A P2P FILE SHARING SYSTEM

DESIGN DOCUMENT

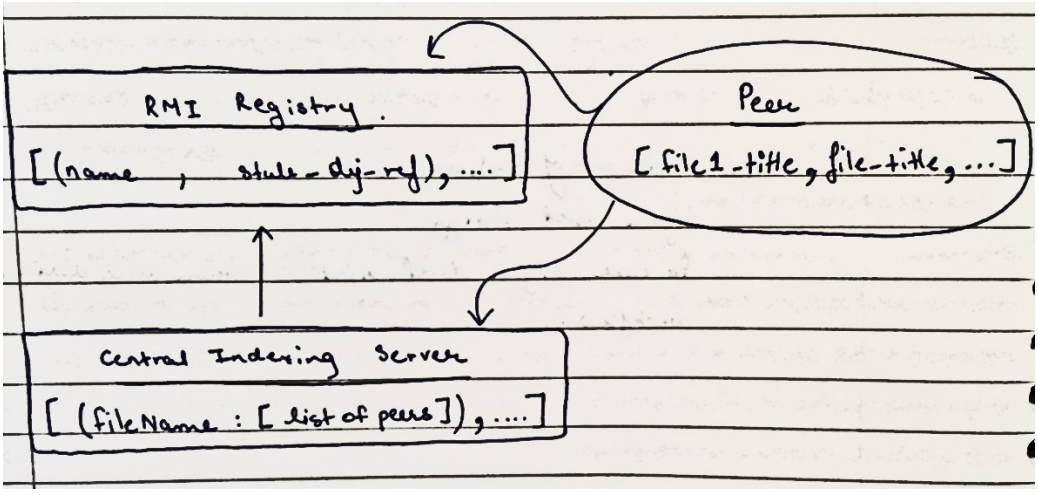
HIGHER LEVEL DESIGN

P2P file sharing system consists of 3 major entities which communicate over a network to make possible this conglomeration cum collaboration. These fundamental components are called : 1) The Central Indexing Server 2) Peer & 3) The RMI Registry.

The central indexing server acts as a mapper mapping the file names available on the network and for each file, the peer system ids which have the fie in their local directory.

The peer act as both a client and server. As a client, it registers its file directory with the central indexing server and as a server it acts as one of the communication end points during a file transfer between two peers.

The registry is a binding/lookup mechanism provided by the rmi package which holds the list of bindings between the different remote object's name aliases and their corresponding remote implementation stubs, which will be passed across the machines so as to achieve remote operations.



CONSTITUTING COMPONENTS' DESIGN

THE CENTRAL INDEXING SERVER : -

The CIS acts as a central directory for peers to query files available on the network, and as an interfacing unit helping establish communication between the client requesting file and a file holding peer.

It implements a remote interface declaring the following two functions : -

```
/**
 * Registers a client and its file directory
 *
 * @param clientName  String ID of a peer registering its file directory with the server.
 * @param fileNames  A list of files the peer wants to register with the CIS.
 *
 * @return true, if registration successful, false otherwise.
 */
public boolean register(String clientName, ArrayList<String> fileNames) throws RemoteException;

/**
 * Looks up the list of peer name ids with the requested file.
 *
 * @param fileName -The file requested by a client/peer.
 *
 * @return A list of string ids of the peers holding the requested file.
 */
public ArrayList<String> lookup(String fileName) throws  RemoteException;
```

THE PEER : -

As mentioned as a client, the peer will use the CIS' *register function* to register all its files with the CIS mapper.

As a server it provides the following interface implementation so as to facilitate file transfer : -

```
/**
 * Retrieves a list of text lines of the file requested.
 *
 * @param fileName -The file requested by a client/peer.
 *
 * @return A list of text lines of the file requested.
 */
public ArrayList<String> retrieve(String fileName) throws RemoteException;
```

```
/**
 * Retrieves the peer's file directory.
 *
 * @return A list of file names in the peer's directory
 */
public ArrayList<String> getDirectory() throws RemoteException ;


/**
 * Retrieves the number of files in the peer's directory
 *
 * @return The number of files in the peer's directory
 */
public int numFiles() throws RemoteException;
```

THE RMI REGISTRY : -

The registry provides a mechanism to gain access to remote objects across the network. It provides a simple *'rebind'* function that binds a remote object with an identifier. Later the same remote object can be searched in the registry by calling invoking its *'lookup'* functionality with the requested peer's identifier as argument to get the corresponding remote object.

* - *

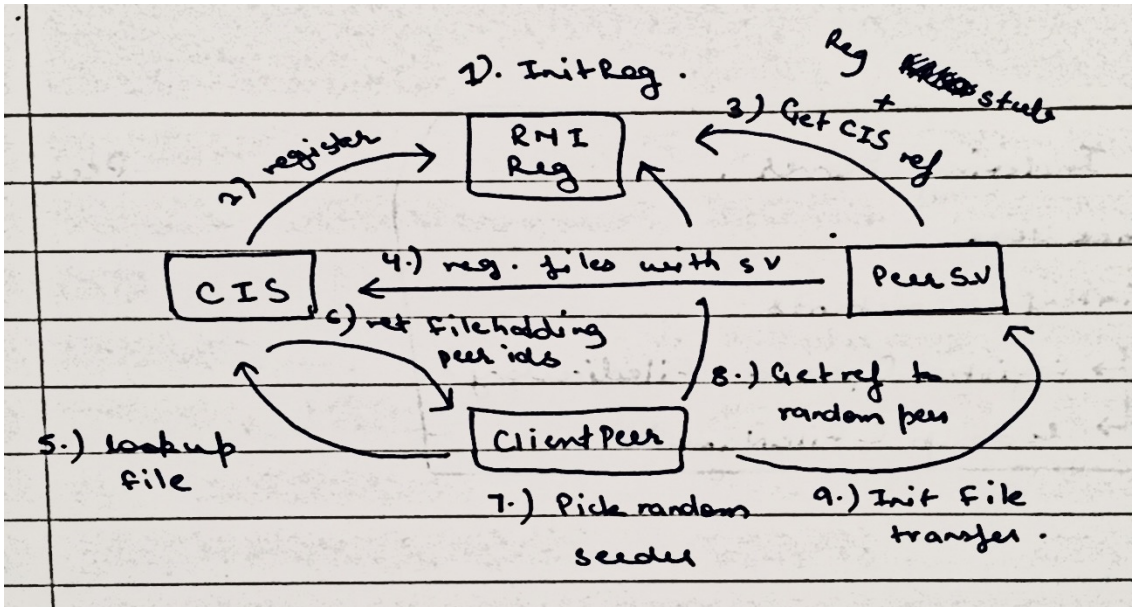
An important thing to emphasize here is that the CIS only holds the 'identifiers' of the remote objects registered with it. Whenever an actual remote object ref. Is needed the rmi registry is queried with the required peer's identifier to get in return the corresponding remote object.

* - *

THE SYSTEM INTERACTIONS AND PROGRAM FLOW

1) The registry is fired up at the default port number 1099.
2) The server application is run.
    a. Here in the server creates its remote object and binds it to the identifier 'CentralIndexingServer' inside the rmi registry.
    b. It also exports its remote object which the waits for any incoming client requests at a random port initialized during run time.

3) All the peers wishing to be part of this P2P network are uplinked.
    a. Herein the peer initializes its directory of files.
    b. It then contacts the rmi registry to gain access to the CIS reference by '*looking up'* the CIS with its identifier 'CentralIndexingServer'.
    c. It then registers all its files, if any, with CIS by calling the *register* function on the server object.
4) A client application requesting a file from the network is fired up on one of the peers.
    a. Herein, the application gains access to CIS through the registry and calls its lookup function to get the list of peer identifiers holding that file
    b. Picking any random seeder from the list of peers received, the client will then use the identifier to gain access to its remote object through the rmi registry.
    c. The client then calls the retrieve function on the peer object to download the file.
5) After the file has been downloaded, the client registers itself with the central indexing server as one of the holders of the newly downloaded file.



A NOTE ON CONCURRENCY & AUTOUPDATE

As already mentioned, the protocol that stands in between the remote machines is the Remote Method Invocation (RMI).

The protocol has been implemented in the project by means of the java RMI package. Java RMI provides in built support for multithreading, that is, each remote method invocation call by multiple clients spawns a separate thread of execution.

In such a case what is important is to ensure thread safety in the code. Care has been taken to ensure that several thread executions of the different remote methods can be interleaved for safe concurrent operation.

The auto update mechanism has been put into place, however, this mechanism only kick starts either after a file download, to reflect the downloading peer as another file holder of the downloaded file, or in case a file was deleted from a peer's directory and the peer was rebooted again.

FUTURE IMPROVEMENTS

1) Adding more functionality to the central indexing server.
    a. Capable of returning the list of peers registered with server
    b. Functionality to return all the files names registered with the server.
    c. Capabilities to download all the files not present with the client.
2) Adding more functionality to the peer class.
    a. Capabilities to download multiple files from the network through simple mods.
3) Using hash codes to enforce security and improve operational speeds (integer lookups/manipulations are always faster than string manipulations)
4) More automated distribution of stub classes across the network.