**Concept Updater using Client Server Architecture**

Development Environments:

Operating System: Mac OSX 10.9.4

IDE: Eclipse Indigo

Java Version: 1.6

Information about running the program: README.txt

Workflow:

1. There exists a client, which sends a message to the server with its current version (hard coded to 1 at the beginning).
2. The server responds to the client’s request by checking if the there exists a newer version on the server. If so, it sends the new version and a download URL. If not it sends back “0” signifying there is no updates available.
3. The client gets the message from the server and checks if it is not 0
   1. If the version sent from the server is zero, it waits for 3 seconds and queries the server again to check if there exists an update. This process goes on until the user kills the client.
   2. If the version sent from the server is >0 then (which means there was an update) the version of the client is updated to whatever version server provides.
   3. The client also parses through the download URL to get the address and the port of the “download server” which is yet another server that hosts the download file.
   4. The client opens a connection with this Download server. The server then starts transferring the download and the client prepares to receive the update.
   5. The client backs up the file it has as a “.bak” file and begins downloading the new file, and over writes the contents of its file.
   6. Once the process is complete, the Client again begins checking for newer version from the server by send it its updated version. This goes back to 3.a.
4. The process of checking for an update after every 3 seconds and downloading and applying the update, continues until we terminate the client and the servers manually.

Assumptions:

1. The Client and the Servers continue to run until manually stopped by the user. (This was done assuming the client and the servers never stop, the client continues to check for updates after a specified interval)
2. IndexServer.xml is the *update* which is hosted on the downloadServer. At the beginning of the execution, the indexServer.xml would have different contents than the indexClient.xml
3. Once the program completes the, the client would make its indexClient.xml as a backup “indexClient.xml.bak” and download and overrite the contents of the indexClient.xml to match what the download server sends it (which is the contents of indexServer.xml)
4. So at the end of the program the contents of indexServer.xml should match the contents in indexClient.xml

Proof of Correctness:

Right now I have a manual testing using the log files, which I have described below. (We can have a secondary test by verifying the contents of indexServer.xml and indexClient.xml were different before starting the program, and became the same after ending the program)

We could have used unit tests to test the individual files for increasing the coverage, and mocked data that we got from external sources.

However I feel it is better to have a functional test of two to actually start the servers, and build a test client and test the message being sent and received from the server.

Another option I could have used was an interceptor to intercept the messages being passed, but did not get enough time to do it.

Client LOG “Client\_Log.txt”:

*//client connected to UpdateServer*

*INFO: Client connected to server.*

*//client sends its version to the update Server*

*INFO: Sent Version no: 1 to Server*

*//client gets this data back from the UpdateServer*

*INFO: received this data from server: 2*

*//decides to perform the update*

*INFO: server version is > client version so need to update*

*//got the URL from the UpdateServer*

*INFO: URL read is http://localhost:5600/indexServer.xml*

*//backing up Files on client side*

*INFO: Backing up Client information*

*//connected to DownloadServer*

*INFO: Connected to DownloadServer*

*//downloading the file*

*INFO: File downloaded (271 bytes read)*

*INFO: File downloaded successfully :)*

*//updated client version to current server version*

*INFO: client version updated to : 2*

*//sending updated version to the server to check for updates*

*INFO: Sent Version no: 2 to Server*

*//receiving that there are no updates available*

*INFO: received this data from server: 0*

*//continues sending every 3 seconds*

*INFO: Sent Version no: 2*

*INFO: received version number: 0*

*INFO: Sent Version no: 2*

*INFO: received version number: 0*

Server LOG from “console”:

*Update Server:*

server sterted

Client accepted Socket[addr=/127.0.0.1,port=52571,localport=5000]

Received message from Client: 1

Received message from Client: 2

Received message from Client: 2

Received message from Client: 2

*Download Server:*

download server sterted

Client accepted Socket[addr=/127.0.0.1,port=52568,localport=5600]

Sending indexServer.xml(271 bytes)

Done.

Enhancements

Create a maven project with 3 maven modules:

a. Module For UpdateServer. After building its jar it can be deployed say Jetty Cargo server (including maven cargo plugin in the pom.xml and using the configuration to deploy the UpdateServer jar)

b. Similarly a Module for DownloadServer: have the logic in a file and deploy its jar on another jetty server.

c. start both servers:

d. Module for Client: used to run against the two servers and real time sending data to the servers and getting back information.