

CS 580 Client-Server Programming
Spring Semester, 2007
Doc 18 Distributed Objects & RMI
April 19, 2007

Copyright ©, All rights reserved. 2007 SDSU & Roger Whitney, 5500 Campanile Drive, San Diego, CA 92182-7700 USA. OpenContent (<http://www.opencontent.org/opl.shtml>) license defines the copyright on this document.

References

Pattern-Oriented Software: A System of Patterns, Buschman, Meunier, Rohnert, Sommerlad, Stal, 1996, pp. 323-338 (Client-Dispatcher-Server), pp. 263-275 (Proxy)

Design Patterns: Elements of Reusable Object-Oriented Software, Gamma, Helm, Johnson, Vlissides, 1995, pp. 207-218 (Proxy)

CS 696 Emerging Technologies: Java Distributed Computing, Doc 5 Distributed Object Basics, 1999, <http://www.eli.sdsu.edu/courses/spring99/cs696/notes/distObj/distObj.html>, Doc 6 RMI Intro, Doc 7 Java Security Model

Permissions in the Java SE Development Kit, <http://java.sun.com/javase/6/docs/technotes/guides/security/permissions.html>

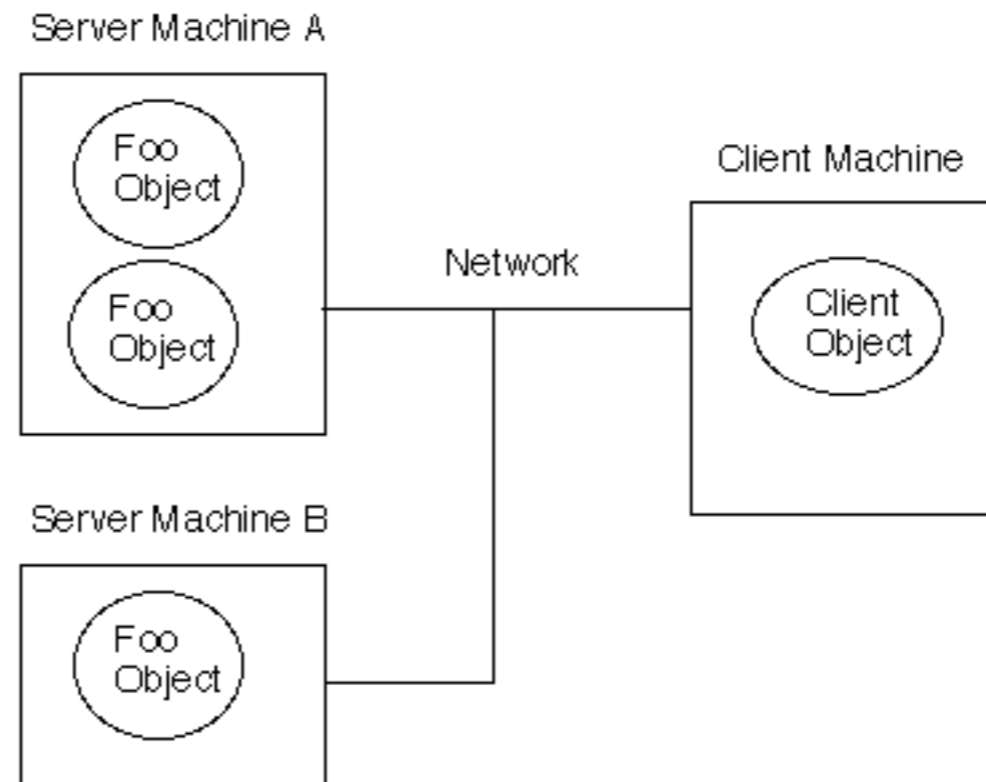
Default Policy Implementation and Policy File Syntax, <http://java.sun.com/javase/6/docs/technotes/guides/security/PolicyFiles.html>

Distributed Objects, http://en.wikipedia.org/wiki/Distributed_object

Java Network Programming 3'rd Ed, Harold, O'Reilly, 2005, Chapter18 Remote Method Invocation

Distributed Objects

System that allows sending of messages to objects on remote machines



```
public class Foo {  
    public String hello() { return "Hi there"; }  
}
```

```
Foo remote = getRemoteObject();  
String message = remote.hello();
```

Some Existing Systems

Java RMI

CORBA

DCOM

Pyro (Python)

dRuby

ReplicaNet (C++)

Local verses Distributed Objects

Life Cycle

References

Request Latency

Parallelism

Communication

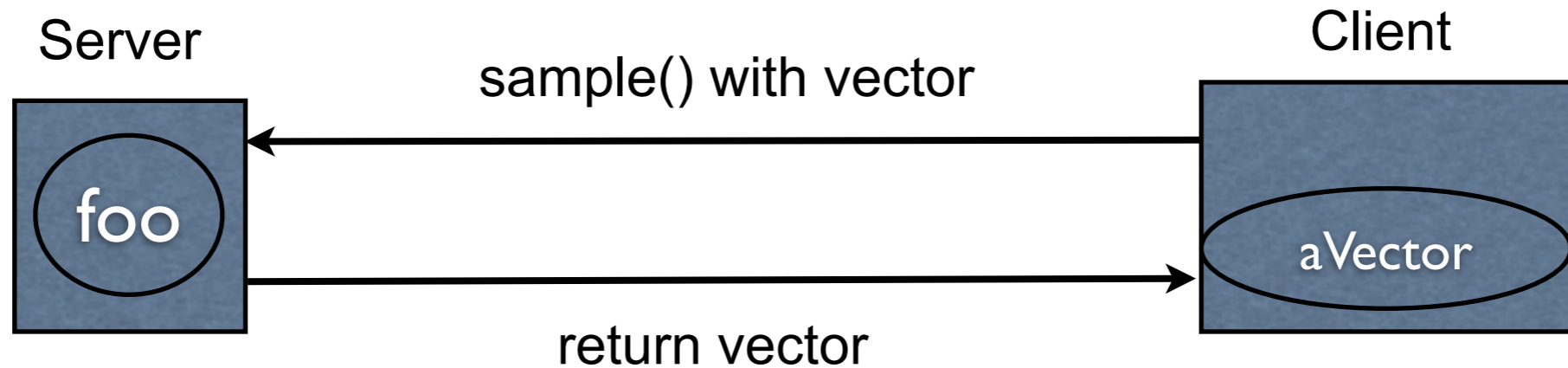
Failure

Security

Example

```
public class Foo {  
    public Vector sample( Vector input, int data ) {  
        Integer remoteInt = new Integer( data );  
        input.addElement( remoteInt );  
        return input;  
    }  
}
```

```
Foo remote = getRemoteObject();  
Vector b = remote.sample(aVector);
```



Issues

How does the client find the server?

Network connections must be made but to where and by who?

How does the client object interact with the server object?

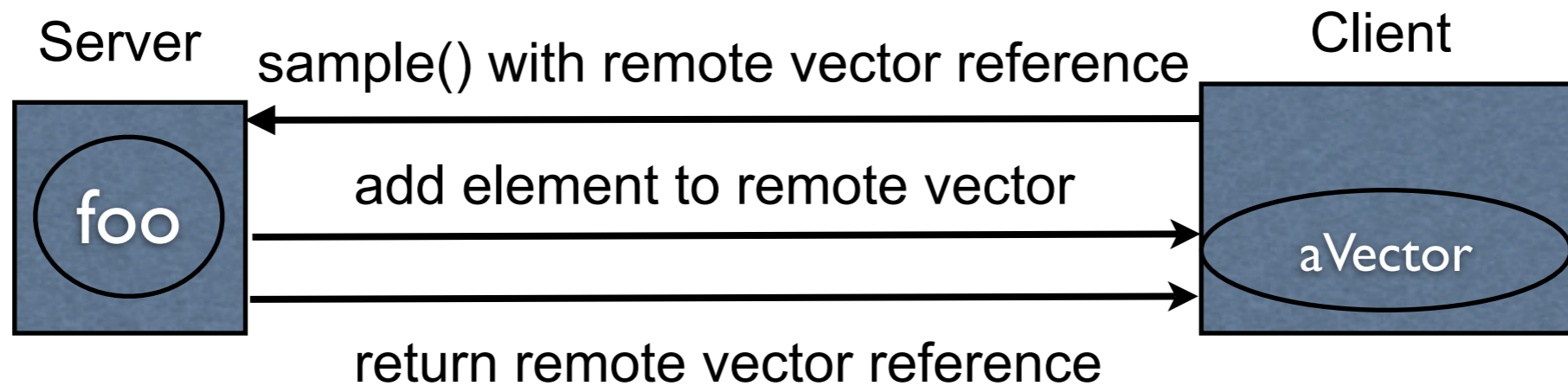
How can we send a method request across the network?

Remote Method Parameters

Don't move the parameters, just send them messages remotely

```
public class Foo {  
    public Vector sample( Vector input, int data ) {  
        Integer remoteInt = new Integer( data );  
        input.addElement( remoteInt );  
        return input;  
    }  
}
```

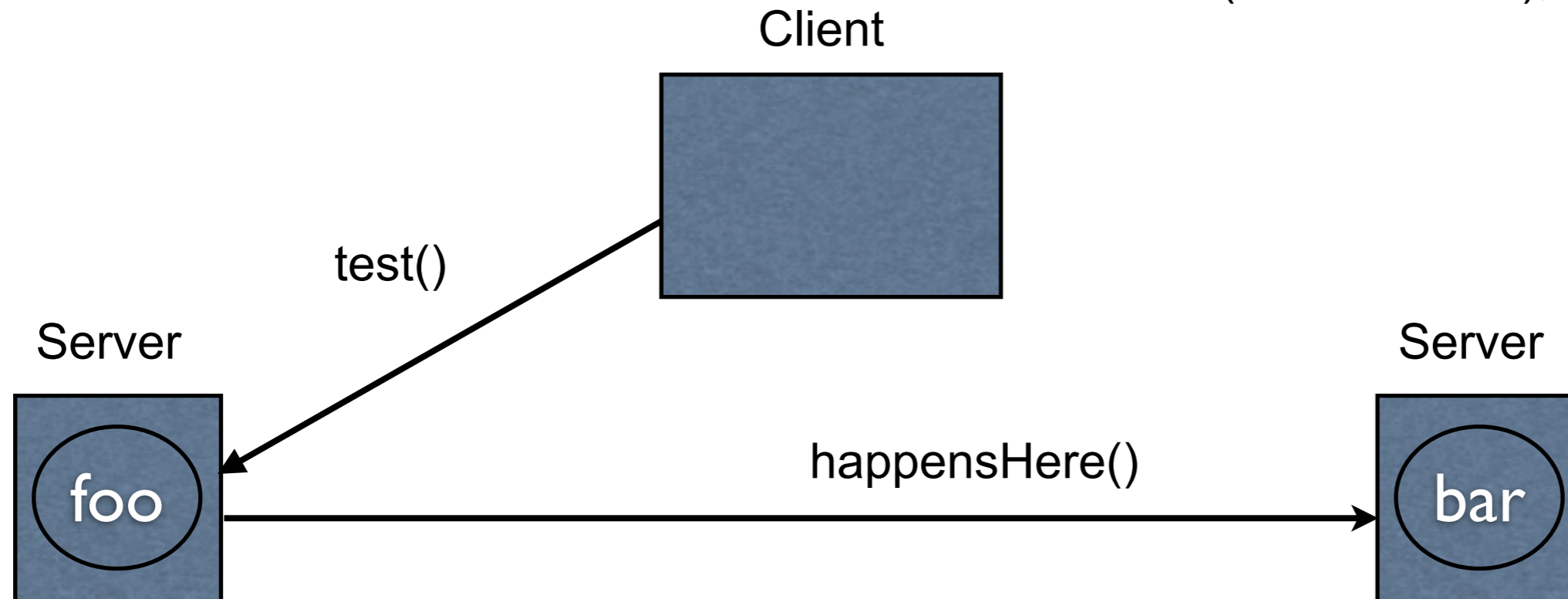
```
Foo remote = getRemoteObject();  
Vector b = remote.sample(aVector);
```



Passing Remote References

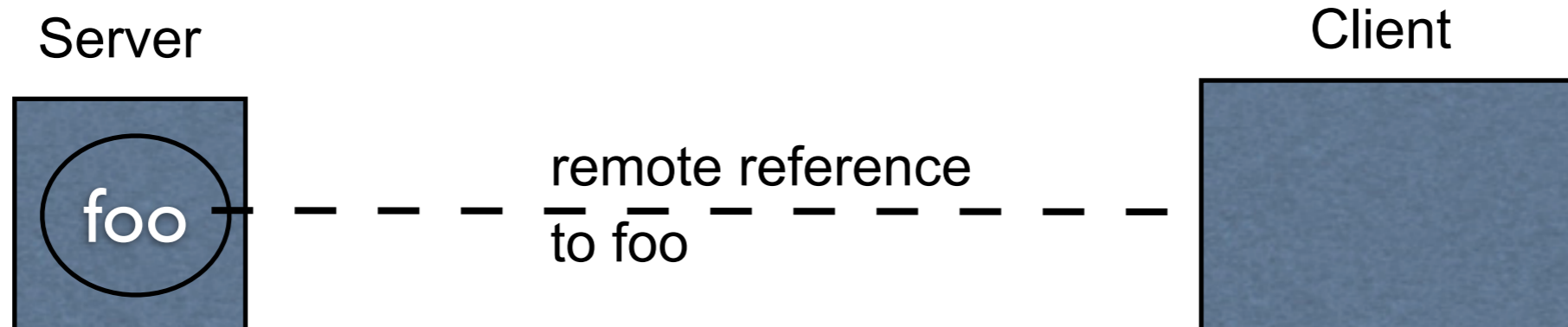
```
public class Foo {  
    public void test(Bar what) {  
        what.happensHere();  
    }  
}
```

```
Foo remote = getRemoteObject();  
Bar otherRemote = getOtherRemote();  
remote.test(otherRemote);
```



Remote References & Garbage Collection

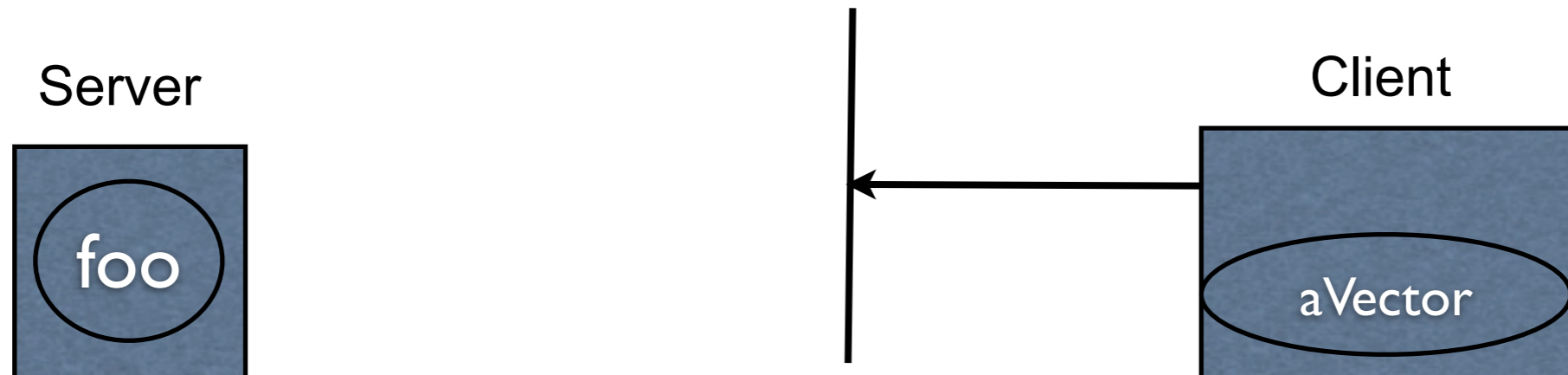
When can foo be garbage collected?



What happens when the network is disrupted?

```
public class Foo {  
    public Vector sample( Vector input, int data ) {  
        Integer remoteInt = new Integer( data );  
        input.addElement( remoteInt );  
        return input;  
    }  
}
```

```
Foo remote = getRemoteObject();  
Vector b = remote.sample(aVector);
```



Some Background

Finding servers - Client-Dispatcher-Server Pattern

Making remote method calls - Proxy Pattern

Security - Java Security features

Client-Dispatcher-Server Pattern

Problems

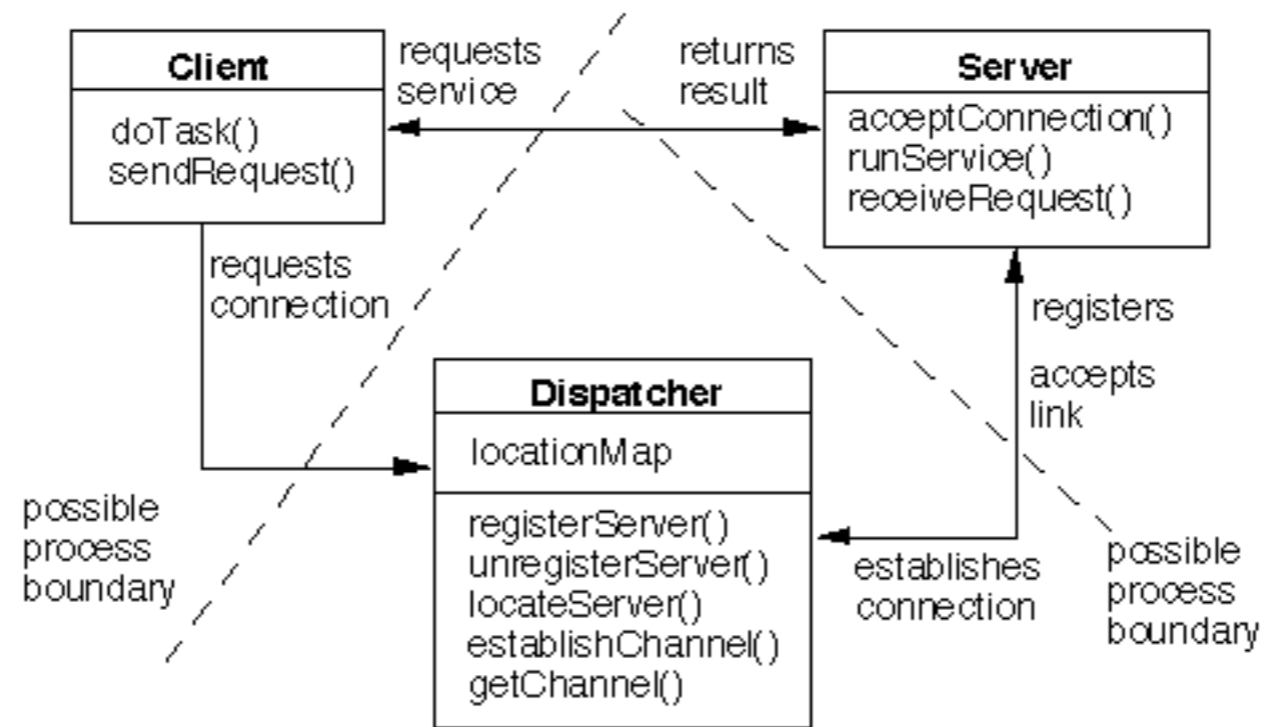
How to find the servers?

Clients should be able to use a server independent its location

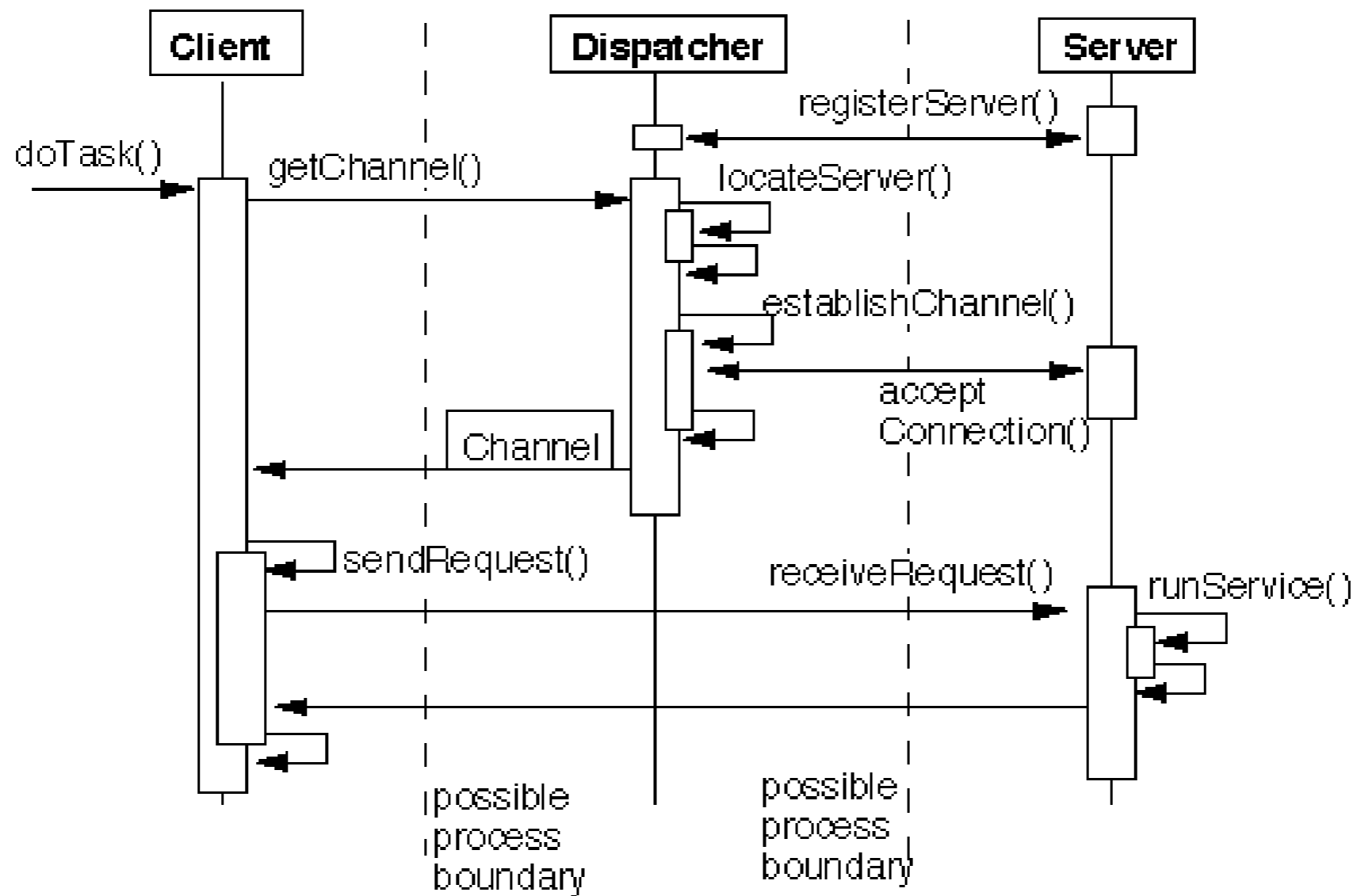
Separate functional core of a client from network code

Solution

Provide a dispatcher components to act as an intermediate layer between clients and servers



Dynamics



Variant - Distributed Dispatchers

Each machine has its own dispatcher

When a client needs to connection to a server on a remote machine:

The client connections to its local dispatcher

The local dispatcher connects to the remote dispatcher

The remote dispatcher returns the server communication channel to the local dispatcher

The local dispatcher returns to server communication channel to the client

The client now uses the communication channel to interact with the remote server

Variant - Communication Managed by clients

The dispatcher returns the physical server location to the client

The client manages all the communication with the server including opening the communication channel

Variant - Client-Dispatcher-Service

Clients request a service from the dispatcher

The dispatcher looks up all servers that provide that service and opens a communication channel to one of those servers

Consequences

Benefits

Exchangeability of servers

Location and migration transparency

Re-configuration

Fault Tolerance

Liabilities

Dispatcher overhead

Sensitivity to change in the interfaces of the dispatcher

Know Uses

Sun's RPC

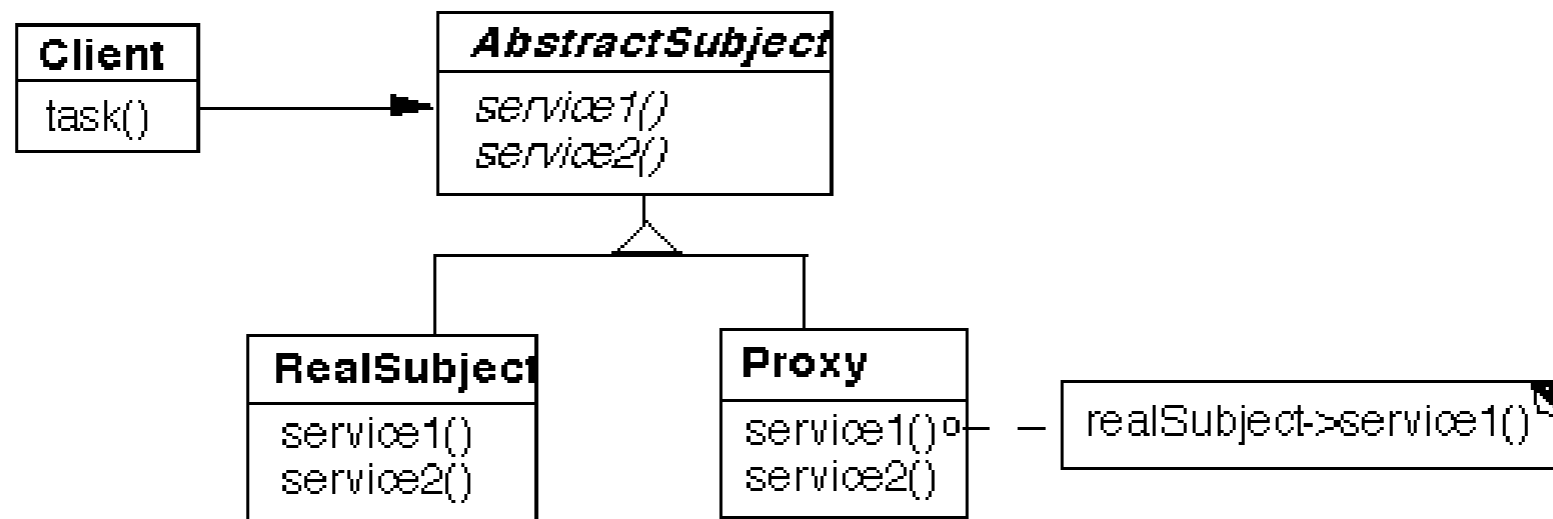
Java's RMI

OMG Corba

Proxy Pattern

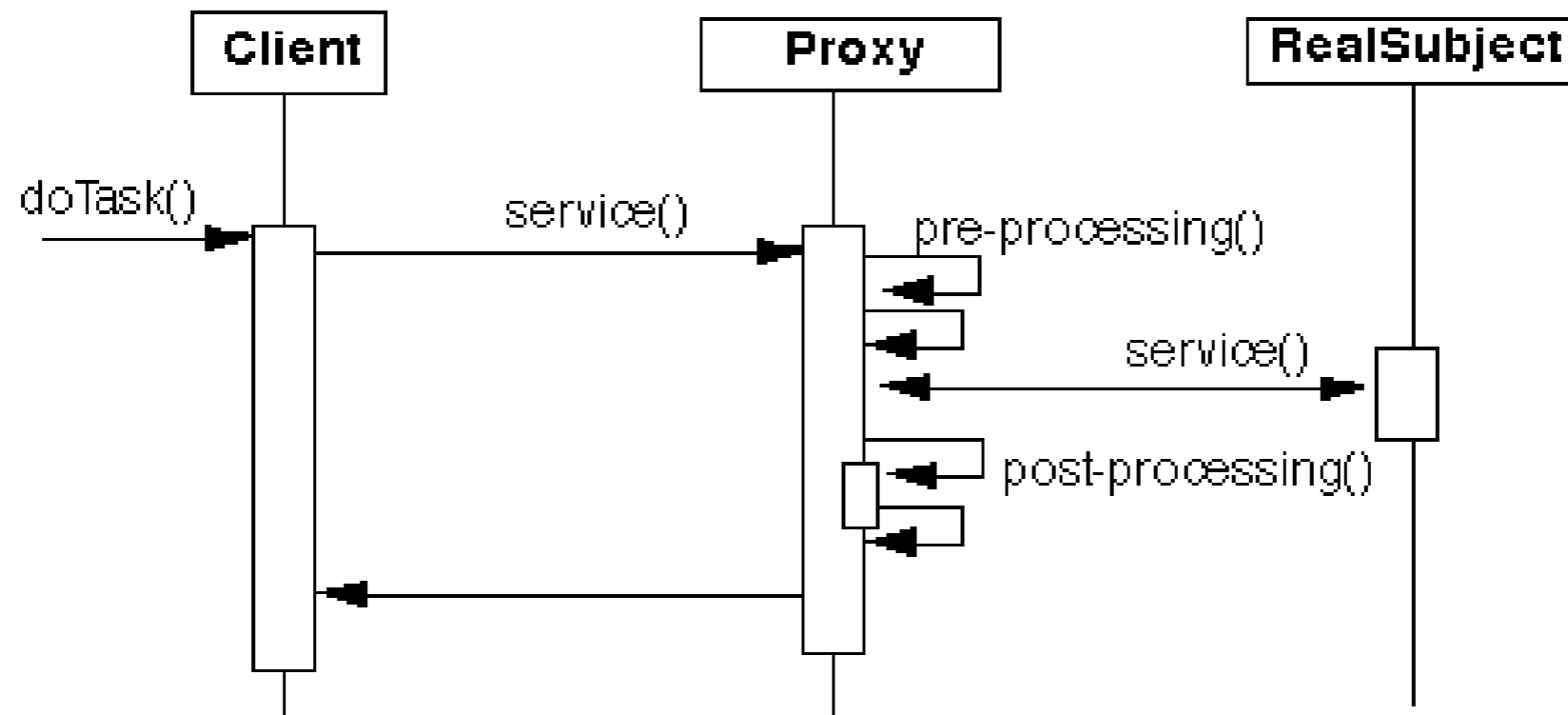
proxy n. pl prox-ies

The agency for a person who acts as a substitute for another person, authority to act for another



Proxy contains a reference to original object, so proxy can forward requests to the original object

Dynamics

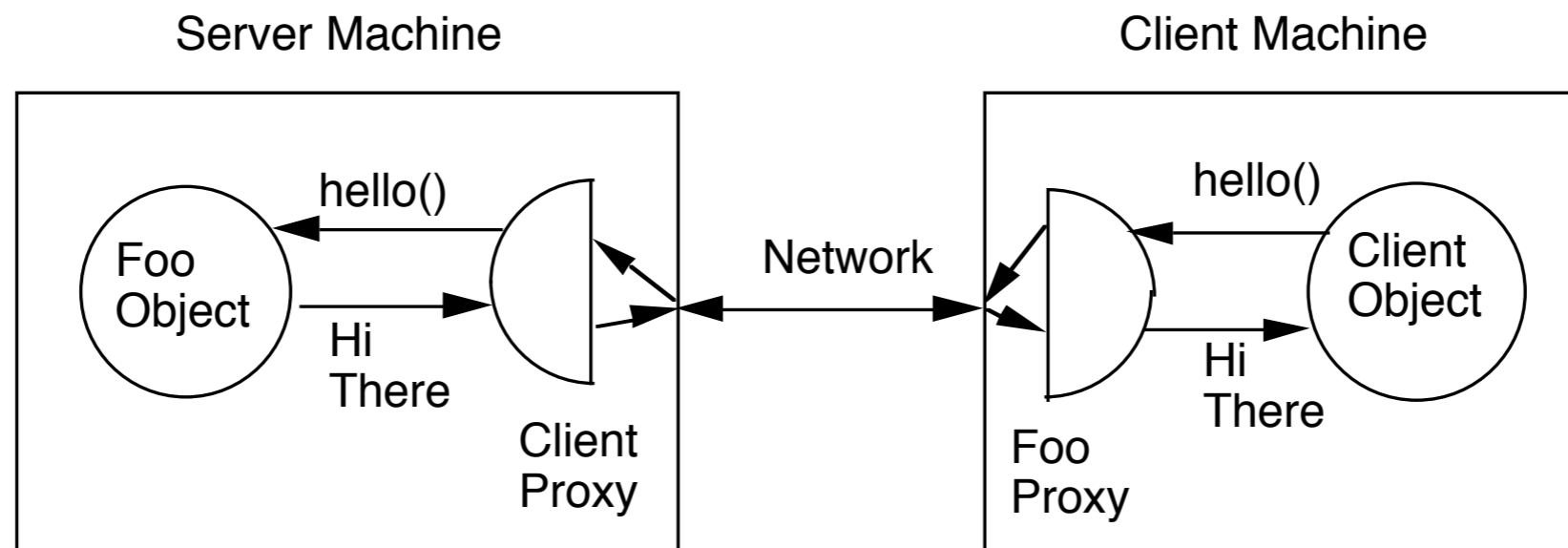


Remote Proxy

The actual object is on a remote machine (remote address space)

Hide real details of accessing the object

Used in CORBA, Java RMI



Java Access Control

Controlling what Java code can do

All Java programs subject to security checks

Fine-grained access control

Easily configurable security policy

Uses

- Security Manager

- Permissions file

Security Manager

Checks all attempts to use system resources

Only one security manager per program

In java.io.File class:

```
public boolean canRead() {  
    SecurityManager security = System.getSecurityManager();  
    if (security != null) {  
        security.checkRead(path);  
    }  
    return canRead0();  
}
```

In constructor for Socket:

```
SecurityManager security = System.getSecurityManager();  
if (security != null) {  
    security.checkConnect(address.getHostAddress(), port);  
}
```

Permissions

Indicate what is allowed and by who

Placed in java.policy file

Global policy file

Individual user policy file

General Format

```
grant [signedBy "signer_names"] [, codeBase "URL" ]{  
    permission permission_class_name "target_name", "action"  
        [, signedBy "signer_names"];  
    ....  
    permission permission_class_name "target_name" , "action"  
        [, signedBy "signer_names"];  
};
```

Existing Permissions

java.security.AllPermission
java.security.SecurityPermission
java.security.UnresolvedPermission
java.awt.AWTPermission
java.io.FilePermission
java.io.SerializablePermission
java.lang.reflect.ReflectPermission
java.lang.RuntimePermission
java.net.NetPermission
java.net.SocketPermission
java.sql.SQLPermission
java.util.PropertyPermission
java.util.logging.LoggingPermission
javax.net.ssl.SSLPermission
javax.security.auth.AuthPermission
javax.security.auth.PrivateCredentialPermission
javax.security.auth.kerberos.DelegationPermission
javax.security.auth.kerberos.ServicePermission
javax.sound.sampled.AudioPermission

Fine Grain Control

FilePermission

On a per file basis can indicate type of access is allowed

read

write

execute

delete

Different parts of program can have difference access

SocketPermission

General Format:

```
grant {  
    permission java.net.SocketPermission "host", "actions";  
};
```

where:

host = (hostname | IPaddress) [:portrange]

portrange = portnumber | -portnumber |

portnumber-portnumber | portnumber-

actions = action | action,actions

action = accept | connect | listen | resolve

Host

host can be:

- a DNS name

- “localhost” for the local machine

- a DNS name with wildcard character “*”

Java 1.5 General Policy File

```
grant codeBase "file:${java.ext.dirs}/*" {
    permission java.security.AllPermission;
};
grant {
    permission java.lang.RuntimePermission "stopThread";
    permission java.net.SocketPermission "localhost:1024-", "listen";
    permission java.util.PropertyPermission "java.version", "read";
    permission java.util.PropertyPermission "java.vendor", "read";
    permission java.util.PropertyPermission "java.vendor.url", "read";
    permission java.util.PropertyPermission "java.class.version", "read";
    permission java.util.PropertyPermission "os.name", "read";
    permission java.util.PropertyPermission "os.version", "read";
    permission java.util.PropertyPermission "os.arch", "read";
    permission java.util.PropertyPermission "file.separator", "read";
    permission java.util.PropertyPermission "path.separator", "read";
    permission java.util.PropertyPermission "line.separator", "read";
    permission java.util.PropertyPermission "java.specification.version", "read";
    permission java.util.PropertyPermission "java.specification.vendor", "read";
    permission java.util.PropertyPermission "java.specification.name", "read";
    permission java.util.PropertyPermission "java.vm.specification.version", "read";
    permission java.util.PropertyPermission "java.vm.specification.vendor", "read";
    permission java.util.PropertyPermission "java.vm.specification.name", "read";
    permission java.util.PropertyPermission "java.vm.version", "read";
    permission java.util.PropertyPermission "java.vm.vendor", "read";
    permission java.util.PropertyPermission "java.vm.name", "read";
};
```

On Rohan

/opt/jdk1.5.0/jre/lib/security/java.policy

Loading Policy Files

System wide Policy File

Located in {JavaHome}/jre/lib/security/ java.policy

Loaded before user policy file

User Policy File

Default location is {UserHomeDirectory}/.java.policy

Command line policy file

```
java -Djava.security.policy=policyFileURL anApp
```

Load only Command line policy

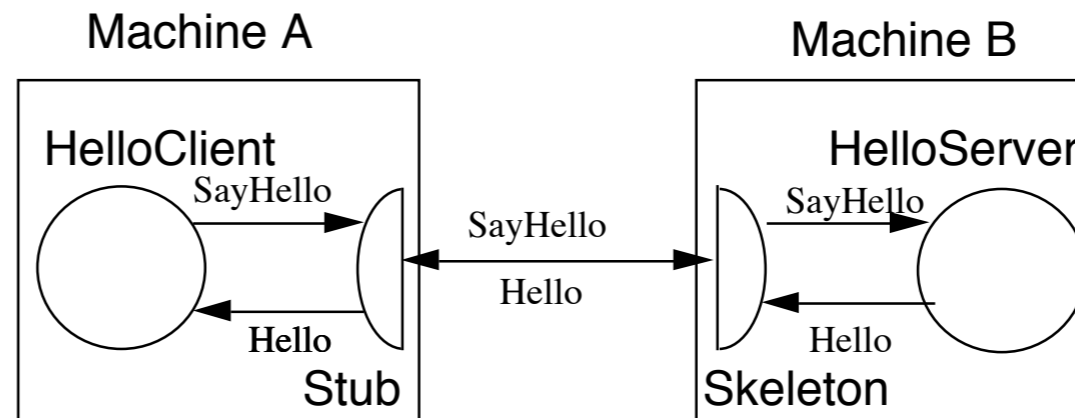
```
java -Djava.security.policy==policyFileURL anApp
```

RMI - Hello World Example

Implement a server with the method sayHello()

Parts needed

- Hello interface
- Client code
- Server Code
- Permission file
- rmiregistry
- Proxy classes



The Remote Interface

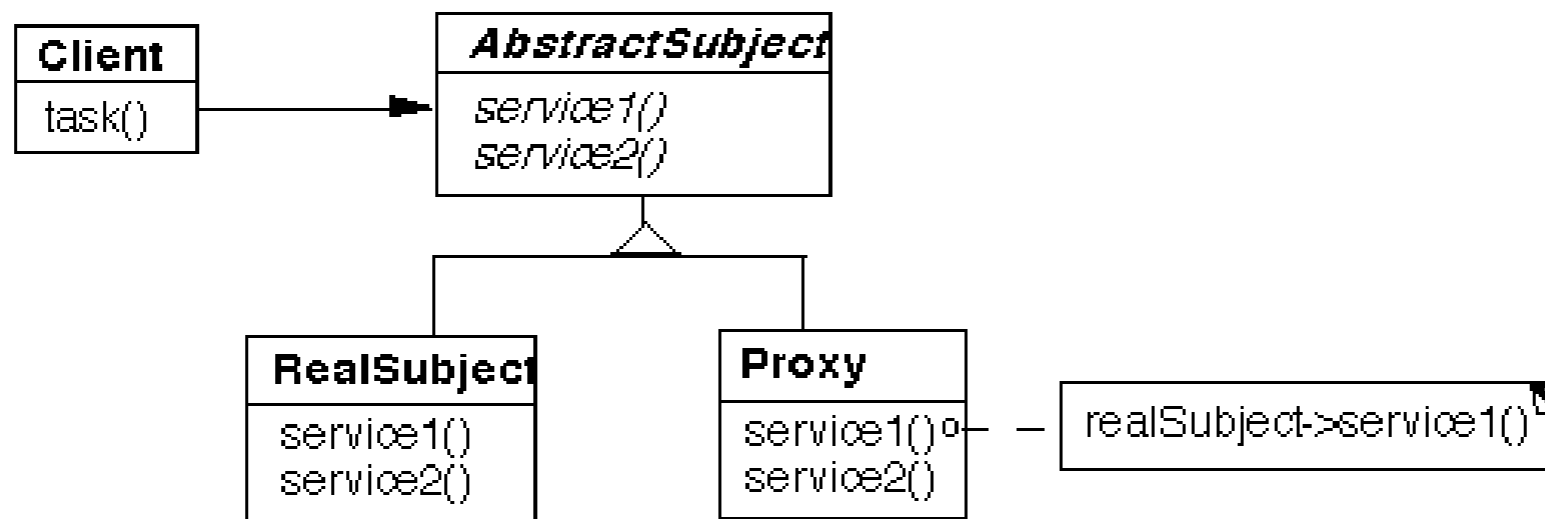
```
public interface Hello extends java.rmi.Remote
{
    String sayHello() throws java.rmi.RemoteException;
}
```

Why The Interface?

Client interacts with proxy for the server

The proxy implements the Hello interface

Client does not know difference



HelloServer

```
import java.net.InetAddress;
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.RMISecurityManager;
import java.rmi.server.UnicastRemoteObject;

public class HelloServer extends UnicastRemoteObject implements Hello {

    public HelloServer() throws RemoteException { }

    public String sayHello() { return "Hello World from " + getHostName(); }

    protected static String getHostName() {
        try {
            return InetAddress.getLocalHost().getHostName();
        }
        catch (java.net.UnknownHostException who) {
            return "Unknown";
        }
    }
}
```

HelloServer Continued

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

    try {
        // Create and register the server object
        HelloServer serverObject = new HelloServer();
        Naming.rebind("rmi://127.0.0.1/HelloServer", serverObject );

        // Signal successful registration
        System.out.println("HelloServer bound in registry");
    }
    catch (Exception e) {
        System.out.println("HelloServer err: ");
        e.printStackTrace();
    }
}
}
```

Why the Null Server Constructor?

```
public HelloServer() throws RemoteException { }
```

HelloServer's parent class constructor throws RemoteException

Since a constructor calls its parent class constructor, RemoteException can be thrown in HelloServer's constructor

Why RMI Security Manager?

```
System.setSecurityManager(new RMI Security Manager());
```

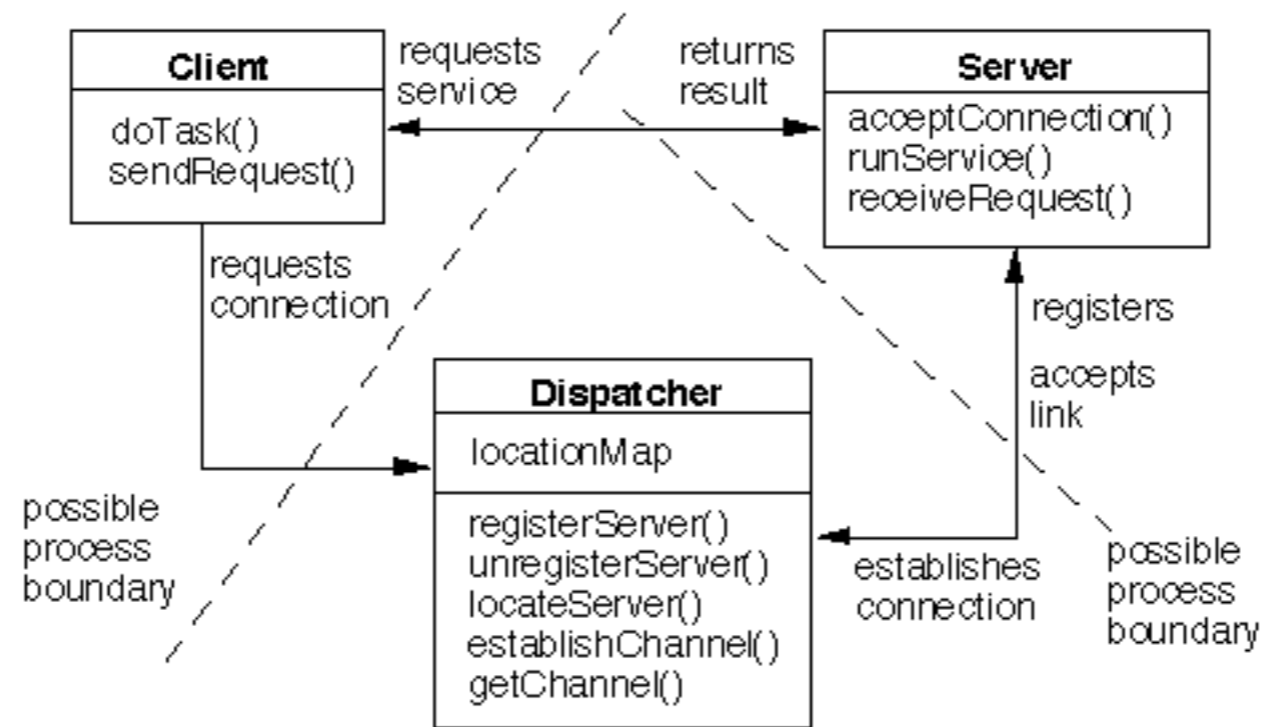
This is optional

RMI Security Manager allows the server to download code from remote sites

Unless you know what you are doing don't use this

What is Naming.rebind()?

Registers server object with dispatcher (rmiregistry)



HelloClient

```
import java.rmi.*;
import java.net.MalformedURLException;

public class HelloClient {
    public static void main(String args[]) {
        try {
            Hello remote = (Hello) Naming.lookup( "rmi://127.0.0.1/HelloServer");

            String message = remote.sayHello();
            System.out.println( message );
        }
        catch ( NotBoundException error){
            error.printStackTrace();
        }
        catch ( MalformedURLException error){
            error.printStackTrace();
        }
        catch ( UnknownHostException error){
            error.printStackTrace();
        }
        catch ( RemoteException error){
            error.printStackTrace();
        }
    }
}
```


Compiling The Example

Server Side

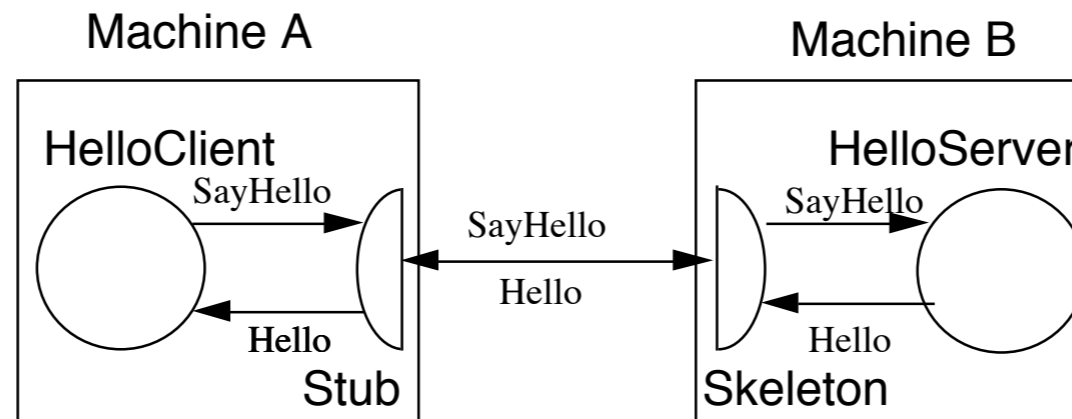
Compile source code

```
javac Hello.java HelloServer.java
```

Generate Stub (Client side proxy)

```
rmic HelloServer
```

Stub & Skeleton



Stub

Server proxy

Used by client

Java 1.5 can auto-generate

if server is subclass of UnicastRemoteObject

Skeleton

client proxy

Used by server

Generated automatically

Generated Stub class

```
public final class HelloServer_Stub
    extends java.rmi.server.RemoteStub
    implements Hello, java.rmi.Remote {
    private static final long serialVersionUID = 2;
    private static java.lang.reflect.Method $method_sayHello_0;

    static {
        try { $method_sayHello_0 = Hello.class.getMethod("sayHello", new java.lang.Class[] {});
        } catch (java.lang.NoSuchMethodException e) {
            throw new java.lang.NoSuchMethodError(
                "stub class initialization failed");
        }
    }

    public HelloServer_Stub(java.rmi.server.RemoteRef ref) { super(ref);}

    public java.lang.String sayHello() throws java.rmi.RemoteException {
        try {
            Object $result = ref.invoke(this, $method_sayHello_0, null, 6043973830760146143L);
            return ((java.lang.String) $result);
        } catch (java.lang.RuntimeException e) { throw e;
        } catch (java.rmi.RemoteException e) { throw e;
        } catch (java.lang.Exception e) {
            throw new java.rmi.UnexpectedException("undeclared checked exception", e);
        }
    }
}
```

Compiling Client Code

Copy server stub (HelloServer_Stub.class) to client machine

Compile client code

```
javac HelloClient.java
```

Running the Server

Step 0. Install a policy file for socket permissions

You only need to do this once

Place the following in a file called “.java.policy” in your home directory.

```
grant
{
  permission java.net.SocketPermission "*:1024-65535",
    "connect,accept,resolve";
  permission java.net.SocketPermission "*:1-1023",
    "connect,resolve";
};
```

Running the Server

Step 1. Insure that the RMI Registry is running

For the default port number

```
rmiregistry &
```

For a specific port number

```
rmiregistry portNumber &
```

Must be running on same machine as server

Running the Server

Step 2. Run the server

```
java HelloServer &
```

Actually this just registers the server object with the rmiregistry

Running the Client

```
java HelloClient
```