CMSC 433 – Programming Language Technologies and Paradigms Spring 2007

Java RMI

May 3, 2007

Distributed Computing

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

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
  - 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

Remote Method Invocation

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

5

A Simple Example

```
class ChatServerImpl ... { // runs on one mach.
  public void say(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.say(br.readLine());
    }
} }
```

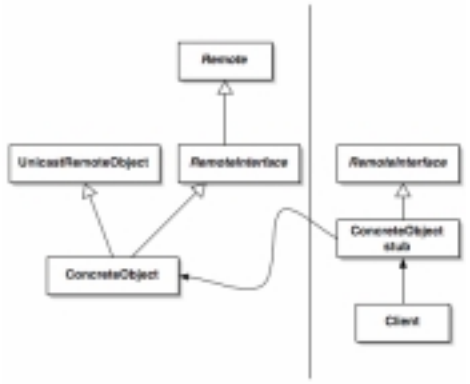
6

Remote Objects

- Object should
  - Extend java.rmi.server.UnicastRemoteObject
    - Constructor declared to throw RemoteException
      - Fine Print: actually, don't NEED to extend UnicastRemoteObject, but it's much easier
  - 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"

7

Remote Interfaces



8

Stubs

- 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

9

Compiling Stubs with rmic

- Generates stub code for a class
  - Generates position-independent code
- Generates stubs for all methods declared in the class' Remote interface
  - Other methods don't get a stub

10

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

11

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:/

12

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

```
System.setSecurityManager(
                new RMISecurityManager());
```

13

Policy Files

- In addition to security manager, need to specify a security policy

```
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>

14

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

15

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

16

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

17

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

18

Server

```
interface Server extends Remote {

      Connection logon(String name, Client c)
            throws RemoteException;

  }
```

19

Connection

```
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;
      }
```

20

Client

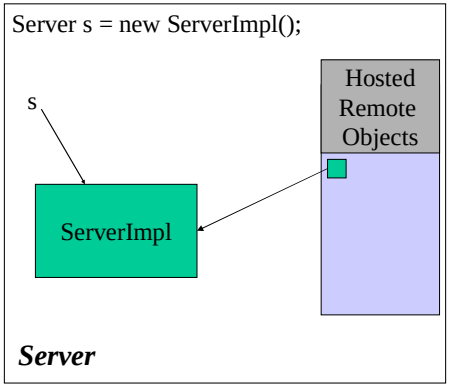
```
interface Client extends Remote {

      void said(String who, String msg)
            throws RemoteException;

      void whoChanged(String [] who)
            throws RemoteException;
      }
```

21

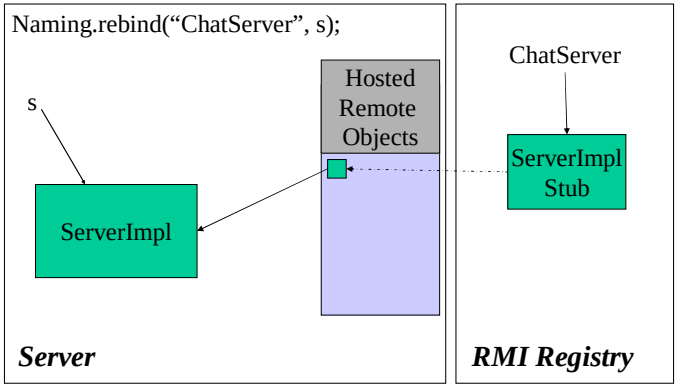
Server's Remote Object creation



*Object added to table because it implements extension of **Remote** interface*

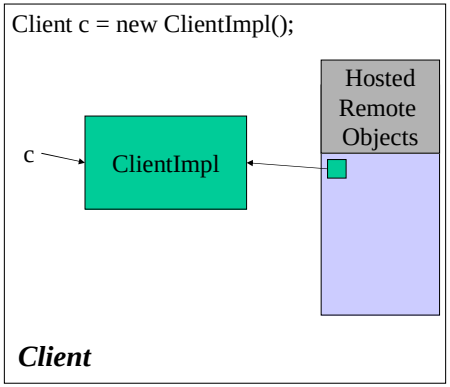
22

Remote Object registry



23

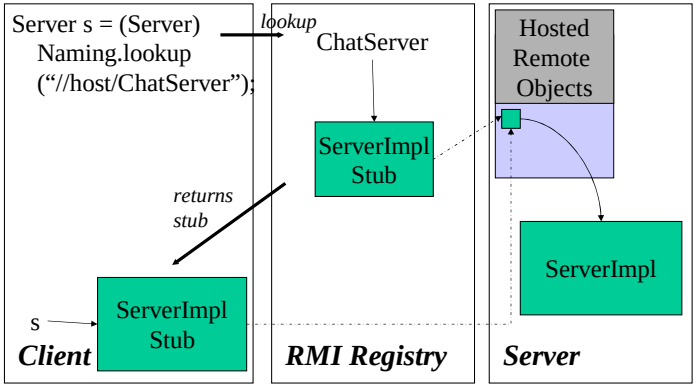
Client's Remote Object creation



*Client object also implements extension of **Remote** interface*

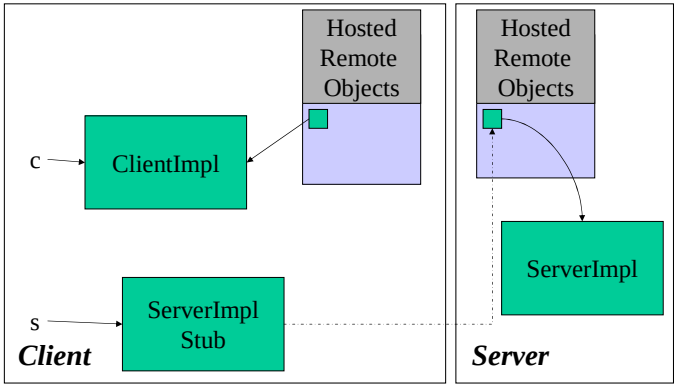
24

Client looks up Server



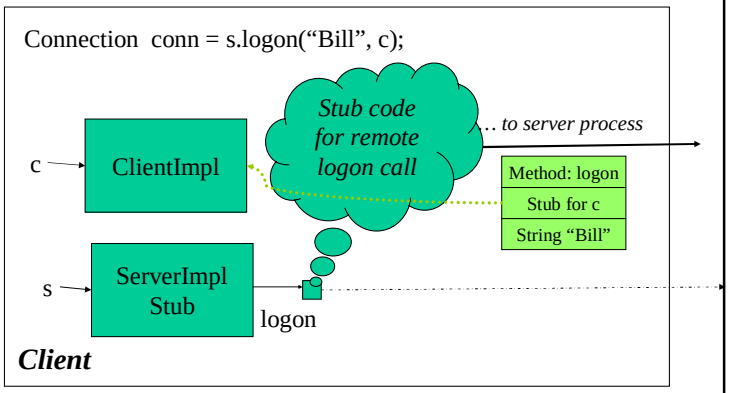
25

After lookup finished



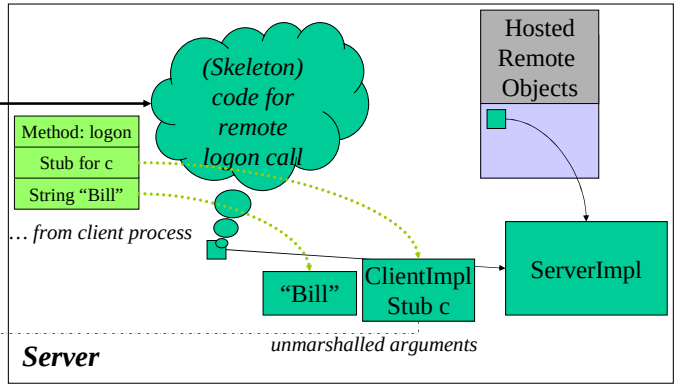
26

Invokes remote Server method



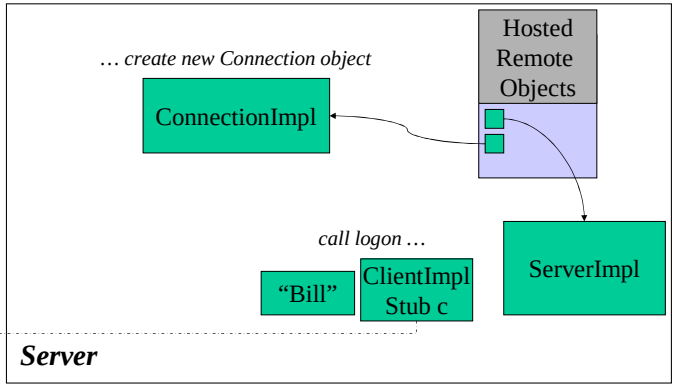
27

Receives remote call



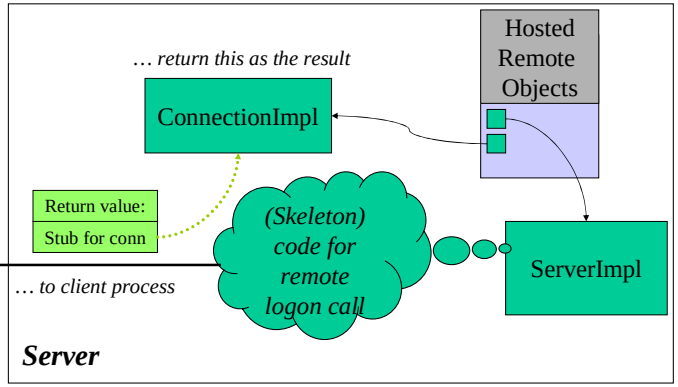
28

Executes the call



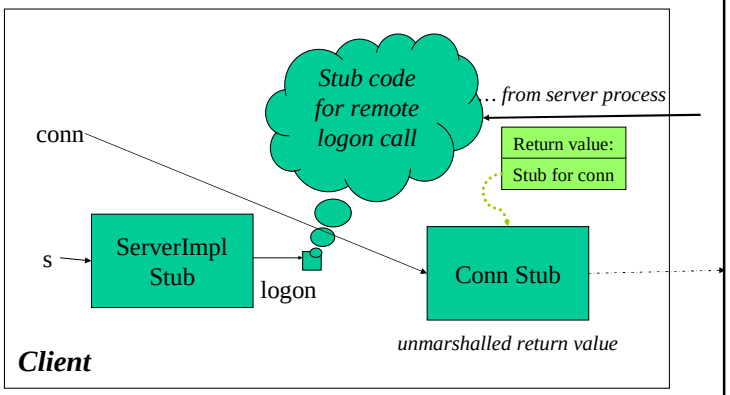
29

Returns the result



30

Receives the result



31