COMP90015: Distributed System – Assignment 1

Multi-threaded dictionary

Name:Yizhou Zhu

Surname:ZHU

Username:yizzhu

Student id: 1034676

Executive Summary:

This report aims to give the reader a brief introduction to the system developed by Yizhou Zhu (1034676) to meet the requirements of a multi-threaded dictionary. The report included five sections:

1. Introduction of the problems
2. Architectures of the System
3. Main components of the system and their functions (with class diagram)
4. Interaction processes
5. Critical analysis the advantage and disadvantage
6. Conclusion

1.Introduction:

The assignment 1 of COMP90015 required student to develop a multi-threaded Dictionary System. The 'multi-threaded' means multiple clients can connect to a (single) multi-threaded server and perform operations concurrently [1]. A single Server will process all the requests, such as add a new word to the dictionary, remove a word from the local dictionary and search for the meanings, from several clients. Moreover, the two sides should both be capable of handling any exceptions such as wrong command exchanged, losing Internet connection, I/O problem and any other potential exceptions.

2.Architecture of the System:

The system is a raw model to test, hence it is a traditional two-tier structure. The client will send request directly to the Server side via no interim agent and middleware. The Server will process the request and extract the query information within. Accordingly, the server will follow the command or reject it according to the Server user.

As stated in the requirement, the server is using multi-threaded technique to allow multiple incoming requests to process. The thread architecture implemented here is a hybrid model. It is half-thread-per-request and half-per-connection. To call it as a hybrid model is due to the features of the system. The client side is designated to connect via a TCP socket. However, the client is designated to send one request via the TCP socket per connection. This architecture will add on the socket listening pressure at some point. However, the trade-off is to keep the load of the workings on the Server side to the minimum extent. There will be no redundant client performing no request occupying the thread. The server can perform more efficiently on that way especially on the occasion that high volume requests are sourcing.

Main components of the Systems and the functions:

The system contains two different packages. The client_terminal package and the Server_terminal package. As the name said, they are the package of components to build a client and a Server.

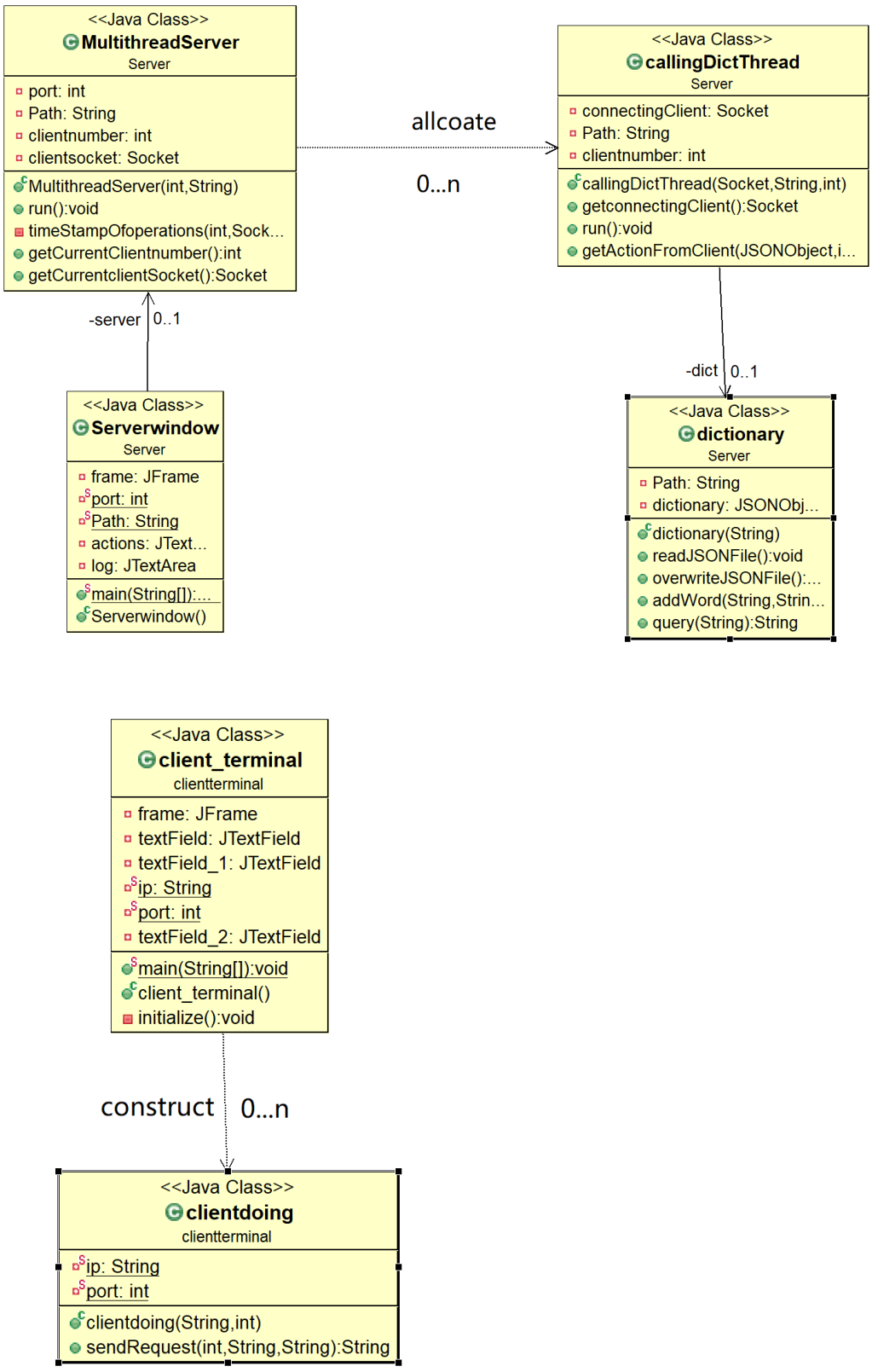
The first package is packed in DictionaryClient.jar. It contains two classes named clientdoing and client_terminal.java. The later one is the Graphical User Interface which will be executed as the main frame as the client terminal. It will construct new clientdoing constructor to build new client to send request to the Server. It can also handle minor exceptions if problem occurred. However, the interface can reflect most of the exceptions handled by the Server.

The second package is packed in DictionaryServer.jar. It contains four classes named MultithreadServer, CallingDictThread, dictionary and ServerWindow. The last one is the main class to execute. The ServerWindow will call the MultithreadServer to run and receiving the request. As long as the socket is connected, the MultithreadServer will construct a new thread based on the CallingDictThread class to interact with the local dictionary file.

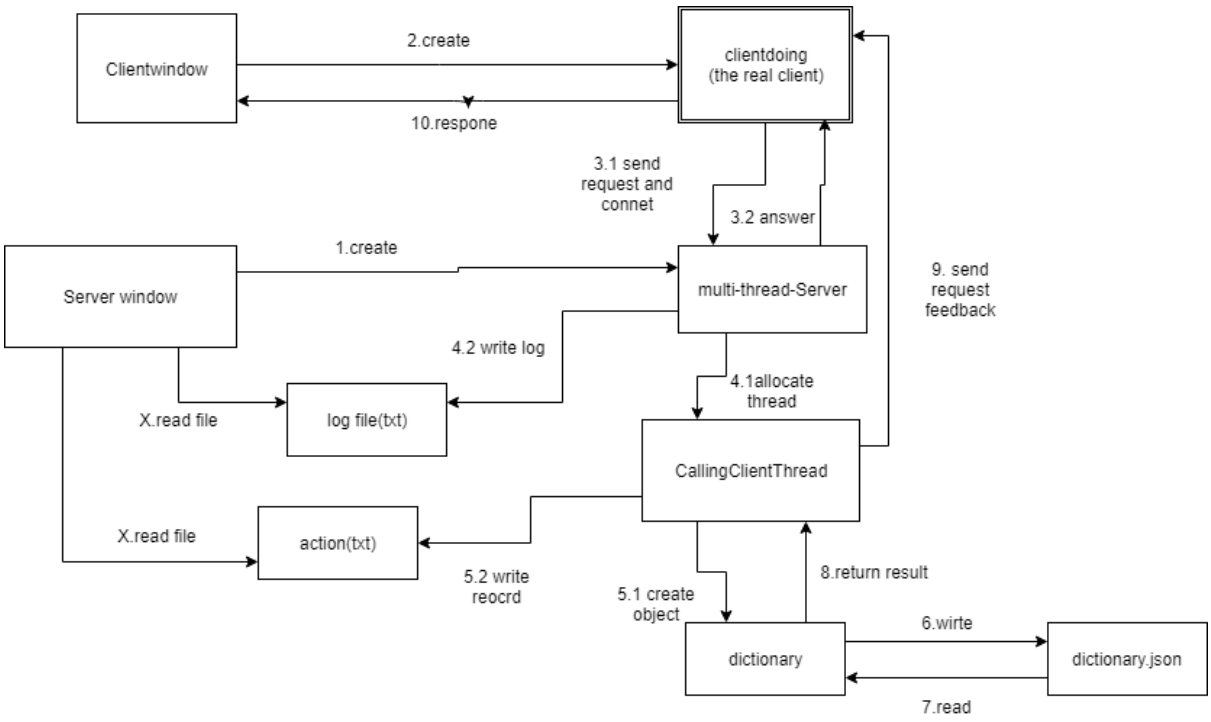
Dictionary class is the main interface to interact with the local Dictionary.json file. The current

thread in Server will always construct a dictionary to complete it task. Moreover, most of the exceptions are handled on the Server side.

Class Diagram:



Interaction process:



As demonstrated in the graph above, the general process to follow beginning from the Server side.

The Server will open the DictionaryServer.jar file and set the parameter as he or she wants. A new Server will be online when the user press ‘connect’, to keep listening for the incoming socket from any clients. Meanwhile, the User will run the DictionaryClient.jar file and set the Server hostname and port number. The user interface will pop out.

If the user wants to add any word with meanings to the dictionary. He or she should input the word into the first text area and input the corresponding meaning into the second text area. Afterwards, he or she should press the ‘add’ button.

If everything is going well, the third test area will show “word added”. If the word is duplicated in the dictionary, it will show ’already existed’. For searching and removing functions, it works to type into the first area with the word you need. The user can ignore the second text area. The response in the third text area read ‘completed’ or ‘no word in the dictionary’ states the result literally.

On the other side, the Server will get two text file stating the actions performed by the clients in the format of “No.x client tried to (add | remove | search) (word) (meaning)”. The corresponding index can be used in another text file named systemlogger.txt to query the details such as hostname, port number used and timestamp.

The information exchanged between the two clients is JSONObject. It can be a good method to encrypt the communication. A more sophisticated encryption protocol can be developed on it.

Critical analysis the advantage and disadvantages:

From a critical perspective, it will be rated 8 out of 10 work.

Beyond the elementary requirement , the bonus point in the model is the Exception handling and the Server log. The log is a raw type and a better version can be developed out of it. Some general review jobs can be conducted such as traceback and flow numbering. A better frame on xml or other formats will get implemented.

The exception handling is another bonus point. Combining the information on the command line terminal and response on the user interface. Most of the exceptions can be handled properly.

Such as the connection state, If the command line terminal reads “connection established” but the response on the interface reads “no connection right now”. It is likely that the Server has some problems to parse the request and send back the acknowledgement or the acknowledgement is corrupted. However, no “connection established” will lead to another explanation such as the Server is down. Let alone the basic invalid command sent type of problem, the information popped can be enough to figure out a solution.

However, the disadvantages are undeniable. The thread allocation is not optimal. The thread is just allocated and ignored in the following process. A valid and dynamic process should be developed to supervise the thread states. Some thread management methodology importable can highly boost the efficiency of the work. The way to handle the problematic thread is on demand.

The TCP socket utilization is far away from best. The remote dictionary would be best to store information. If the further development asked for a credential key, the project will be reconstructed a lot for the waster of TCP socket. It is appropriate to conduct several operations with only one credential key acknowledged. All the features can be better performed with UDP socket to some extent.

Conclusion:

All in all, the system prototype can be self-evaluated as a feasible product for further testing. The improvement in thread and socket utilization should be concerned.