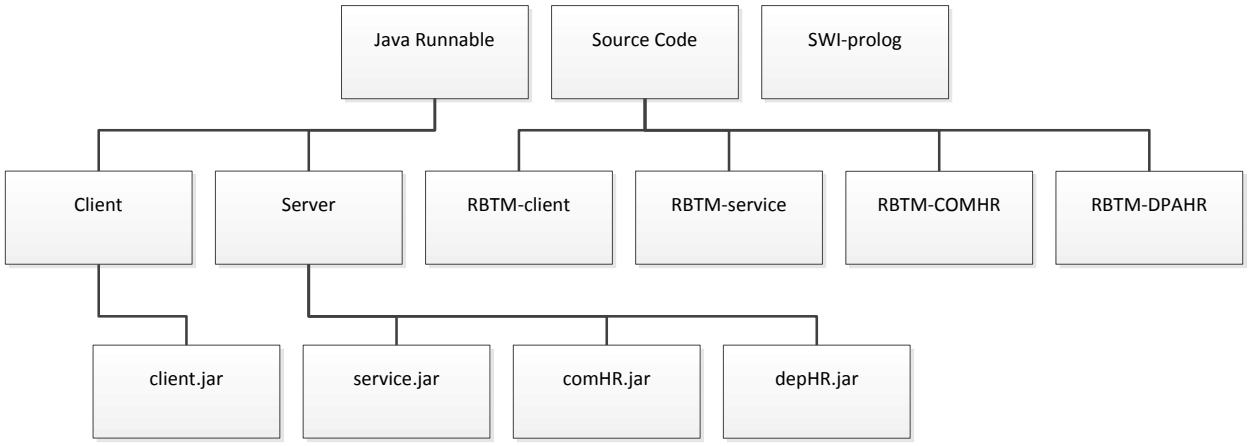
User Manual for Implementation of Final Project of CS557

Yunhe Tang, Jun Tang, Di Wang

Project File Structure

There are three files under the project root directory. Each file contains one or more sub files in it. Below is a graph that illustrate the file structure of the project.



1. To run the program, go to Java Runnable directory. There are in total 4 programs needed to be run: client.jar, service.jar, comHR.jar, and depHR.jar. It should be noticed that three server programs should be launched **before** client program starts. The detail information about how to run the project is given in the next section.
2. To access the source code of project, go to Source Code directory. Each sub project file is exported from Eclipse workspace. It should be able to be imported into your IDE directly. **Beware**, some environmental settings need to be done before running the program. Again, it will be described in the next section.
3. The project takes advantage of Prolog language to implement some core functions. Specifically, we use SWI-prolog to do that. To make Java program cooperates with the prolog program, we use JPL, a Java interface to Prolog. These tools are included in SWI-prolog file.

How to Run the Program

This section is a tutorial to describe how to run our program. This sections include 4 parts: key generation, prolog installation, authorization user command, and program execution.

1. Key generation
   Since our program needs two way authentication, meaning that both server and client need to confirm the identity of their peer before constructing the SSL connection, we need to assign a key pair (including a private key and a public key) to each client and server. To do this, we take advantage of Java keytool. It will generate a key store, which contains key pairs and certificates it trusts.
   Since we do not have a centralized CA, we manually add certificates of peers to one's key store. Once the certificate is added into a key store, whoever uses this key store will trust the peer on the certificate.
   There are three commands you can use to generate your own key, this is also where you determine the password of key store. Suppose that you want to generate a key for someone named A.
   1) To generate a new key store, which contains a public key and private key in side.

```
keytool -genkey -v -alias A -dname "CN=A,OU=CS557,O=WPI,L=WORCESTER,ST=MA,C=US" -keyalg RSA
-keypass 123456 -keystore A.key -storepass 123456
```

   2) To fetch a certificate from the key store you just created.

```
keytool -export -alias A-keystore A.key -file A.cer -storepass 123456
```

   3) Add this certificate to another key store.

```
keytool -import -trustcacerts -alias A -file A.cer -keystore B.key -storepass 123456
```

   By completing the steps above, you should be able to use this new key. In our project, we have already done the key generation part, so basically you shouldn't have any problem with the keys. By the way the key is placed along with the Jar file.

2. Prolog Installation
   We take advantage of SWI-prolog to run the prolog program in side our Java program. To this end, we need to setup the prolog environment in advance.

When building up the project, jpl.jar is needed to be added into the build path. Jpl.jar is a java interface to prolog, which is 100% implemented in Java. JPL uses the Java Native Interface (JNI) to connect to prolog engine through the prolog Foreign Language Interface (FLI), therefore, we need to specify the path where jpl.jar could find necessary native library. **Beware**, there is one issue that may bring uncertainty to this setup. Since we build our project in Windows platform, we only have to add the path of related dll file to the build path and it's done. However, I am not sure how to set it up in Linux/Mac platform.
In windows platform, if your IDE indicate that jpl.jar cannot find dependent library. Then please add the swipl/bin path to your system environment variable PATH.

3. User Command
   Last thing to mention before start running the project. User command is the available commands that supported by the client program. They are:
   1) readDoc,X (X < 100)
   2) readCode,X (X > 100)
   3) writeCode,X,Y (X > 100)
   4) addProj,X,Y (X > 100)
   5) exit

   In the above command set, X and Y represents the index and value respectively. Also, if you want to act on code, the index (or the value of X) should be larger than 99. On the other hand, the Doc index should be smaller than 100. If the format or the parameter of command is invalid, client will receive an error message from server. For example, if you want to write code file 120, and assign its content 34, then you should type command "writeCode,120,34".

   Exit command indicates that the client want to log off and close the program.

4. Program Execution.
   To run the program, the user need to launch three servers and one client individually. Since this project is command-line based, we recommend you to launch these programs in four terminals.

When launching the server, you are required to input the port number and a password. The password is used to decrypt the local data store and key store. In our program, all the password is set to: 123456 (I know it’s stupid, just for convenience here)
After you have launched three servers, you can now launch the client program. You should input a username and password. If your username and password is inconsistent with the key file you provided, you would fail to login. For example, if you are baker, but you login with tom.key file, then it is invalid. To change the key file you want to use (so as to login another user), simply, you can change the path in application.properie file, which configure the launch setup environment.