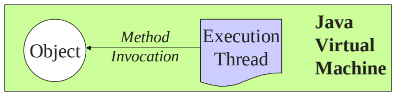
Remote Method Invocation
Java RMI & Web-Services

CS 4119 - Computer Networks
Columbia University - Spring 2003

Alexander V. Konstantinou
`akonstan@cs.columbia.edu`

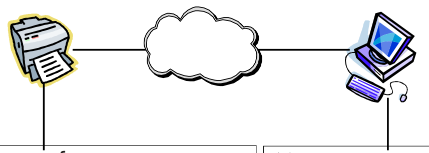
Introduction : Remote Computation

- 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



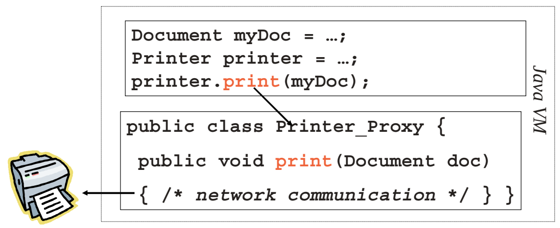
```
Printer {
 void print(Document d) {
  // printer-specific
  // protocol (e.g. PDF)
 }
}
```

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

printer.print(myDoc);
// …
```

Remote Print Object Implementation

- How can Java support remote operations ?
  – Use of **proxy** objects (encapsulate comm.)

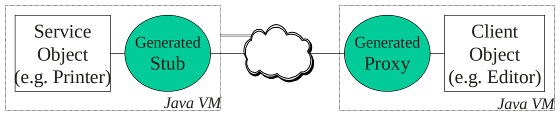


```
Document myDoc = …;
Printer printer = …;
printer.print(myDoc);
```

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

Remote Method Invocation Overview

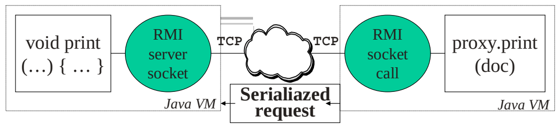
- RMI is Java's mechanism for **automatically** generating *proxy* classes.
- User implements service and client objects
- RMI compiler generates network communication code



Remote Interface Example

- What ties server, stub, proxy & client together ?
  – *Answer*: 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 parameters
  - – 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	• Object Oriented
• Language Independent	• Language Specific
• External data representation (XDR)	• Java Object Serialization
• Basic types as parameters	• Any object implementing serialization as parameter
• Pointers require explicit handling	• References to local and remote objects handled automatically (deep copy)
• No code mobility (same for CORBA, DCOM)	• 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
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 :

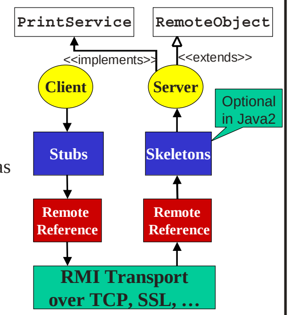
- Clients interact with remote ***interfaces***.
- Non-remote arguments, and results from, a remote method invocation are passed by ***copy*** rather than by reference.
- A remote object is passed by ***reference***, not by copying the actual remote implementation.
- Clients invoking remote objects must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

Java Object Serialization

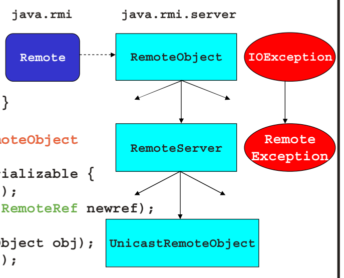
- RMI parameters passed as serialized objects
- Serialized objects are converted to a **stream of bytes**.
- Serialization stores the class structure along with the values of the object (class structure only stored once per class).
- Serialization handles **references** by traversing them and serializing 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).

Java RMI Architecture

- Servers extend `RemoteObject`
- Servers *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)



RMI Interfaces and Classes

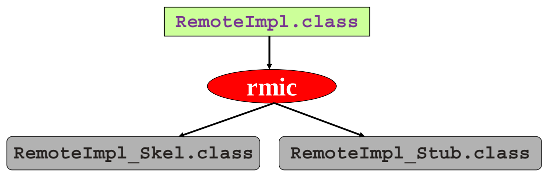


```
public interface Remote {}

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();
}
```

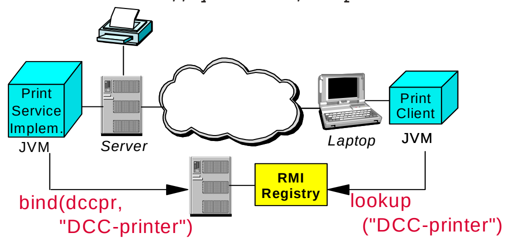
Stub & Skeleton Generation

- 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.
- 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 :

  `ListPrinter` : the interface for our remote printer server

  `ListPrinterImpl` : an implementation of the **`ListPrinter`** interface

  `Client` : a client that will create and send a list for printing

RMI Example (Remote Interface)

- 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

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

RMI Example (Server Impl.)

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

public class ListPrinterImpl
                 extends UnicastRemoteObject
                 implements ListPrinter {

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

RMI Example (Server Impl. 2)

```
  /** Implement ListPrinter method */
  public boolean print(List list)
                     throws RemoteException {
    for(Iterator iter=list.iterator();
                iter.hasNext(); ) {
      System.out.print(iter.next());
      if (iter.hasNext())
        System.out.print("->");
    }
    System.out.println("");
    return true;
  }
```

RMI Example (Server Impl. 3)

```
public static void main(String[] args) {
  System.setSecurityManager
                  (new RMISecurityManager());
  try {
    ListPrinterImpl obj = new
        ListPrinterImpl("ListPrinterServer");

    // Bind to the registry (rmiregistry)
    Naming.rebind("//sutton:1099/myprinter",
                  obj);
  } catch (Exception e) { /* Handle */ }
} // main
```

RMI Example (Print Client)

```
public class Client {
  public static void main(String[] args){
    List list = new LinkedList();
    list.add(new java.util.Date());
    list.add("Today is");
    try {
      ListPrinter lpr = (ListPrinter)
        Naming.lookup
          ("//sutton:1099/myprinter");
      lpr.print(list);
    } catch(RemoteException e) { /*Handle*/ }
  }
}
```

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
- `ListPrinterImpl_Skel.class` : server side dispatcher

RMI Example (Execution)

```
$ rmiregistry 6234
```

You must restart the registry after changing the remote interface!

Start the RMI server

```
$ 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 introduced the RMI daemon (Remote Activation)
- Daemon registers information about remote object implementations that are created **on-demand**.

RMI 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

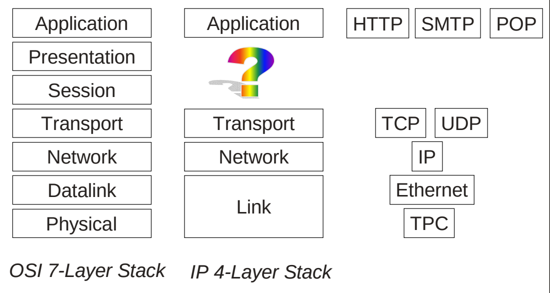
- Define remote **interface**
- Write class **implementing** remote interface (and extending UnicastRemoteObject)
- Use **`rmic`** to **compile** class stub and proxy
- **Start** RMI registry (with stub & proxy classes in classpath)
- **Execute** server and bind to RMI **registry**
- **Lookup** remote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

Java RMI Resources

- Sun Microsystems, Java RMI home
  - **`http://java.sun.com/products/jdk/rmi`**
- A. Konstantinou, et al. *Beginning Java Networking*. Wrox Press, 2001.

Web-Services

The Internet "Middleware" Quest



Historic Contenders

- Remote Procedure Call (RPC)
  - Language-independent BER-encoding
  - Procedural, no distributed services
- CORBA
  - Distributed objects infrastructure
  - Complex, "expensive", "heavy"
- HTML & HTTP
  - Simple, universally adopted
  - Not suitable for program-based communication

eXtensible Markup Language (XML)

- HTML for Data (W3C standard)
  - Text-based (least-common denominator)
  - Tags denote meaning (not presentation)
  - Extensible schema (user defined tags)

```
<p>
John Doe<br>
500 W 120th Str<br>
New York, NY 10027<br>
<tt>jdoe@columbia.edu</tt>
</p>
```

```
<card>
   <name>
      <given>John</given>
      <family>Doe</family>
   </name>
   <address type="business">
      <street>500 W …</street>
```

VCard XML Example (W3C)

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:vCard = "http://www.w3.org/2001/vcard-rdf/3.0#">
  <rdf:Description rdf:about = "http://foo.com/staff/corky" >
    <vCard:FN> Corky Crystal </vCard:FN>
    <vCard:N rdf:parseType="Resource">
      <vCard:Family> Crystal </vCard:Family>
      <vCard:Given> Corky </vCard:Given>
      <vCard:Prefix> Dr </vCard:Prefix>
    </vCard:N>
    <vCard:TITLE> Computer Officer Class 3 </vCard:TITLE>
    <vCard:ROLE> Programmer </vCard:ROLE>
    <vCard:TEL rdf:parseType="Resource">
      <rdf:value> +61 7 555 5555 </rdf:value>
      <rdf:type rdf:resource="http://www.w3.org/2001/vcard-rdf/3.0#work"/>
      <rdf:type rdf:resource="http://www.w3.org/2001/vcard-rdf/3.0#voice"/>
    </vCard:TEL>
```

Using XML for Communication

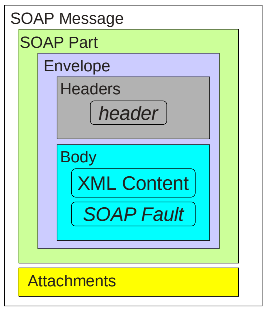
- XML over HTTP
  - Static document transfer
  - Dynamic request (URL encoded/HTTP POST)
  - Question: how to encode the request?
- Answer: Standardize request structure
  - Encode method and arguments as XML doc!
  - POST the XML request

SOAP: XML-based RPC

- Simple Object Access Protocol (SOAP)
- Communication protocol (document transfer)
- Format for sending messages
- Platform & language independent (XML)
- Simple and extensible
- Standardized (W3C)
- Widely adopted
  - Sun J2EE & Microsoft .Net

SOAP Message

- Headers
  - Identification, Routing, Correlation, QoS, …
- Body
  - Mandatory info (method, args)
  - Fault information
- Attachments
  - XML + binary data

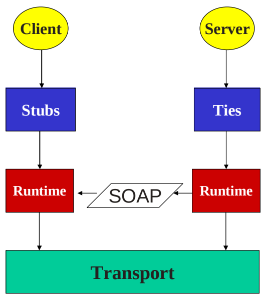


SOAP Example

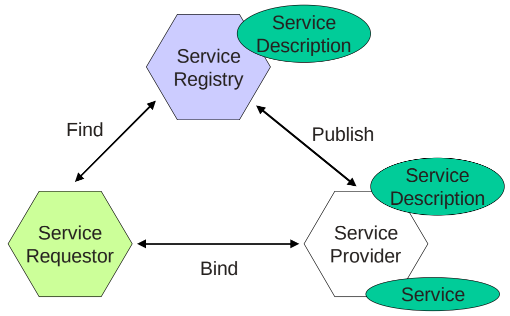
```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV=
    "http://schemas.xmlsoap.org/soap/envelope/"
  <SOAP-ENV:Body>
    <m:print
      xmlns:m="http://columbia.edu/printer">
      <message>Hello world<message/>
    </m:print>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

JAX-RPC: Java XML RPC

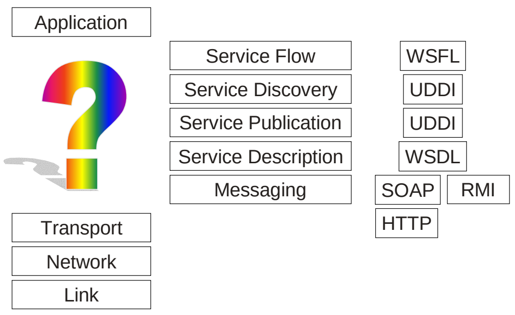
- Remote interfaces
  - Similar to RMI!
  - Restricted types
- Procedure:
  - Code interface + impl.
  - Generate Web-Service descr. (**`wsdeploy`** tool)
  - Generate stubs (**`wscompile`** tool)
  - Use `wscompile` tool



Web Services Model



Web Services Stack



Java XML APIs

- J2EE: Java2 Enterprise Edition
  - Web Services execution environment
- JAXP: XML Processing
  - Parsing: SAX (event), DOM (tree)
  - Processing: XSLT (transformations)
- JAX-RPC: RMI-type binding
- JAXM: SOAP & Messaging (lower level)

XML Protocols

- SAX, DOM: parsing models & APIs
- DTD, XSchema: schema definition
- XPath: paths into documents
- XSL & XSLT: transformation
- XPointer: links documents
- XQuery: query
- … encryption, signature, key mgmt …

Java Web Services References

- Sun Java Web Services
  - http://java.sun.com/webservices
    - White-papers, tutorial, developer-pack, examples
- W3C Web Services Standardization
  - http://www.w3.org/2002/ws/
- UDDI: Description & Discovery
  - http://www.uddi.org/