result to stub
 - Marshal Streams are used to transmit parameters
- Remote Reference Layer
 - Handles the semantics of the call
 - Currently multicast delivery is not supported
- Transport Layer
 - Responsible for connection management
 - Currently uses stream-based transport

2/19/01

ICSS235 - Java RMI

6

Timed.java

- Consider a simple time server and client. The following interface describes the server:

```
public interface Timed extends java.rmi.Remote {
    String getTime() throws java.rmi.RemoteException;
}
```

- Remote objects must directly or indirectly implement `java.rmi.Remote`
- Remote interfaces must be public and throw `java.rmi.RemoteException`

2/19/01

ICSS235 - Java RMI

7

TimedImpl.java

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

public class TimedImpl extends UnicastRemoteObject implements Timed {
    public TimedImpl() throws RemoteException { super(); }

    public String getTime() throws RemoteException {
        return new Date().toString();
    }

    public static void main(String args[] ) {
        System.setSecurityManager(new RMISecurityManager());

        try {
            TimedImpl obj = new TimedImpl();
            Naming.rebind("TimeServer", obj); // URL based lookup
        } catch (Exception e) {
            System.out.println("TimedImpl err: " + e.getMessage());
            System.exit(1);
        }
    }
}
```

2/19/01

ICSS235 - Java RMI

8

TimedClient.java

```
import java.rmi.*;

public class TimedClient {
    public static void main(String argv[] ) {
        String curTime = "";

        System.out.println("Attempting Lookup");
        try {
            Timed obj =
                (Timed)Naming.lookup("TimeServer");
            curTime = obj.getTime();
        } catch (Exception e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }

        System.out.println(curTime);
    }
}
```

2/19/01

ICSS235 - Java RMI

9

Putting it Together

- Create `.class` files in the usual way

```
javac Timed.java TimedImpl.java TimedClient.java
```

- Create the stub/skeleton using `rmic`

```
rmic Timed.class
```

- Start the Java registry on the host machine

```
rmiregistry
```

- Start the server

- Start the client

2/19/01

ICSS235 - Java RMI

10

Passing Objects

- RMI can pass full objects as parameters and return values
- Since objects contain behavior in addition to state, this means that it is possible to pass behavior between machines.
- RMI uses Object Serialization to convert an object to a byte stream for transport.

2/19/01

ICSS235 - Java RMI

11

Object Serialization

- To become serializable, a class need only declare that it implements the `java.io.Serializable` interface
- The `java.io.Serializable` interface has no methods and serves only to identify that a class may be serialized.
- For most objects the default methods, that read/write each field individually, provided by object serialization are sufficient.

2/19/01

ICSS235 - Java RMI

12

LogMessage.java

```
// Define what system log messages look like.
import java.util.*;

public class LogMessage implements java.io.Serializable {
    private String msg; private Date time;

    public LogMessage() {}

    public void setMessage(String what) {
        msg = new String(what);
        time = new Date(); }

    public String toString() {
        StringBuffer output = new StringBuffer();

        output.append(time).append(" ");
        output.append(msg);

        return new String(output);
    }
}
2/19/01 ICSS235 - Java RMI 13
```

Log.java

```
// A simple log server. Provides a single method that
// writes a LogMessage to the system log.

public interface Log extends java.rmi.Remote {
    void log(LogMessage msg) throws java.rmi.RemoteException;
}
2/19/01 ICSS235 - Java RMI 14
```

LogImpl.java

```
// The actual system logger. Accepts LogMessages from any number
// of clients. The log is a plain ASCII file.

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

public class LogImpl extends UnicastRemoteObject implements Log {
    public static final String defaultFilename = "syslog.log";
    private static PrintWriter logFile;

    public LogImpl() throws RemoteException { super(); }

    public synchronized void log(LogMessage msg)
        throws RemoteException {
        logFile.println(msg);
        logFile.flush();
    }
}
2/19/01 ICSS235 - Java RMI 15
```

LogImpl.java (continued)

```
public static void main(String args[]) {
    System.setSecurityManager(new RMISecurityManager());

    try {
        logFile =
            new PrintWriter(new FileOutputStream(defaultFilename,true));
    } catch (Exception e) { System.exit(1); }

    try {
        LogImpl obj = new LogImpl();
        Naming.rebind("LogService", obj);
    } catch (Exception e) { System.exit(1); }

    logFile.print("Log Service started at ");
    logFile.println(new Date());
    logFile.flush();
}
}
2/19/01 ICSS235 - Java RMI 16
```

Main.java

```
// A log service client
import java.rmi.*;

public class main {
    public static void main(String argv[]) {
        Log syslog = null;
        StringBuffer message = new StringBuffer("log entry X");

        try { syslog = (Log)Naming.lookup("LogService"); }
        catch (Exception e) {
            System.out.println("main: " + e.getMessage());
            System.exit(1);
        }

        for (char ch='0'; ch<='9'; ch++) {
            message.setCharAt(message.length()-1,ch);

            try { syslog.log(new LogMessage(new String(message))); }
            catch (Exception e) {
                System.out.println("main: " + e.getMessage());
                System.exit(1);
            }
        }
    }
}
2/19/01 ICSS235 - Java RMI 17
```

Client Identification

- The big problem with the basic log service is that it does not record the identification of the client making the log entry.
- Consider a second version of the log service that allows clients to obtain an ID, and to use this ID when making log entries.

LogPlus.java

```
public interface LogPlus extends java.rmi.Remote {
    LogId getId() throws java.rmi.RemoteException;
    void log(LogMessage msg, LogId id)
        throws java.rmi.RemoteException;
}
```

2/19/01

ICSS235 - Java RMI

19

LogId.java

```
// The interface for a LogId

// Note that this interface does not extend
// java.rmi.remote which means it cannot be a remote
// object.

public interface LogId {
    void setId();
    String toString();
}
```

2/19/01

ICSS235 - Java RMI

20

SimpleLogId.java

```
// Generate a very simple ID for a client

import java.net.*;

public class SimpleLogId
    implements java.io.Serializable, LogId {

    private String id;

    public SimpleLogId() {}

    public void setId() {
        try {id = InetAddress.getLocalHost().getHostName();}
        catch(Exception e) {
            id = "Unknown host";
        }
    }
}
```

2/19/01

ICSS235 - Java RMI

21

LogImpl.java

```
// A LogPlus implementation. This service also supports
// old style Log messages

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

public class LogImpl extends UnicastRemoteObject
implements Log, LogPlus {
    public static final String defaultFilename = "log.log";

    private static PrintWriter logFile;

    public LogImpl() throws RemoteException {
        super();
    }
}
}19/01 ICSS235 - Java RMI 22
```

LogImpl.java (continued)

```
public synchronized void log(LogMessage msg)
throws RemoteException {
    logFile.println(msg);
    logFile.flush();
}

public synchronized void log(LogMessage msg, LogId id)
throws RemoteException {
    logFile.print(id);
    logFile.print(" ");
    logFile.println(msg);
    logFile.flush();
}

public LogId getId() throws RemoteException {
    return new SimpleLogId();
}
}19/01 ICSS235 - Java RMI 23
```

LogImpl.java (continued)

```
public static void main(String args[]) {
    System.setSecurityManager(new RMISecurityManager());

    try {
        logFile =
            new PrintWriter(new FileOutputStream(defaultFilename,true));
    }
    catch (Exception e) { System.exit(1); }

    try {
        LogImpl obj = new LogImpl();
        Naming.rebind("LogService", obj);
    }
    catch (Exception e) { System.exit(1); }

    logFile.print("Log Service started at ");
    logFile.println(new Date());
    logFile.flush();
}
}2/19/01 ICSS235 - Java RMI 24
```

Client.java

```
// A simple LogPlus client
import java.rmi.*;

public class main {
    public static void main(String argv[]) {
        LogPlus syslog = null;
        LogId myId = null;

        LogMessage msg = new LogMessage();

        try {
            syslog = (LogPlus)Naming.lookup("LogService");
        } catch (Exception e) {
            System.out.println("main: " + e.getMessage());
            System.exit(1);
        }
    }
}
```

2/19/01

ICSS235 - Java RMI

25

Client.java (continued)

```
        try {
            myId = syslog.getId();
        } catch (Exception e) {
            System.out.println("main: " + e.getMessage());
            System.exit(1);
        }

        myId.setId();
        System.out.println(myId);
        msg.setMessage("Log message");

        try {
            syslog.log(msg, myId);
        } catch (Exception e) {
            System.out.println("main: " + e.getMessage());
            System.exit(1);
        }
    }
}
} 2/19/01
```

ICSS235 - Java RMI

26

Remote Execution

```
interface Executor extends Remote {
    void exec(Runnable task);
}

class Exec implements Executor {
    void exec(Runnable task) { task.run(); }
}

class User {
    void m() {
        Executor e = (Executor)(Naming.lookup("Executor"));
        e.exec(new Runnable(), Serializable {
            public void run() { someLongComptuation(); } });
    }
}
```

2/19/01

ICSS235 - Java RMI

27

References

- [1] Birrell, A.D., and Nelson, B.J., "Implementing Remote Procedure Calls", ACM Transactions on Computer Systems, vol. 2, pp. 39-59, Feb 1984.
- [2] JDK 1.1.4 Documentation, Sunsoft,
<http://java.sun.com/products/jdk/1.1/docs/index.html>.
- [3] Java Remote Method Invocation - Distributed Computing for Java, Sunsoft, <http://java.sun.com/marketing/collateral/javarmi.html>.
- [4] Getting Started with RMI, Sunsoft,
<http://java.sun.com/products/jdk/1.1/docs/guide/mi/getstart.doc.html>.
- [5] Remote Method Invocation Specification
<http://java.sun.com/products/jdk/1.1/docs/guide/mi/spec/rmiTOC.doc.html>.
