Part 1.

Phase 1 Distributed Objects. Week 1+2

What we want to do in the assignments for this module is to get experience with distributed systems programming. Our programs will eventually call and use programs on other computers, using sockets or RMI.

I would like to start with a file sharing system that works on one computer using a number of different GUI clients, eventually these clients will be running on different machines.

We will program to interfaces because this will make it a lot easier for us to extend the application to become distributed. But for the moment both clients and server can run on one machine, several instances of the same application.

We want to maintain a shared folder where each instance of the application can get files through a Monitor that provides access.

Each instance of the application also has its own local folder where it can store its own files.

The objective of this application is that a user of an instance can download the files he/she wants from the shared area which is controlled by the monitor.

The monitor should be instantiated in the client.

The monitor should use a separate thread to serve requests to check for changes to the shared folder to ensure that the application is not hung by any process. You should ensure that a thread is spawned where necessary to ensure that the application can continue processing even if a service it requires is busy.

Write an interface with the methods like the following ….. add or remove as you wish

String [] getNames()// returns the names of all the files in folder1

bool openFile(string name)// opens a file called name

Byte getB()//Gets a byte from the currently open file or any mechanism to read file for download

bool closeFile(string name)// closes the open file

bool checkForChange ()// sets a bool if a file has been added

Note:-

Write a class (called Monitor) that implements this interface. The class is to monitor one subfolder (Shared Folder) for additions. When asked by a client class the Monitor returns true if the contents of the shared folder has changed. The class Monitor can use a fixed constant directory folder (Shared Folder) to hold the monitored files. This can be set up manually on a machine and files can be dropped in this folder manually.

Write a class called myMediaPlayer (the client) which uses the class Monitor (containment not inheritance). It has a javaFX GUI, use SceneBuilder if you like.

The gui displays the file names of the files found in the Shared folder, followed by a play button and lastly a download button. The play will be enabled if the file is available in the client’s local folder, the download button is enabled if the file is not yet in the client’s local folder.

Note : Each client gui will access the shared folder only via the instance of the Monitor which is serving access to the shared folder, there should be no direct access to the shared folder.

Also the Monitor does not save directly to the client, transfer has to happen by the client calling methods on the monitor.

See music players, Swing and Fx filechoosers on blackboard.

Phase 2. Week 3

Apply design patterns to lab1. E.g. the observer , the singleton etc …..(anyone applicable)

Phase 3. Week 4

Add the functionality so that you can upload file to a shared folder which will be shared among several running Gui’s ( eventually this will also be distributed).

Phase 4. Add threading to your application anywhere applicable. Week 5 +6.

**Deliverable at week 6 with a demo required.**

# Part 2.

Do the same as in part a develop a distributed application with sockets and then separately with RMI. If you have programmed to interfaces then changing from sockets to RMI should be easier (a lot).

You should demonstrate the semantics of distributed systems interaction like showing parameter passing (all kinds) and returning values. Demonstrate Polling and call-back. Demonstrate non-blocking calls and explain in documentation why a blocking call was not used.

Add functionality to upload files

**Deliverable week 12 with a demo required.**