CMSC433, Spring 2002 Programming Language Technology and Paradigms
Java RMI

Adam Porter

November 21, 2002

Different Approaches to Distributed Computation

- High-performance, parallel scientific apps
- Connecting via sockets
  - custom protocols for each application
- RPC/DCOM/CORBA/RMI
  - make what looks like a normal function call
  - function is 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
- Arguments and return values can be passed over network

Remote Objects and Interfaces

- Remote Objects are those that can be referenced remotely
  - extends java.rmi.UnicastRemoteObject
  - constructor throws java.rmi.RemoteException
- Remote interfaces describe services that can be provided remotely
  - extends java.rmi.Remote interface
  - all methods throw java.rmi.RemoteException

`rmic` - RMI Compiler

- Generates stub code for a class
  - For 1.1, also generates skeleton class
  - skeleton not needed for 1.2+
- Generates stubs for all methods declared in the class's Remote interface
  - other methods don't get a stub

Passing arguments

- Can pass arbitrary values as arguments
- Can return arbitrary values as results
- To pass a value, it must either be
  - Serializable, or
  - Remote
- Passing the same Serializable object in different calls
  - will materialize different objects at receiver

Downloading code

- When you pass a reference to a remote class
  - receiver needs stub class
- When you pass a reference to serializable class
  - receiver needs class
- Annotate ref's with RMI codebase
  - where code can be loaded from

SecurityManager

- Must install some Security Manager to allow download of classes from RMI codebase
- Can use RMISecurityManager
  System.setSecurityManager(new RMISecurityManager());
- Modify policy file to grant permissions

Naming.lookup

- Naming.lookup is used to bootstrap RMI communication
  - Get your first reference to a remote object
- Run an RMI Registry
  - a separate Java VM (command **`rmiregistry`**)
  - listens to a particular port (default 1099)
- Can bind/unbind/rebind name on localhost
- Can lookup name on any host

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

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

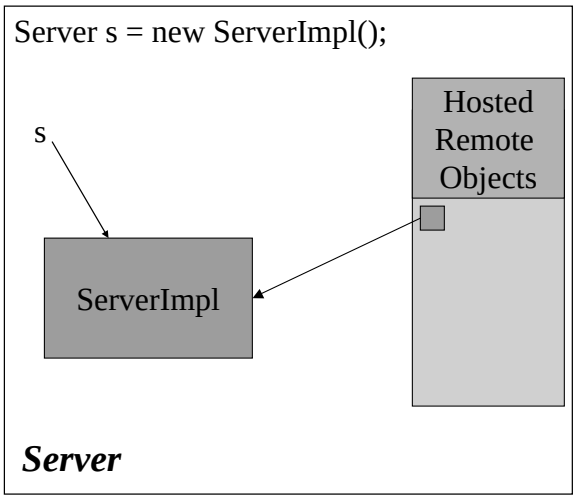

Connection

```
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;
        }
```

Client

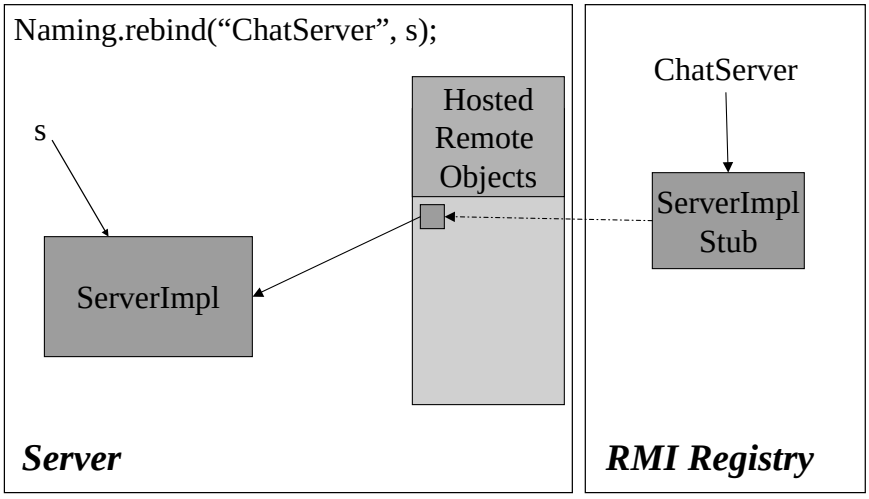
```
interface Client extends Remote {
        void said(String who, String msg)
                throws RemoteException;
        void whoChanged(String [] who)
                throws RemoteException;
        }
```


Server's Remote Object creation

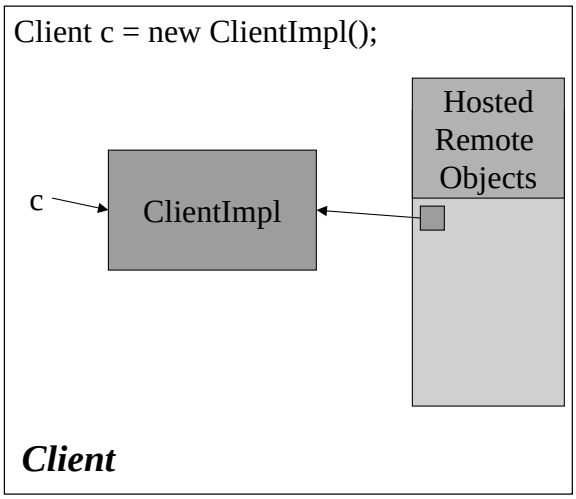


*Object added to table because it implements extension of **Remote** interface*

Remote Object registry

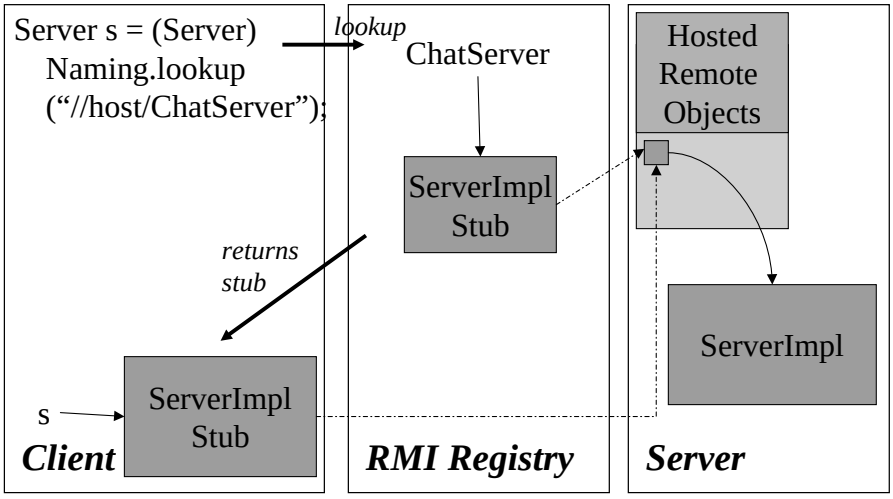


Client's Remote Object creation

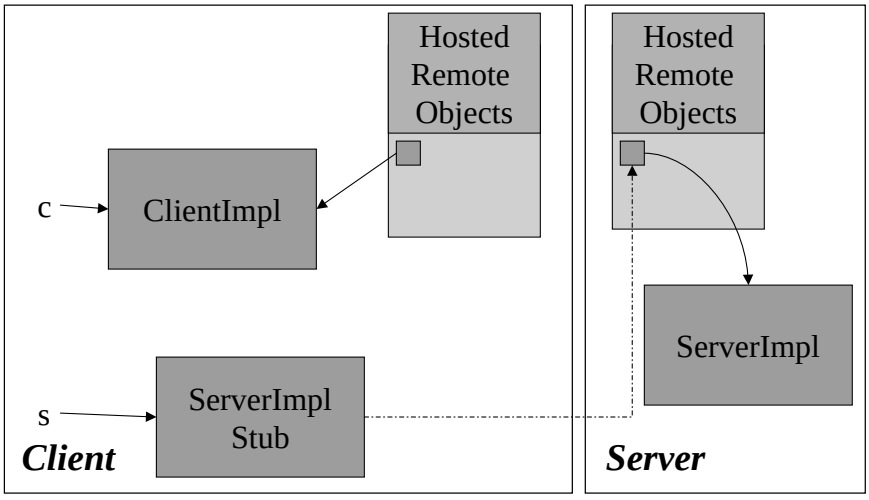


*Client object also implements extension of **Remote** interface*

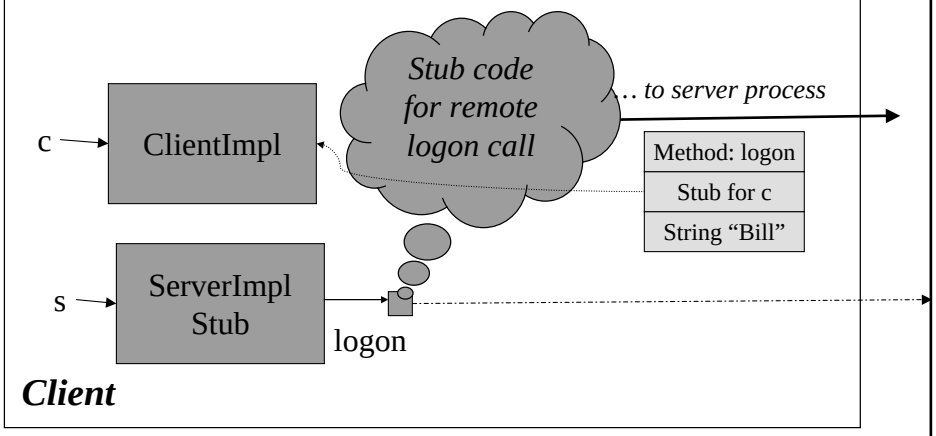
Client looks up Server



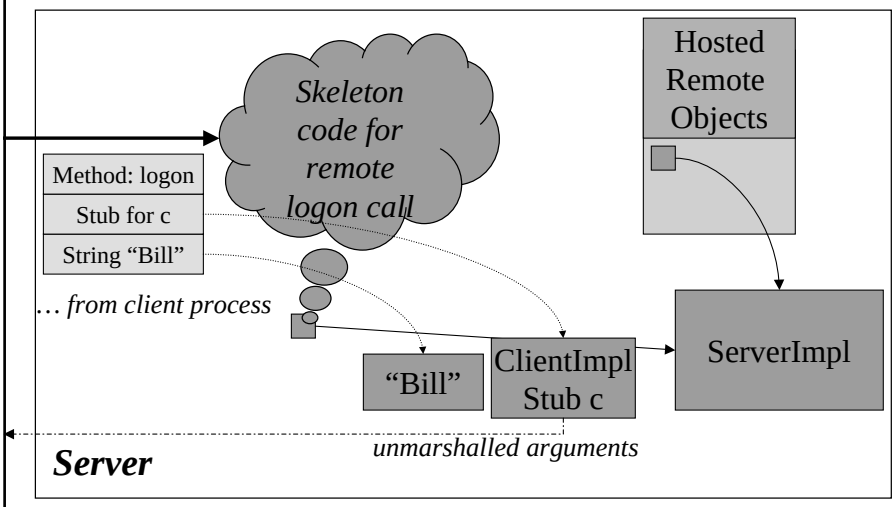
After lookup finished



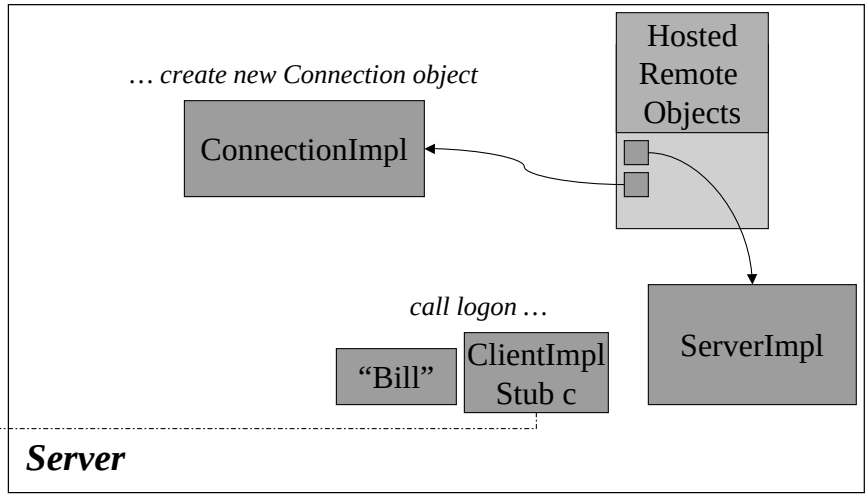
Invokes remote Server method



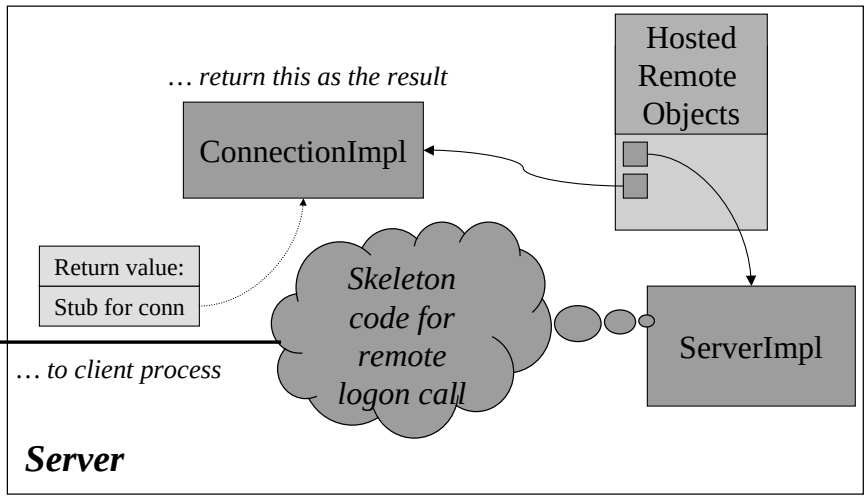
Receives remote call



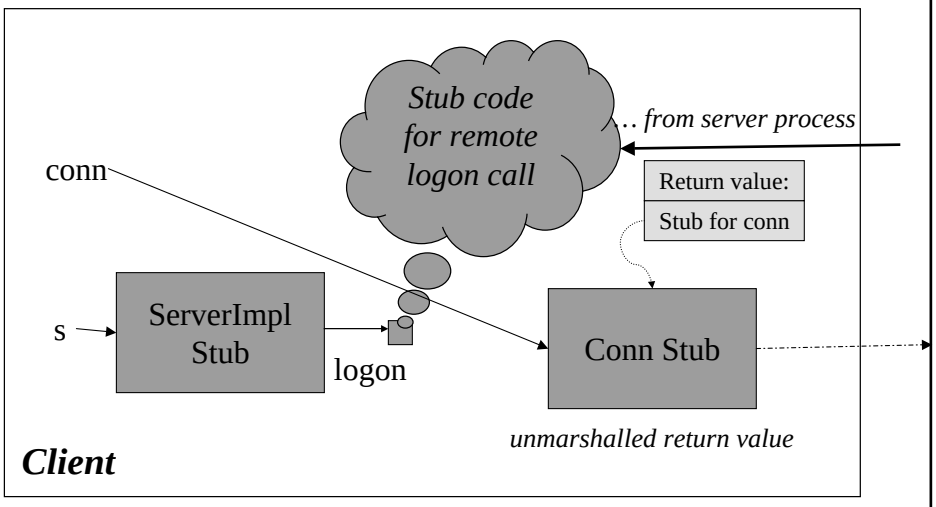
Executes the call



Returns the result



Receives the result



CMCS 433, Fall 2002 -Adam Porter 23