CPS221 Lecture: The Application Layer and the Client-Server Model

last revised 10/5/14

Objectives

1. To introduce the client-server model and its major variants (thin, thick client; three layer)
2. To discuss some key application layer protocols: DHCP, DNS, HTTP
3. To introduce the idea of remote objects / java RMI
4. To discuss the basic idea of Web 2.0

Materials:

1. Projectable of three variants of client-server model
2. Executable of date.cgi modified to run on laptop
3. Projectable of source code for date.cgi
4. Projectables: formGET.html, formPOST.html, hiGET.html, hiPOST.html
5. Above installed in www.cs.gordon.edu/~bjork
6. Projectables of original and distributed version of registration system
7. Projectables of apparent and actual implementations of a remote object
8. Ability to demo RMI version of registration system - server on workstation, client on laptop
9. Projectable of XML description of book
10. Projectable of place of Content Management System (Casad Figure 19.1)

I. Place of the Application Layer in TCP/IP

A. As you recall, the ISO/OSI networking model had a total of seven layers - four corresponding to layers we have already discussed and three corresponding to layers above the Transport Layer. What were they? What was the basic function of each?

ASK

1. The Session Layer deals with matters like authentication, which are needed by many kinds of applications (e.g. email, ftp, some web sites) In the ISO/OSI model, this is factored into a separate layer because

    a) Not all applications or uses of an application like a web browser need the notion of a session.

    b) But, facilities like authentication are useful for many different kinds of application

2. The Presentation Layer deals with matters like encryption and various character formats (e.g. ASCII vs EBCDIC). In the ISO/OSI model, this is factored into a separate layer because

    a) Not all applications or uses of an application like a web browser need encryption

    b) But, facilities like encryption and character code management are useful for many different kinds of application

3. The Application Layer deals with the particulars of an application (mail, file transfer, web browsing, etc.)

B. In TCP/IP, all of these functions are subsumed in the Application layer. As a result, applications that need services provided by the other two ISO/OSI layers need to incorporate them in the application layer

    This may be handled as it is in http, where

    1. Encryption is provided for by a variant of the basic http protocol known as https.

    2. Session management is handled by add-on products, of which there are many in the web world (e.g. Cold Fusion used at Gordon)

II. **Models for Applications**

A. Many applications are structured on the basis of the client-server model of computation.

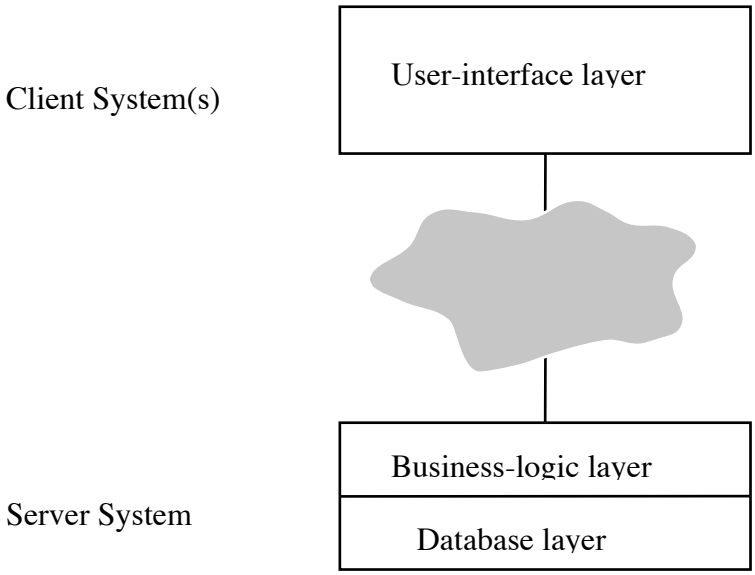
1. In the simplest case, a client-server system consists of a server system and (one or more) client subsystems. For example, a web browser relates as a client to a web server; the mail program running on a personal computer acts as client to a mail server, etc.

2. More complex systems can be understood in terms of a layered model: a user-interface layer, a business-logic layer, and a database layer. (These are layers within the application layer - to be distinguished from the lower networking layers)

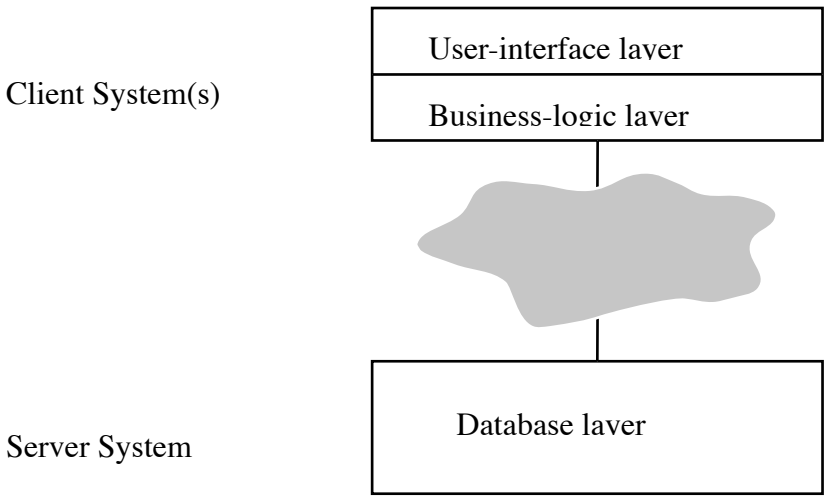
   For example, many e-commerce systems are set up this way: the user interface layer is a web page (perhaps with embedded javascript) viewed by a web browser; the business-logic layer is the software that provides information in response to user requests and processes orders, and the database layer stores information about the products and records user orders. There are three different ways these might be distributed:

   PROJECT

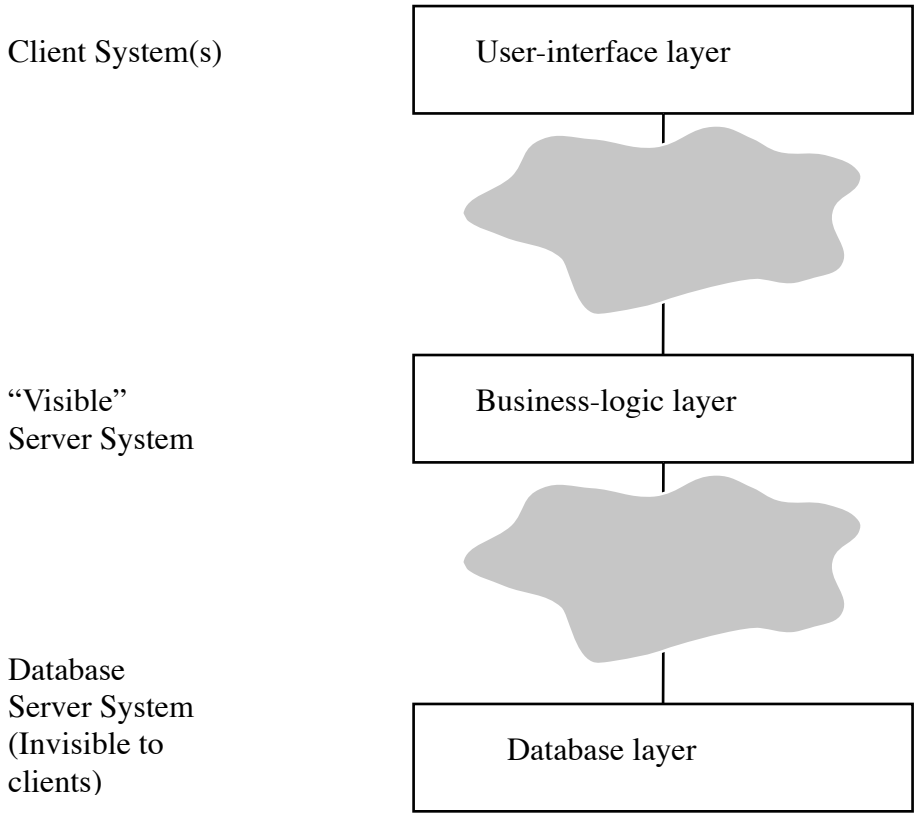
   a) An approach often used by e-commerce systems (the so-called "thin client" approach)



b) The so-called "thick client" approach, used when it is desirable to install the business logic software on the client system (This wouldn't work for e-commerce, of course, but is sometimes used for specialized applications)



c) A three-layer approach that can also be used as an alternative to the thin client approach. (Note that the client would never see any difference between this approach and the thin-client approach; in fact, many e-commerce systems are in fact built this way)



3. While the client-server architecture is most commonly seen in distributed systems, it can also be used for software systems running on a single computer - e.g. a program that uses a relational database is often structured as a client relating to a separate database server program running on the same computer. (In fact, a database server program running on a computer may simultaneously be serving several different clients on that same machine.)

B. An alternative architecture for distributed systems is a Peer-to-Peer architecture in which there is no designated server as such. Instead, each participating system can function either as a client, or as a server to some other system.

   (We will not develop this here)

III. **Models for Servers**

A. There are several different models that can be used for server systems.

B. The simplest model is to have a single server, running a single thread. Incoming requests are queued up for this thread, which processes them one at a time in the order received.

1. This is the simplest model, and is viable for situations in which requests are independent of one another.

   This is the case, for example, with simple http. (Each page request is handled by a separate TCP connection that is set up when the page is requested and is torn down after the page is sent.)

2. It quickly becomes complex in situations where multiple requests from a given source are related to each other.

   For example, this is the case when the notion of a "session" is present - e.g. when one logs in to an account and subsequent requests are related to that login.

C. An alternative is to use a multithreaded server, so that each user is served by a separate thread that is created when the session is established.

1. The thread is responsible for maintaining state information about that session, which can greatly reduce complexity.

2. Use of a multithreaded server can also improve performance even when requests are independent of one another.

   To see why, suppose that request requires that the thread perform an operation like a disk read that must block until the operation completes.

a) With a single-threaded server, all other requests have to wait until the server thread finishes with the current request.

b) With a multi-threaded server, the threads serving other requests can continue in parallel with the wait.

c) Note that, since there is a limited number of CPU cores, the multi-threaded server cannot do computation on behalf of more threads at a time than there are cores - just that it doesn't have to be blocked when one thread is blocked. (But on a multicore CPU even this will be an advantage).

   Illustration: the main street my street comes out onto (Elliott Street) has only one lane of traffic in each direction, so only one car at a time can pass through the intersection in a given direction.

   However, the westbound direction has a left-turn lane, so if a car is blocked waiting to turn left other cars can still go in that direction. However, there is no left-turn lane in the eastbound direction, so if an eastbound car wants to turn left all remaining eastbound traffic is blocked until the turn is completed.

D. When the rate of requests for service exceeds the rate at which the server can process requests, it is possible to use multiple, basically identical servers. In this case, the IP of the service actually refers to a proxy server, that simply parcels the requests out to different the actual servers in a way that balances the load.

IV. Important Application Layer Protocols

A. There are a large number of important application layer protocols in the Internet world.

B. We will focus on just one, which is perhaps the most familiar of all: http and its relative: https.

1. There are actually three approaches:

a) Static html - html text resides as file on server, fetched on demand by an http request.

b) Dynamic html - html text is constructed by server on demand based on script/template stored on server, typically with some parameters specified by the http request.

Example: Demo a request for weather information on yahoo - note how location is specified in URL. Now change locations - note similar appearance of page.

c) Active html - html text contains code to be executed on the client (e.g. JavaScript)

d) The latter two may be combined to yield Dynamic+Active html

2. We will only discuss dynamic html here

a) Static html is straight-forward - many of you are already familiar with it.

b) Active html is discussed in the Internet Programming course - as well as a lot more about dynamic html

3. There are actually two approaches to dynamic html

a) An html page can contain embedded code that is executed by the server, so that what the client receives is a mixture of html stored in the file plus html created by the server.

(1) This is called server side scripting

(2) It requires special support in the server. There are a number of variants, some supported by commercial servers and some by open source servers (e.g. ASP or JSP or Cold Fusion ...)

b) Html can be produced by a separate program (or shell script) that is run by the server - typically as a result of a special URL.
- A simple example: date.cgi created by Jon Senning

(1) Execute on Linux

(2) Demo via web: www.cs.gordon.edu/~senning/date.cgi

(3) PROJECT code - Some things to note

(a) Must specify full path for any programs run (web server has no default path)

(b) Must output a header, blank line, then actual html (web server expects this)

(c) Must be executable by username server runs under

c) This example didn't need any parameters - but many usages do.

d) Example: PROJECT one variant of hi.cgi

There are two basic approaches - Both of which typically depend on form code on a web page accessed by client

(1) GET method - parameters are appended to URL in GET request.

(a) This can be done automatically by code associated with a form button,

(b) Parameters can be in a URL on the page.

(c) CGI program accesses in one of two ways depending on how passed - we won't get into details.

(d) Example: www.cs.gordon.edu/~bjork/formGET.html; enter name - note parameter in URL.

(e) PROJECT formGET.html

(2) POST method - CGI program reads parameters from standard input, as lines that will be of the form name=value

(a) PROJECT formPOST.html

(b) PROJECT hi.POSTcgi

(c) Execute hiPOST.cgi - type line saying name=russ

(d) Demo on server www.cs.gordon.edu/~bjork/formPOST.html

C. There are a number of other application layer protocols that are frequently encountered:

1. Telnet / ssh
2. ftp / sftp
3. Various protocols used with email: smtp, pop (currently 3), imap, mime

V. **Web 2.0**

A. The term Web 2.0 has appeared as a sort of buzzword, not to describe any change to the basic web protocols, but rather to describe a change in the way the web is used.

1. The original web model saw users as consumers of information provided by servers.

2. The Web 2.0 model sees users as also being producers of information that can be shared with others via the web - "prosumers".

3. Examples of Web 2.0-style applications?

   ASK

B. On the server side, Web 2.0 applications may make use of some sort of Content-Management System (CMS) on top of a database to allow the user to update content to be seen by other users.

   PROJECT

C. Though Web 2.0 does not call for any change to basic Web protocols, it does rely heavily on client-side technologies that build on these protocols. Sometimes the term "AJAX" is used - which stands for "Asynchronous Java Script and XML"

1. Web 2.0 applications typically make use of client-side scripting - what we earlier called Active html.

2. Web 2.0 applications typically are asynchronous - they perform updates "on the fly" rather than using the strict request page - wait for page - display page model originally used by the web.

3. Web 2.0 applications may make use of notations like XML or the more recent JSON (Java-Script object notation).

   a) Standard html focuses on how information is displayed. Tags like the following are common in html:

   <p>

   <hr>
   <h 1-6>
   <table>

   b) Notations like XML and JSON focus on what information means, with cascading style sheets used to specify how it is displayed. For example, the following is an XML description of the book we are using

   PROJECT

```
<book>
  <title>Sam's Teach Yourself TCP/IP</title>
    <authorJoe Casad</author>
    <isbn>978-0-672-33571-6</isbn>
    ...
</book>
```

D. A related idea is the notion of a web service.

1. Here, the idea is to use web protocols for more general machine-to-machine communication beyond simple web-browsing

2. A web service is an information source that provides information (typically encoded in a form like XML or JSON) on request from a client other than a standard web browser.

   Example: Calebe Maciel's senior project

VI. **Remote Objects/RMI**

A. Finally, we want to look at an approach that can be used in the application layer that does not necessarily make use of the Internet (though most implementations are built on top of TCP/IP): Remote Objects.

B. The basic idea is to create a situation in which a program running on one machine can access objects actually running on another machine. There are several approaches to doing this:

1. Using an object request broker (ORB).

a) An object request broker is a piece of software that needs to be running on both machines. It allows a client program running on one machine to invoke methods of an object in a program running on a server machine. In effect, once access to a remote object has been established, a client program can interact with the remote object the same way it interacts with a local object - the location is not a factor in how it uses an object. (The remote object looks just like a local object to the program using it.)

(1) There is an industry standard for ORBs called CORBA (Common Object Request Broker Architecture.)

(2) Thus, when we talk about using an ORB, we usually mean using an ORB that complies with the CORBA standards.

b) The major advantage of CORBA is that it is a cross-language, cross-platform industry standard.

(1) Because CORBA is an industry standard, CORBA-compliant ORBs from different software vendors can inter operate with each other just as well as ORBs from the same vendor. Thus, a client and a server do not have to be running the same ORB software.

(2) CORBA is cross platform - Any CORBA-compliant ORB on one machine can communicate with any CORBA-compliant ORB on another machine, regardless of the machine or operating system.

(3) CORBA is cross language - A CORBA-compliant ORB allows a program written in (say) Java to access objects in a program written in (say) Smalltalk on another machine.

(a) This is accomplished by specifying the interface to an object using a language-neutral Interface Definition Language (IDL).

(b) A CORBA-compliant ORB includes IDL compilers which translate an IDL specification into appropriate language-specific interfaces and implementation skeletons.

(c) Note that the Java library includes a package org.omg that provides the Java side of the link to a CORBA-compliant ORB on the same computer, which can then be used to link to ORB's on other computers. (The org.omg package is not an ORB, it provides facilities for using an ORB.)

c) A major disadvantage of CORBA is that it is complex and CORBA-compliant ORBs are fairly expensive.

d) We will not discuss CORBA further.

2. Using Java RMI (Remote method invocation).

a) RMI provides capabilities similar to those provided by an ORB, but only to Java programs.

b) A major advantage of RMI for us is that it is integrated into the Java language and libraries

c) A major disadvantage of RMI is that it is integrated into the Java language and libraries!

d) We will use RMI in a lab (which will be something of a "cookbook:" lab., because it is readily available to us. If you understand RMI, you will find it easier to understand approaches like CORBA or .NET, because there are significant similarities (though obviously there are important differences as well).

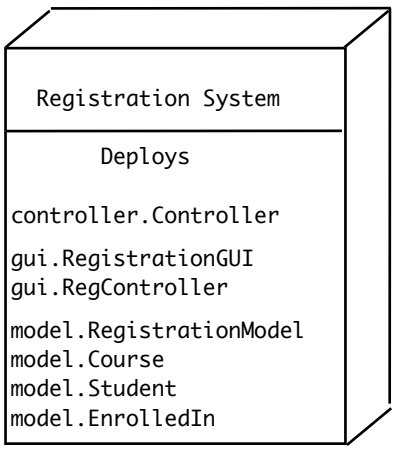
3. Numerous other alternatives we won't discuss:

a) Microsoft proprietary technologies: DCOM and .NET remoting

b) Remote Procedure call (Sun RPC). [Actually, RMI can be thought of as the Java version of RPC]

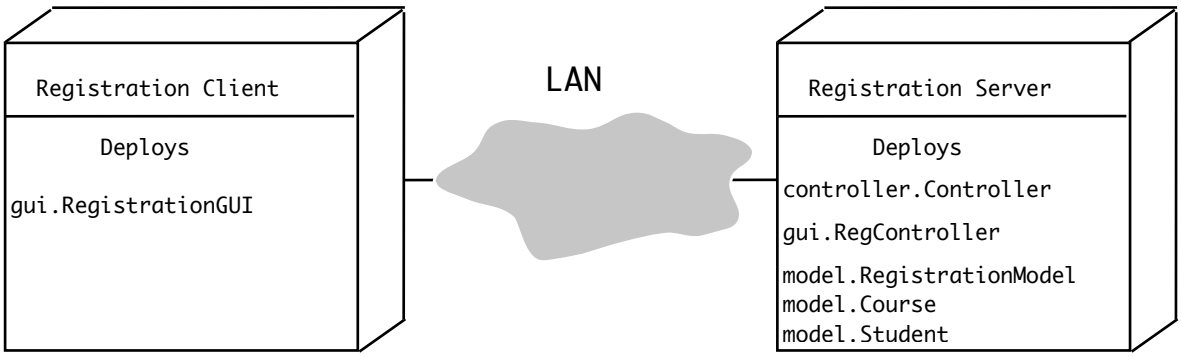
C. An example (using RMI): Suppose we wanted to use rmi to convert the registration system you used in a couple of CPS122 labs into a client/server version.

1. The following deployment diagrams show the structure of the original version



PROJECT

2. Our goal is to create a distributed version that looks like this:



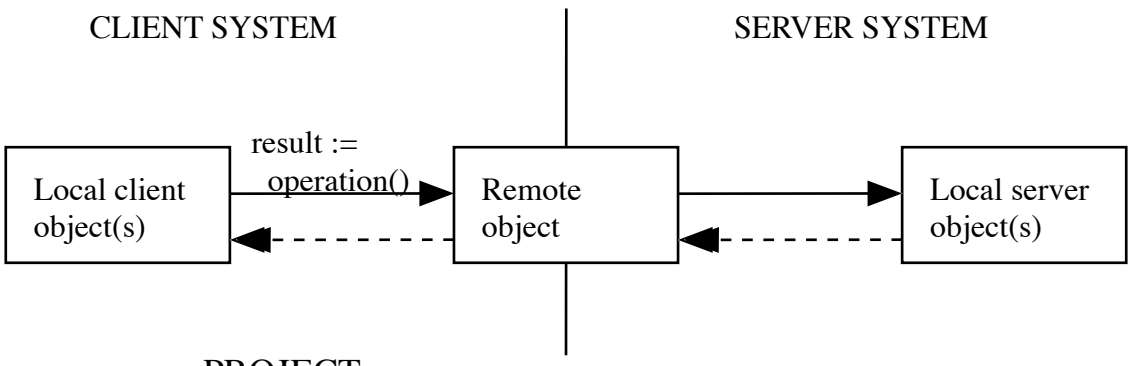
PROJECT

a) What is the only class that appears on both the client and the server nodes?

ASK

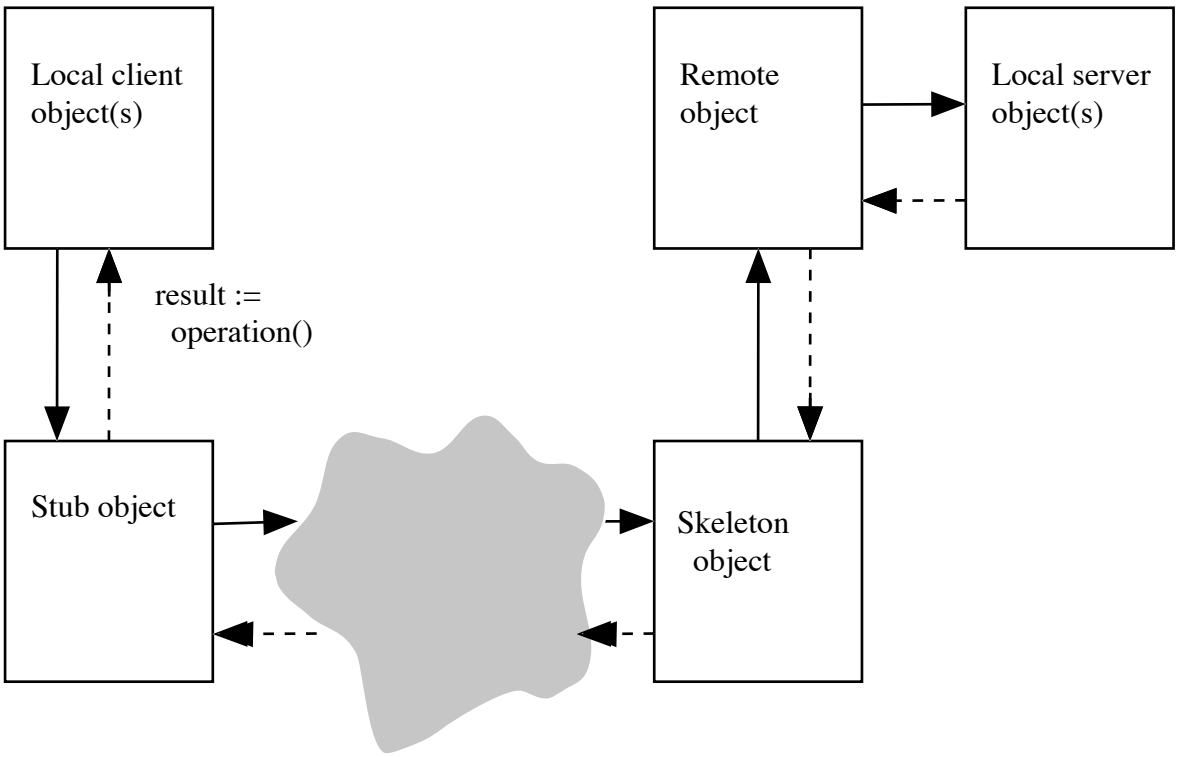
b) Actually, `gui.RegController` is an interface that describes a remote object - an object that "lives" on the server, but can be accessed by objects that "live" on the client (in this case, the GUI)

c) What we will create is something that looks like this:



PROJECT

d) This will actually be implemented like this:



PROJECT

e) This is an instance of the proxy pattern. There exist two proxy objects - one each on the client and the server.

(1) The client proxy is called a stub in RMI.

(2) The server proxy is called a skeleton in RMI.

(3) When the GUI on the client needs to access the remote object (the controller), it executes a method of the stub. The stub, in turn sends a message over the network to its partner on the server (the skeleton). The skeleton forwards the request to the real controller, receives a result, and passes it back over the network to the stub. The stub, in turn, returns the result to the GUI on the client.

3. *DEMO:* The RMI version of the registration system

a) The Server will run on a workstation

(1) `login to micah`

(2) `cd cps221/RMIExampleServer/build/classes`

(3) `rmiregistry &`

(4) `cd ../..`

(5) `java -cp build/classes registrationsystem.controller.Controller`

b) The client will run on laptop: DEMO

(1) `java -jar RMILabClient.jar micah.cs.gordon.edu`

(2) Repeat the above to run a second copy - hence a second window.

(3) Observe how changes made in one window are visible in the other - two copies of the client both accessing the same server

(4) Ask a student to run from USB stick

4. One thing that is missing from the our discussion thus far is (and is not present in the version of the code we are using) is *synchronization*.

a) Each time the rmi registry is contacted by a client, it creates a separate thread.

b) Thus, if our GUI is running on two or more different systems accessing the same server, two different threads could perform conflicting operations at the same time.

c) This could be handled by explicit synchronization in the server code. How should this be done, and where should it go?

ASK

Each of the methods that alters the database (`doEnroll()`, `doDrop()`, `doGrade()`) needs to be synchronized. Since they all share the same controller object, this would ensure that two threads don't perform an operation that alters the database at the same time.