Distributed Computing

- Programs that cooperate and communicate over a network
  - e-mail
  - web server and web client
  - SETI@Home

Remote Method Invocation

- Easy way to get distributed computation
- Have stub for remote object
  - calls to stub get translated into network call
- Arguments and return values are passed over network
  - Java takes care of the details

Key Features of Distrib. Comp.

- Machines are not all the same
  - But all adhere to same communication protocol
- Network is “slow”
  - Sending a message takes a lot of time
- Network is unreliable
  - Machines may join and leave with no warning
  - Part of the network may fail

Different Approaches to Distributed Computation

- Connecting via sockets
  - E.g., project 3
  - custom protocols for each application
- RPC/DCOM/CORBA/RMI
  - make what looks like a normal function call
  - function actually invoked on another machine
  - Arguments are marshalled for transport
  - Value is unmarshalled on return

A Simple Example

```java
class ChatServerImpl { // runs on one mach.
  public void receive(String s) {
    System.out.println(s);
  }
  ...
}

class Chatter { // runs on another mach.
  public static void main(String[] args) {
    ChatServer c = // get remote object;
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    while (true) {
      System.out.print("> ");
      c.receive(br.readLine());
    }
  }
}
```
Remote Objects

- Object should
  - extend `java.rmi.server.UnicastRemoteObject`
  - constructor declared to throw `RemoteException`
- implement a remote interface
  - a remote interface extends `java.rmi.Remote`
  - all methods in a remote interface throw `RemoteException`
  - “Something bad happened on the network”
- Side note: actually, don’t need to extend `UnicastRemoteObject`, but it’s much easier

Remote Interfaces

- Client only sees the `RemoteInterface`
  - `ConcreteObject` can have other methods
- Remote objects represented using stub
  - stub sends arguments over network
  -stub receives result back from network

Compiling Stubs with `rmic`

- Generates stub code for a class
  - for 1.1, also generates skeleton class
    - stub on client side communicates with skeleton on remote side
  - skeleton not needed for 1.2+
    - and 1.2+ generates position-independent code
  - use `-v1.2` if you can (e.g., for project 5)
- Generates stubs for all methods declared in the class’s `Remote interface`
  - other methods don’t get a stub

Passing Arguments

- To pass an argument to a remote method
  - (or return a result from a remote method)
  - it must be either
    - a primitive type (int, double, etc.),
    - Serializable (e.g., String), or
    - Remote (i.e., implement a sub-interface of `Remote`)
  - Primitives passed as you’d expect

Passing Serializable vs. Remote

- Serializable objects passed by value
  - same `Serializable` in different calls materializes different objects at receiver
- Remote objects passed by reference
  - same `Remote object` in different calls yields same stub object, which passes arguments back to same remote object
**Stub Code**

- Objects contain both data and code
  - When you receive a remote object, you need the stub for that remote object
- Solution #1: All clients have stub code on their classpath
  - Or stub code for another class with same remote interface

**Policy Files**

- In addition to security manager, need to specify a security policy
  ```java
  grant {
  permission java.net.SocketPermission "*:1024-65535", "connect,accept";
  permission java.net.SocketPermission "*:80", "connect";
  }
  ```
- Set security policy when JVM started
  - java -Djava.security.policy=<file name>

**Downloading Code**

- Solution #2: Provide a code base where stub code for objects can be downloaded
  - `java -Djava.rmi.server.codebase=<url> ...
  - Specifies location of classes originating from this server
  - url can be, e.g., http:// or file://

**Getting the First Remote Object**

- Can make objects available in RMI registry
  - Each object has a name (that you specify)
  - Registry listens on a port (1099 default)
- `Naming.lookup(url)` gets object from reg.
  - E.g., `Naming.lookup("rmi://localhost/Chat")`
  - Use to get first reference to remote object
  - Don’t need to lookup for objects returned by remote methods

**Security Manager**

- Downloading code (even stub code) from the internet is potentially risky
  - Need to limit what downloaded code could do
  - Must install a Security Manager before you download any code from RMI code bases
- Can use
  ```java
  System.setSecurityManager(
  new RMISecurityManager());
  ```

**Starting an RMI Registry**

- Method 1: Separate RMI registry process
  - Command `rmiregistry`
    - Run with stubs in classpath, or specify codebase
    - Listens on port 1099 by default
- Method 2: Start in same JVM
  - `LocateRegistry.createRegistry(int port)`
  - Advantage: dies when your program dies
    - No registries lying around on machine
Advertising Remote Objects

- Call Naming.{bind/unbind/rebind} to place objects in registry
  - E.g., Naming.bind("rmi://localhost/Chat");
- Can bind/unbind/rebind name on localhost
- Can lookup name on any host

Example: RMI Chat Server

- Server
  - runs the chat room
- Client
  - participant in chat room
  - receives messages from others in room
- Connection
  - uniquely identifies a client
  - used to speak in chat room

Server

```java
interface Server extends Remote {
    Connection logon(String name, Client c)
        throws RemoteException;
}
```

Server’s Remote Object creation

```
Server s = new ServerImpl();
```

Object added to table because it implements extension of Remote interface

Server

```
interface Connection extends Remote {
    /** Say to everyone */
    void say(String msg)
        throws RemoteException;
    /** Say to one person */
    void say(String who, String msg)
        throws RemoteException;
    String [] who()
        throws RemoteException;
    void logoff()
        throws RemoteException;
}
```

Client

```
interface Client extends Remote {
    void said(String who, String msg)
        throws RemoteException;
    void whoChanged(String [] who)
        throws RemoteException;
}
```

Remote Object registry

Client’s Remote Object creation

Client looks up Server

After lookup finished

Invokes remote Server method

Receives remote call
Executes the call

... create new Connection object

Server

... call logon ...

"Bill" Stub c

ClientImpl

ServerImpl

Returns the result

... return this as the result

Hosted Remote Objects

ConnectionImpl

ServerImpl

Receives the result

... conn

ServerImpl

Stub code for remote logon call

ClientImpl

Conn Stub

unmarshalled return value