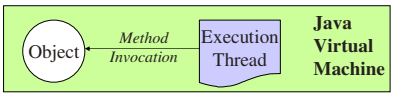
Java RMI : Remote Method Invocation

CS 4119 - Computer Networks
Columbia University - Spring 2000

Alexander V. Konstantinou
akonstan@cs.columbia.edu

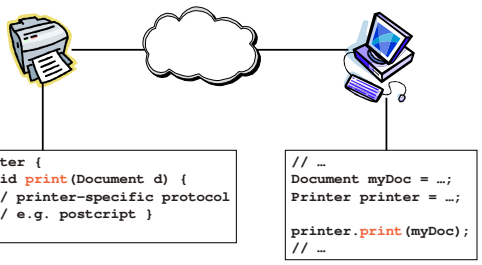
Introduction : Remote Computation

- Objects encapsulate **data + operations**
- Usually stored and evaluated **locally**



- **Remote** storage/evaluation can also be useful :
  - Object encapsulates physical resource (e.g. Printer)
  - Data resides remotely and is very large (e.g. phone directory lookup)

Example: Print Object



```
Printer {
  void print(Document d) {
   // printer-specific protocol
   // e.g. postcript }
}
```

```
// …
Document myDoc = …;
Printer printer = …;

printer.print(myDoc);
// …
```


Remote Print Object Implementation

- How can Java support remote operations ?
- Use of **proxy** objects :

```
public class Printer_Proxy {
  public void print(Document doc){
    // magic communication
} }
```

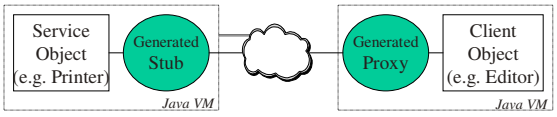
```
Document myDoc = …;
Printer printer = …;

printer.print(myDoc);
```

Java VM

Remote Method Invocation Overview

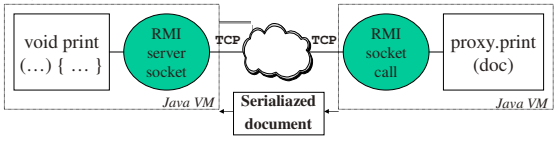
- RMI is Java's mechanism for **automatically** generating proxy classes.
- User codes service and client objects
- RMI compiler generates network communication code



Remote Interface Example

- What ties it all together ?
- *Answer*: Server, stub, proxy and client all share the same **remote interface**

```
public interface PrintService extends java.rmi.Remote {
   public void print(Object obj)
       throws java.rmi.RemoteException;
}
```



RMI Features

- Language specific (Java)
- Object oriented
  - Full objects as parametears
  - Supports design patterns
- Mobile behavior
  - Move interface implementation from client to server, and server to client
- Safe & Secure (Java VM security)
- Connects to existing/legacy (JNI/JDBC)

RPC versus RMI

- Procedural
- Language Independent
- External data representation (XDR)
- Basic types as parameters
- Pointers require explicit handling
- No code mobility (same for CORBA, DCOM)

- Object Oriented
- Language Specific
- Java Object Serialization
- Any object implementing serialization as parameter
- References to local and remote objects handled automatically (deep copy)
- Mobile code (Java byte-code)

RMI Terminology

- A ***remote object*** is one whose methods can be invoked from another Java Virtual Machine, potentially on a different host.
- ***Remote method invocation*** (RMI) is the action of invoking a method of a remote interface on a remote object.

```
// Local method invocation example
HashTable table = new HashTable();
table.put("akonstan", "secRet!");
```

```
// Remote method invocation example (incomplete)
PasswordDb db = (PasswordDb) Naming.lookup("//myhost/cs4119db");
db.put("akonstan", "secRet!");
```

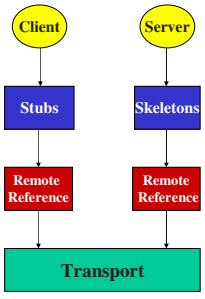

Remote Invocation Semantics

The semantics of remote method invocations differ in some ways from those of local method invocations :

- Clients interact with remote ***interfaces***.
- Non-remote arguments, and results from, a remote method invocation are passed by ***copy*** rather than by reference.
- A remote object is passed by ***reference***, not by copying the actual remote implementation.
- Clients invoking remote objects must handle ***additional failure modes*** (exceptions)

Java RMI Architecture

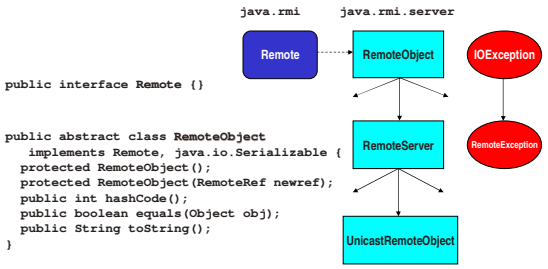
- Servers extend `RemoteObject` and **implement** remote interfaces.
- Any serializable object can be sent as a parameter or returned as a response
- The RMI compiler generates client stubs (proxies) and server skeletons(dispatchers)



Java Object Serialization

- RMI parameters passed as serialized objects
- Serialized objects are converted to a **stream of bytes**.
- Serialization stores the class structure along with the values of the object (class structure only stored once per class).
- Serialization handles **references** by traversing them and serializing objects along the way.
- You do not need to write any special code to utilize the serialization routines. It is sufficient to implement the `java.io.Serializable` interface (this is a marker interface and does not define any methods).

RMI Interfaces and Classes

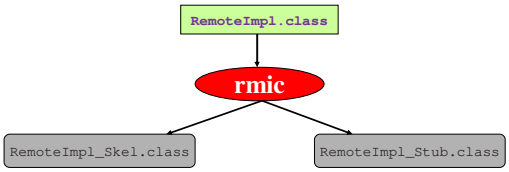



```
public interface Remote {}
```

```
public abstract class RemoteObject
   implements Remote, java.io.Serializable {
  protected RemoteObject();
  protected RemoteObject(RemoteRef newref);
  public int hashCode();
  public boolean equals(Object obj);
  public String toString();
}
```

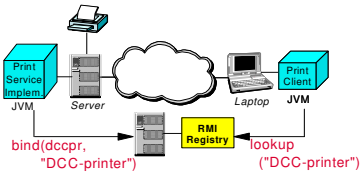

Stub & Skeleton Generation

- Client Stubs & Server Skeletons are generated by the `rmic` compiler.
- The `rmic` compiler takes as input a class implementing remote interfaces and outputs a Stub and a Skeleton class



Locating servers with RMI Registry

- RMI registry is the object directory service.
- Objects are bound to the registry using string names.
- The registry process may execute on any network host.
- RMI URL: `rmi://myhost:1099/DCC-printer`



RMI Example

RMI Example : Remote List Printer

- Implement a remote Printer Server.
- Print Server will accept a linked list of Java Objects, and print them to standard out.
- We will define the following classes :
  - `ListNode` : a node in the link list
  - `LinkedList` : a linked list object supporting limited operations
  - `ListPrinter` : the interface for our remote printer server
  - `ListPrinterImpl` : an implementation of the `ListPrinter` interface
  - `Client` : a printing client that will create and send a list for printing

RMI Example (List Classes)

A simple definition of a linked list. Note that both classes must implement the serialization interface so that they may be used in remote methods.

```
public class ListNode
implements
java.io.Serializable {
  private Object value;
  private ListNode next;

  public ListNode
    (Object value,
     ListNode next) {}
  public Object getValue()
  public void setValue
    (Object value) {}
  public ListNode getNext()
  public void setNext
    (ListNode next) {}
}
```

```
public class LinkedList
implements java.io.Serializable {
  private ListNode head;
  private ListNode tail;

  public LinkedList() {
    head = null;
    tail = null;
  }
  public ListNode getHead() {}
  public ListNode getTail() {}
  public void insert(Object obj)
  public void append
    (LinkedList list) {}
  public boolean isEmpty() {}
  public void print() {}
}
```

RMI Example (Remote Interface)

```
public interface ListPrinter extends java.rmi.Remote {
  boolean print(LinkedList list)
               throws java.rmi.RemoteException;
}
```

- Declare a public interface that extends `java.rmi.Remote`
- Each method must declare `java.rmi.RemoteException` in its throws clause
- A remote object passed as an argument or return value must be declared as the remote interface, not the implementation class

RMI Example (Server Implement.)

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

public class ListPrinterImpl extends UnicastRemoteObject
implements ListPrinter {

// Constructor
  public ListPrinterImpl(String name) throws RemoteException {
    super();
  }

  // Implement ListPrinter method
  public boolean print(LinkedList list) throws RemoteException{
    list.print();
    return true;
  }
}
```


RMI Example (Server Impl. Cont.)

```
  public static void main(String[] args) {
    // Create and install a security manager
    System.setSecurityManager(new RMISecurityManager());

    try {
      ListPrinterImpl obj = new ListPrinterImpl("ListPrinterServer");

      // Bind to the registry (rmiregistry)
      Naming.rebind("//sutton.cs.columbia.edu:1099/myprinter",
                    obj);
      System.out.println("myprinter bound in registry");
    } catch (Exception e) {
      System.out.println("ListPrinterImpl err: " + e.getMessage());
      e.printStackTrace();
    }
  } // main
} // ListPrinterImpl
```


RMI Example (Print Client)

```
import java.util.Date;
import java.rmi.*;
public class Client {
  public static void main(String[] args){
    LinkedList a = new LinkedList();
    a.insert(new Date());
    a.insert("Today is");

    try {
      ListPrinter lpr = (ListPrinter) Naming.lookup
        ("//" + ListPrinterImpl.serverHost + ":" +
         ListPrinterImpl.serverPort + "/myprinter");
      lpr.print(a);
    } catch(Exception e) {
      System.out.println("Client exception: "+e.getMessage());
    }
  }
}
```


RMI Example (Compilation)

```
$ javac ListPrinterImpl.java
$ javac Client.java
$ rmic ListPrinterImpl
```

Compilation will generate :

- `class` files for each java class,
- `ListPrinterImpl_Stub.class` : client side proxy for the remote object,
- `ListPrinterImpl_Skel.class` : server side dispatcher for calls to the actual remote object implementation,

RMI Example (Execution)

Execute the following in separate windows :

```
$ rmiregistry 6234
```

You must restart the registry after changing the remote interface!

```
$ java ListPrinterImpl
```

The ListPrinterImpl process should output :

```
myprinter bound in registry
```

Start the client in a separate window :

```
$ java Client
```

The ListPrinterImpl process should output :

```
Today is -> Sun Mar 08 19:02:31 EST 1998 -> EOL
```

Advanced RMI Topics

Remote Activation

- Registering a remote object with the RMI registry requires the object to be continually active
- JDK 1.2 introduces the RMI daemon (Remote Activation)
- Daemon registers information about remote object implementations that are created **on-demand**.

Performance Issues

- RMI calls should be used for **large**-grain computation.
- Every RMI method invocation results in :
  - A new TCP connection to the remote server
  - Creation of a new thread on the remote server

Concluding Notes

- RMI moves RPC to the object world
- Object serialization simplifies marshaling of data
- Language specific mechanism
  - may be exported using the Java Native Interface (JNI)
- RMI may be used to implement agents
- Other advanced RMI topics :
  - RMI over Secure Socket Layer (SSL)
  - Exporting class byte-code using HTTP

How-to Overview

- Define remote **interface**
- Write class **implementing** remote interface (and extending UnicastRemoteObject)
- Use `rmic` to **compile** class stub and proxy
- **Start** RMI registry (with stub & proxy classes in classpath)
- **Execute** server and bind to RMI **registry**
- **Lookup** remote object in registry
- **Invoke** remote method on proxy
- **Handle** remote invocation failures

Java and Java RMI Resources

- Sun Microsystems, Java RMI home
  - `http://java.sun.com/products/jdk/rmi`
- Sridharan, Prashant. *Advanced Java Networking*. Prentice Hall PTR, Upper Saddle River, New Jersey, 1997
- The Java Tutorial (RMI chapter)
  - `http://java.sun.com/docs/books/tutorial/index.html`