Purpose	Learn to apply a framework in developing an application
Prerequisites	Read the handout "Basing Software on Reusable Technology" from Lethbridge *Object-Oriented Software Engineering*, 2nd Ed.
Files	Download OCSF 2.31 software. Jar file includes source, Javadoc, and binary code. Add this JAR file to your project as "external archive".

Overview

The purpose of this lab is to practice using a framework. We use the OCSF framework to provide network services and management of connections.

Without the framework, we'd need several labs just to implement simple network communication.

You should observe that a framework reduces development time and usually results in higher quality software. Most frameworks are extensively tested both by developers and users, so they typically has fewer bugs and a better architecture than software written from scratch.

In the next lab, you will design a Chat program using OCSF, and implement the first iteration of Chat.

Using OCSF

The OCSF framework is described in the handout by Lethbridge.

Your client uses methods of OCSF's AbstractClient or ObservableClient classes to connect to a server and send/receive objects.

The first step is to create a subclass of AbstractClient or ObservableClient.

Callback Methods (Don't Call Us, We'll Call You)

The framework has several *callback methods* that you can *override* in your client (your subclass of AbstractClient or ObvervableClient).

`handleMessageFromServer`	(required) this method is invoked whenever the client receives a message from the server
`connectionClosed( )`	(optional) this method is invoked if the connection to server is closed

For other callback methods, see the handout and OCSF Javadoc.

Control and Utility Methods

These are methods that you *invoke* but don't override. They are provided by AbstractClient.

`sendToServer( Object )`	send a message to server. May throw exception.
`openConnection( )`	attempt to connect to server. May throw exception.
`closeConnection( )`	close the connection.
`isConnected()`	test returns true if currently connected to a server

Problem 1: Connection using Strings

Create a client as a subclass of AbstractClient or ObservableClient. ObservableClient provides the same methods as AbstractClient and also extends Java's Observable, so its useful for writing a GUI interface.

Your client should do this:

a) Connect to server and display a "connected" message on client UI.

b) Send messages from client to server.

c) Display messages received from the server.

d) Close the connection when you signal that you want to quit.

Example:

```
Connected to server 158.108.32.99    <-- message from your program
> Hello. Please Login                <-- message received from server
Message:  Login Fataijon            <-- your name (bold)
> Hello Fataijon
Message: Get Task 1
> What is 2+2?
Message: 5
> Sorry, wrong answer.
Message: 4
> Correct!
Message: quit
Disconnected.
```

When you complete this task, your name will be added to the scoreboard.

Problem 2: Write a Chat Server for 1-to-1 Chat

Write your own server using OCSF's AbstractServer class.

You should create a server that requires clients to send a login message, and then send messages to each other. Instead of broadcasting messages to everyone, only send the message to designated receiver. Your application must recognize 4 kinds of messages, which are all Strings.

Login Message	One-to-One Message	Logout Message	Server message

`Login` *`username`*	`To:` *`receivername`*
	`message text`
		`Logout`	server can send any text message (string)

Since one-to-one messages require 2 lines, you will need to write a client, too.

2.1 When a new client connects, you should wait for the client to send "`Login` *`clientname`*". When he sends this, add *clientname* to the ClientConnection object. For example:

```
client.setInfo("username", clientname );
```

2.2 When a user logs in, servers send a message to all clients telling them "*`Clientname`* `connected`"

2.3 When a logged-in client sends a one-to-one message like this:

```
To:  Anchan
```

```
Hi, Anchan. How are you?
```

your server should find a client connection with login name "*Anchan*" and send the message. Be sure to tell Anchan who the message is from! If Anchan is not logged in, tell the sender that recipient is not logged in. If Anchan is logged in more than once, forward the message to *all* clients named "Anchan".

2.3 If a client sends the String message "`Logout`" then close the client connection and tell all other clients "*Anchan logged off*".

2.4 If the client sends any other message, the server responds that message is not recognized.

How to Record Client Name

OCSF creates a ClientConnection object for each connected client, and passes this object as a parameter in handleMessageFromClient.

You can save the client name using the setInfo / getInfo methods. For example:

```
handleMessageFromClient(Object msg, ClientConnection connection)
{
  //todo determine the client's name from message (on login only)
  String loginname = ?
  // setInfo uses a Map. You can use can String as a key
  connection.setInfo( "username", loginname );
  // to get the client's name from ClientConnection use:
  String username = (String) connection.getInfo( "username" );
```

How to Find a Client By Name

OCSF AbstractServer doesn't provide an easy way to search the current ClientConnection objects, so you can find a client by username.

One solution is to maintain your own list of **`ClientConnection`** in your own server class. Override the *callback* methods:

`clientConnected( ClientConnection conn )` - a new client is connected

`clientDisconnected( ClientConnection conn )` - a client has disconnected

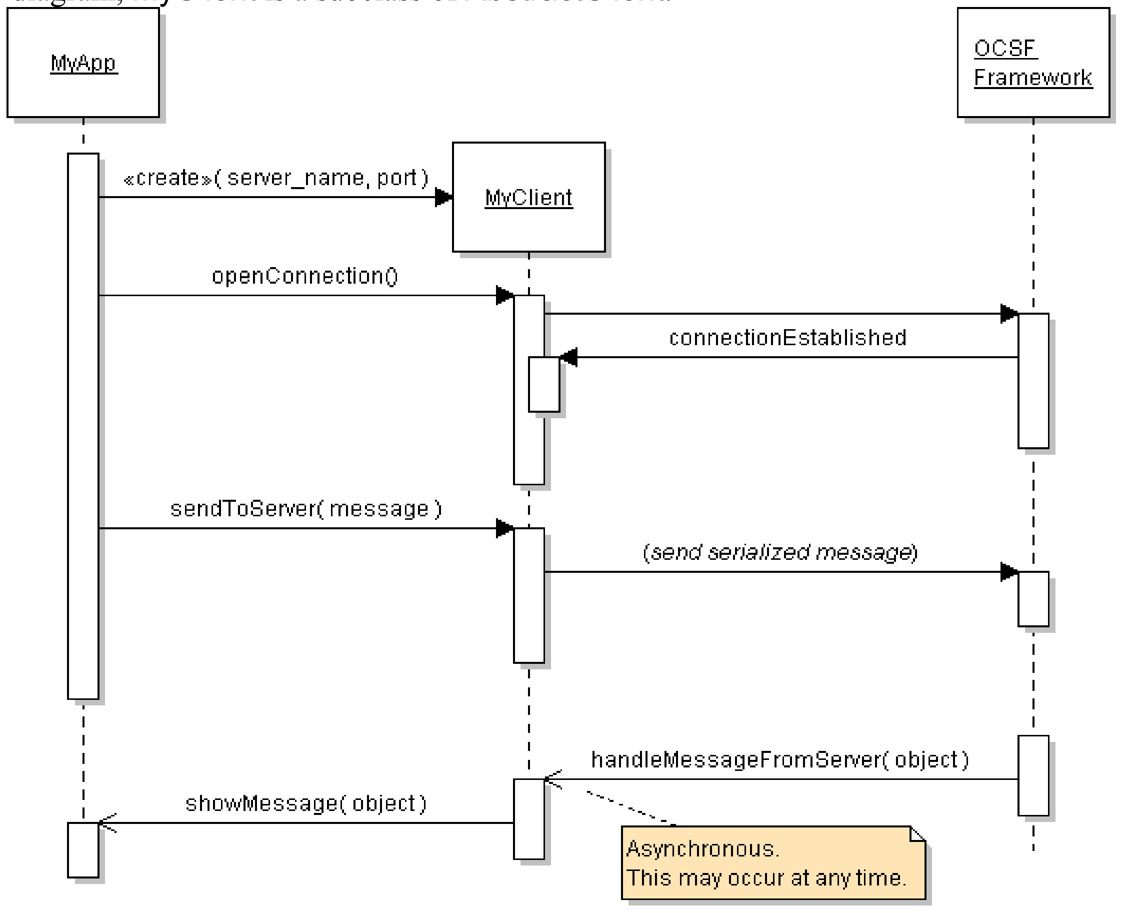
use these hooks to add client connection to your List and remove disconnected clients from your List. You could also use a Map, but must design the case where same user is logged in more than one time.

Better Protocol

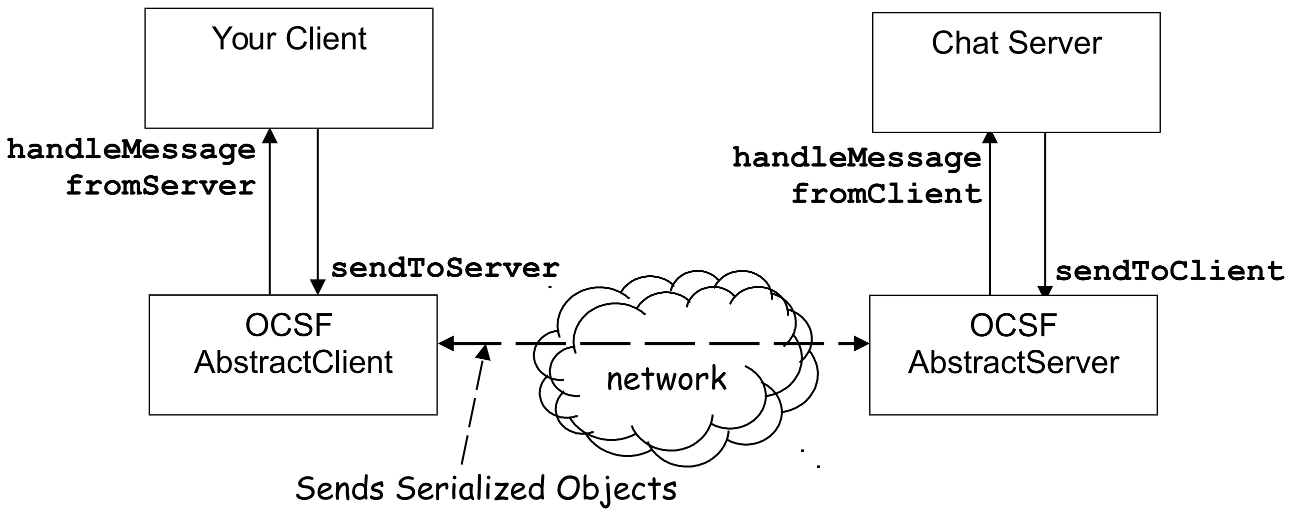
The above examples use Strings for message. OCSF lets us send any Serializable objects, we can create a better protocol by sending objects. If you want to write a Chat server, a better design would be to define a hierarchy of Message objects and send serialized Messages instead of Strings.

Typical Usage

In this diagram, MyClient is a subclass of AbstractClient.



Conceptual View of OCSF Operation



References

Lethbridge and Lagariere, *Object-Oriented Software Engineering,* 2E. Textbook describes use of OCSF and a chat project.

A standard, high-performance framework for chat and other applications is XMPP.

XMPP is a standard protocol for real-time messaging; XMPP was originally called *Jabber*. Google Talk uses XMPP. You can use XMPP to write your own Chat client or other Internet application. There are many several free XMPP servers (such as *Jabberd* and *OpenFire*), clients, and libraries. XMPP can be used for more than just chat.

SMACK is an open-source XMPP library for Java. It is used by several chat applications. http://www.igniterealtime.org/projects/smack/

- How to use SMACK to write a Java client: http://www.javacodegeeks.com/2010/09/xmpp-im-with-smack-for-java.html
- Other two articles in the same series describe infrastructure for using XMPP.

XEP-0045 Multi-User Chat. Protocol for a multi-user chat using XMPP. http://xmpp.org/extensions/xep-0045.html#bizrules-message