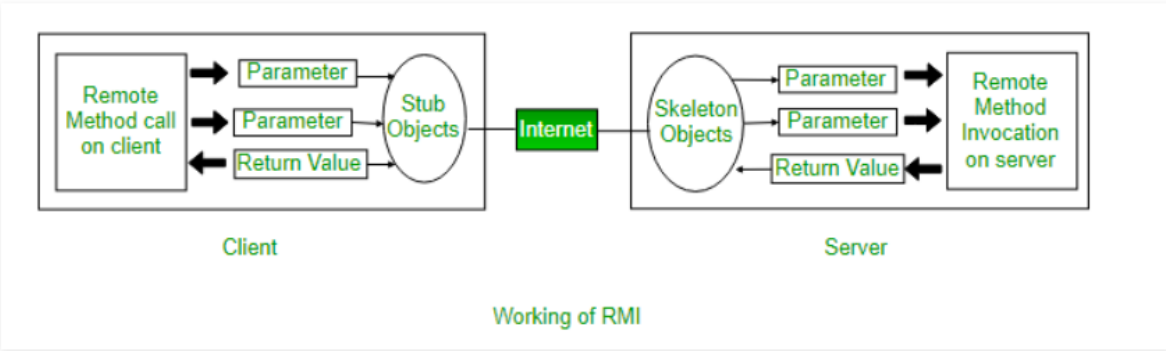
Distributed File Search Engine

The aspect of Distributed File Search Engine is to make available the files and file structure through any kind of user queries across the network to any users or developers without much changes in the structure of code.

This is achieved by the RMI which is the object-oriented method invocation on server.



Steps to implement Interface

Figure 1

RMI Overview

The steps involved in RMI are,

1. Define Remote interface (*FSQuerI*)
2. Implement Remote Interface (*FileSystemServer*)
3. Created Stubs and getting stub objects (*QueryDirDepth, QueryDirStructure, QueryFileSearch*).
4. Created Server Application (*FileSystemServer)*
5. Created Client Application (*FileSystemQueryClient*)

UML Class Diagram:

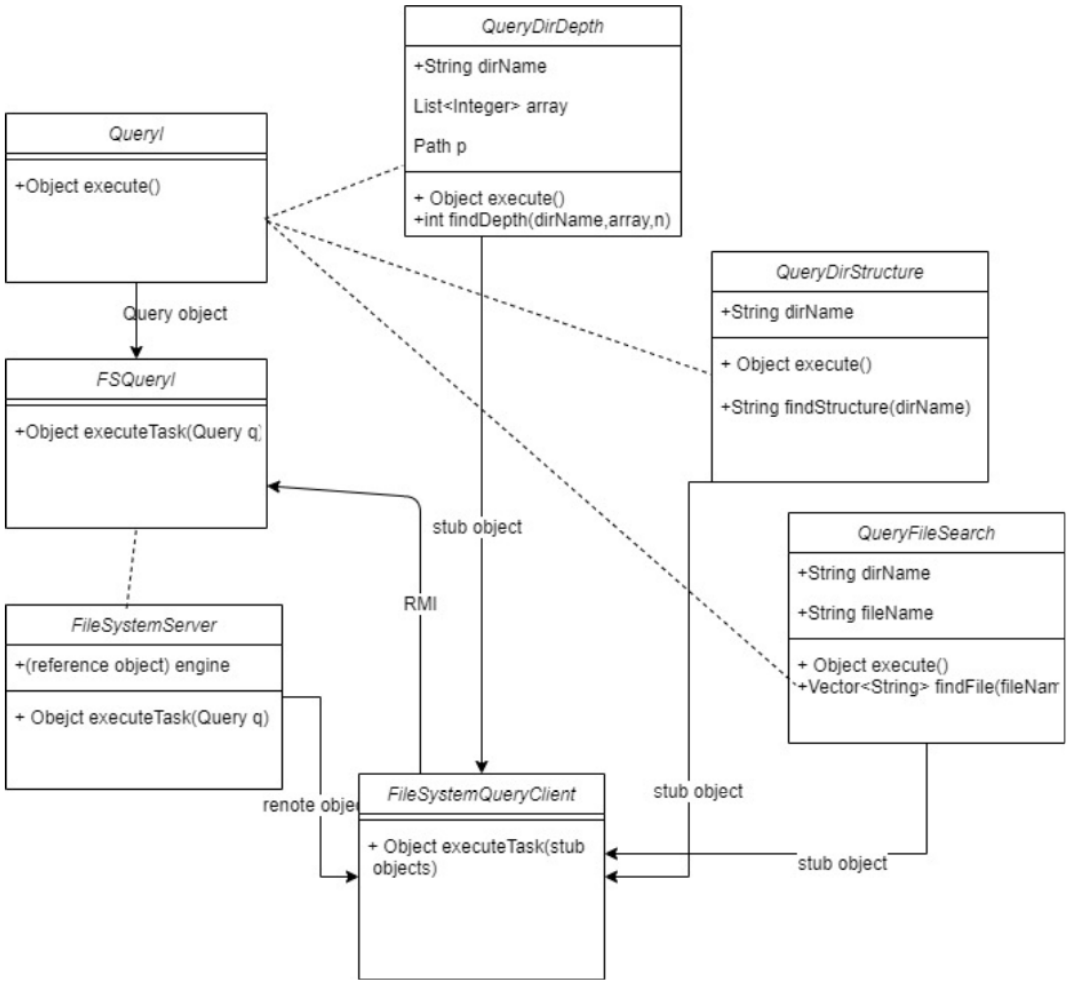


Figure 2 UML Class Diagram

Class Diagram Brief:

1. **FSQueryI :** It is the remote interface which has the method call which has to be implemented by the client according to the need. So, it is not necessary to modify the interface since it defines only the definition and not the actual task or query.

2. **FileSystemServer:** It creates the remote object and provide it to the interface. And, it performs the task without knowing the actual implementation.
   It acts as the skeleton for the server.

3. **FileSystemQueryClient:** It takes the remote object from the server interface and then invokes the actual method on the FileSystemServer with the remote object. It gets all the stub objects from the query implementation classes and passes these stub objects to the skeleton which is the FileSystemServer for execution.

4. **QueryI:** Query Interface has the implementation declaration method which has to be implemented by any client according to their requirement. So, here also there is a flexibility that the interface code need not be changed for the arbitrary queries anytime.

5. **QueryDirDepth:** It is one of the client programs requesting for the depth of the directory given the directory name.

6. **QueryDirStructure:** It is one of the client programs displaying the directory structure for the corresponding directory name input.

7. **QueryFileName:** It is used to get the absolute path of the corresponding filename from wherever present in the directory.

Code – Class Diagram Comparative study:

1. **Server Side:**

- Server programs creates references for remote objects and makes available for client to access the methods using these remote objects. The reference object here is 'engine' which is made available to the client access.
- Also, the reference object is only created for the Interface and not the implementation.

```
try {
    String name = "rmi://" + args[0] + "/FileSystemSearch";
    FSQueryInterface comp = (FSQueryInterface) Naming.lookup(name);
```

```
public static void main(String[] args) {

if (System.getSecurityManager() == null) {
    System.setSecurityManager(new RMISecurityManager());
}
String name = "rmi://localhost/FileSystemSearch";

try {
    FSQueryInterface engine = new FileSystemServer();
    Naming.rebind(name, engine);
    System.out.println("FileSystemSearch Engine bound");

} catch (Exception e) {
    System.err.println("FileSystemSearch exception: " +
            e.getMessage());
    e.printStackTrace();
```

- Then Binding the object with the rmi naming registry.

- **Remote Interface:** We have the remote interface – FSQueryInterface (FSQueryI – as in UML Class diagram).

```
public interface FSQueryInterface extends java.rmi.Remote {

    /*
     * Q (1.2)
     *
     * Please define your remote interface here
     * [Hints: have a look at Compute.java (Lab 3) and the surgery exercise
     *
     *
     */

     public Object executeTask(Query q) throws java.rmi.RemoteException;

}
```

- **IMPLEMENTATION:**

  The corresponding remote implementation won't be available to server. It is present in the client and everytime the client calls the method and execute the method through the server with the remote object.

  It is simply the return statement in server not the actual implementation.

```
     *
     */
    public Object executeTask(Query q) throws RemoteException {
        return q.execute();
    }
```

2. Client Side

The client program has the actual implementation of the queries and passes the query object to the server for execution by calling the remote method exectueTask(Query q).

```
try {
    String name = "rmi://" + args[0] + "/FileSystemSearch";
    FSQueryInterface comp = (FSQueryInterface) Naming.lookup(name);
```

Here, 'comp' is the remote object to invoke the remote method on server taken from rmi.

```
String directorystructure =((String)(comp.executeTask(queryDirStructureTask)));
Integer dirdepth=((Integer)(comp.executeTask(queryDirDepth)));

Vector<String> file= ((Vector<String>) (comp.executeTask(queryFile)) )   ;
```

With 'comp' method, we are able to call the remote method executeTask()

Reference:

1. GeeksforGeeks. 2020. *Remote Method Invocation In Java - Geeksforgeeks*. [online] Available at: <https://www.geeksforgeeks.org/remote-method-invocation-in-java/> [Accessed 23 March 2020].