COMP90015 Project 1 Report

Shenglan Yu 808600 shenglany1@student.unimelb.edu.au tutor: Minxian Xu

1 Introduction

The project requires to implement a multi-threaded dictionary server using client-server architecture which allows concurrent clients to search the meanings of a word, add a new word and remove an exiting word. All communication should occur via sockets in a reliable manner. Multi clients should be able to connect to server and make operations concurrently.Clients should be informed clearly whenever errors happen. Message exchange protocol chosen in the project is JSON.

2 System Description

2.1 Architecture

The system implements a two-tier single-server multi-client architectural model. Each connected client can use dictionary service concurrently through Java threads. Each client's requests are treated with the same priority. For the threads architecture, the system implements a Thread-per-connection architecture. Using Thread-per-connection architecture, no matter how many operations users take, only one tread was created for each client when connection successful. Compared to Thread-per-request architecture, the implemented architecture saved overhead cost for creating threads. Compared to Worker-Pool architecture, the implemented architecture was much more flexible to scale and suitable for long-term service.

2.2 Concurrency

Shared resources are synchronized at entry level. No concurrent operations on the same dictionary entries are allowed. Concurrent operations, however, are still allowed on different dictionary entries.

2.3 Service Operations

Once connected, clients are able to do three dictionary operations:

- Search: *Query the meaning of a word.*
  Parameter: the key to be queried
  Output: the operation result *Success, Failure* and failure reason *e.g key not exist* if failure or word meaning if success
- Add: *Add new entry to the dictionary with given key and meaning.*
  Parameter: the key to be added and its meaning
  Output: the operation result *Success, Failure* and failure reason *e.g key already exist; missing meaning* if failure
- Remove: *Remove one dictionary entry.*
  Parameter: the key to be removed
  Output: the operation result *Success, Failure* and failure reason *e.g key not exist* if failure

2.4 Interaction

All interactions happen through Java socket interface using TCP protocol in interest of reliability. One advantage of using Java socket is its programming language independence, allowing the dictionary service to cover board range of clients. The system uses JSON format to exchange message as JSON is easy to convert between string and JSON objects. For clients, each request would have one *task* type $(Search, Add, Remove)$, and corresponding fields value. For example:

$$\{task : "Search", key : "apple"\}$$

Dictionary server then parses received JSON object into string, makes corresponding operation and sends back result JSON object. The result object has one *Result* field indicating whether the request was dealt successfully and one *Description* field giving detailed operation result. For example:

$$\{result : "Success", description : "apple : fruit"\}$$

2.5 GUI

The system implements GUI for both client and serve using JavaFx tool. The style applied base 16 dark background colors in interest of modern taste.

1. Splash Screen
   The client interface will introduce a splash screen for 3 seconds before the login interface shows up.

Figure 1: Splash Screen

2. Login Interface
   The client login interface has *Server Address* and *Port Number* two textfields. *Server Address* has default value and *Port Number* prompted users to enter numeric value.

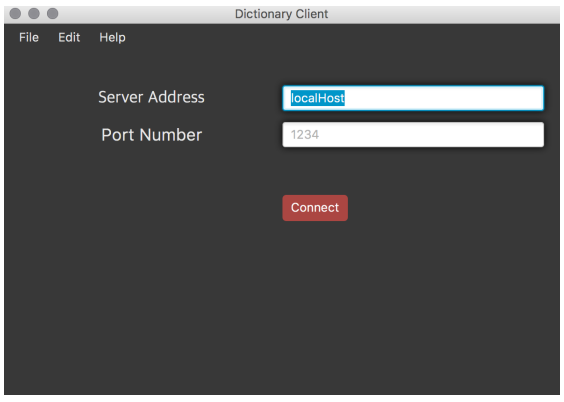

Figure 2: Client Login Interface

3. Client Operation Interface
   Once connected successfully, clients will have operation interface which has the dictionary functional services and disconnect button which returns back to login interface.

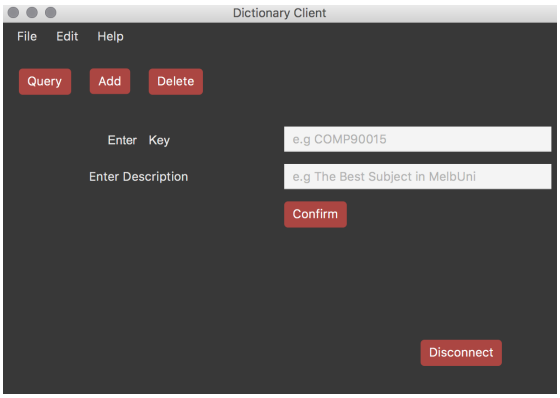

Figure 3: Client Operation Interface

4. Server Launch Interface
   The server launch interface has *Dictionary File Path* and *Port Number* two textfields. *Dictionary File Path* has default path and *Port Number* prompted users to enter numeric value.

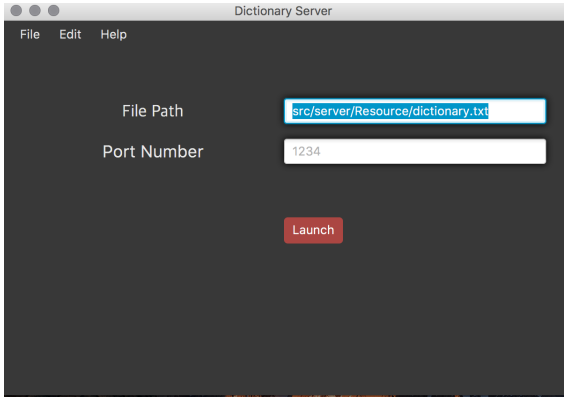

Figure 4: Server Launch Interface

5. Server Running Interface
   Once launched, the running interface shows up.which has a running message and a shut down service button.

2.6 Errors

The system applies a called *60 seconds silence out rule*, in which clients would shut connection automatically after making no requests for 60 seconds.

The system caught several types of exceptions and inform users with helpful messages:

1. FXMLException
   Invoking Issue: Errors occurred loading interfaces which were written in FXML document.

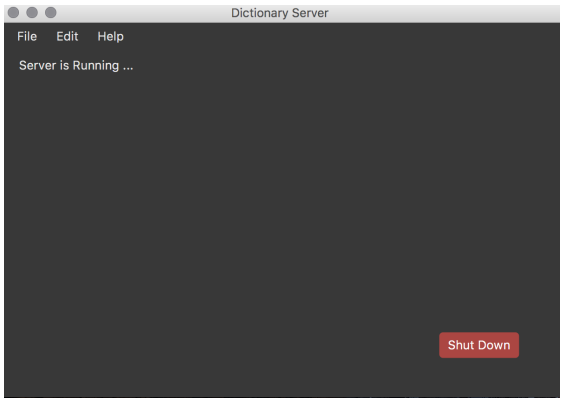

Figure 5: Server Running Interface

Interaction Message: FXML parsing error,please contact designer. *(Information of errors which occurred in primary stage would be printed in console)*

2. ServerInvalidException
   Invoking Issue: Client can not connect to server given the server address and port number
   Interaction Message: Can't connect ,check address and port.
3. BadCommunicationException
   Invoking Issue: Errors occurred when sending and receiving messages
   Interaction Message: Stream Error,maybe connection lost,connect again.
4. CloseException
   Invoking Issue: Errors occurred when trying to close sockets or input & output streams
   Interaction Message: Can't close socket, please try later.
5. InvalidParaException
   Invoking Issue: Errors occurred when users missed key or meaning to make operation
   Interaction Message: Key and Meaning can not be blank.
6. NumberFormatException
   Invoking Issue: Errors occurred when users enter non-numeric contents in port number textfield.
   Interaction Message: Please enter valid port number.
7. FileNotFoundException
   Invoking Issue: Errors occurred when server can not find dictionary file
   Interaction Message: Please give valid dictionary path.
8. FileFormatException
   Invoking Issue: Errors occurred when server can not parse dictionary file
   Interaction Message: Unreadable File Format.
9. InvalidPortException
   Invoking Issue: Errors occurred when server can not create socket on the given port
   Interaction Message: Can't build socket on given port.
10. ClientLostException
    Invoking Issue: Errors occurred when clients lost connection
    Interaction Message: StackTrace would be printed in console.
11. CustomPathWarning
    Invoking Issue: Warnings created when users try to use custom dictionary file path
    Interaction Message: Warning,data integrity unguranteed.

All interaction messages except those printed in console were displayed in *wrong label field* through GUI, for example:

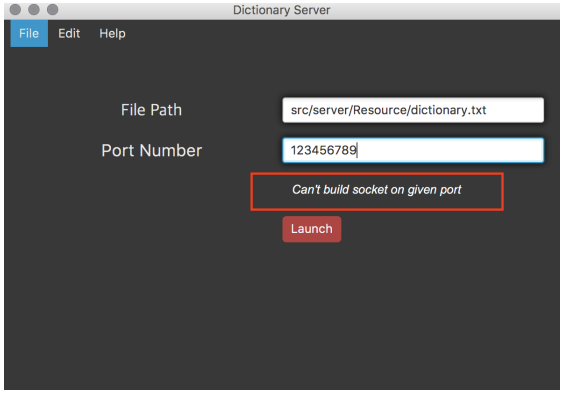

Figure 6: Example of Error Message

3 Class Design

The overall class design follows JavaFx framework with controllers handling interfaces. The distributed features are encapsulated in specific classes.

3.1 Client

Main class contains MainController and SplashController classes. When clients connect successfully, MainController invokes DictionaryClient and OperationController to take stage. Request parameters are captured

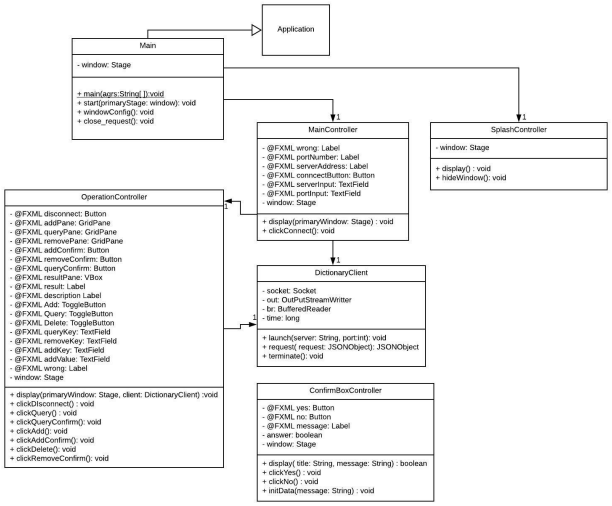

Figure 7: Client UML

through operation interface and sent by DictionaryClinet's request method. ConfirmBox will show up whenever users attempt to close the application. A brief interaction flowchart and UML are attached.

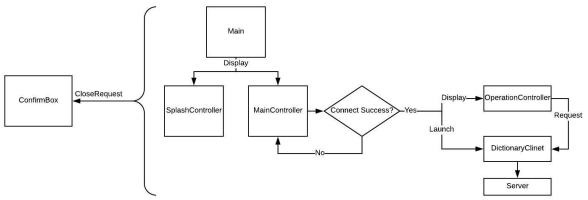

Figure 8: Client Interaction

3.2 Server

ServerWrapper and DictionaryServer implement Runnable interface. ServerWrapper takes care of server sockets and accept concurrent clients' connection while DictionaryServer takes care of each clients' requests. The dictionary service is encapsulated in BasicDictionaryService class. When server launches successfully, Running interface shows up and ServerWrapper starts running.

4 Critical Analysis

Following issues deserve active discussions:

- Concurrency Management: Multiple clients can access the sharing dictionary resource, which was a JSONObject. However, clients can not access the same dictionary entry at the same time. The design is implemented by synchronized

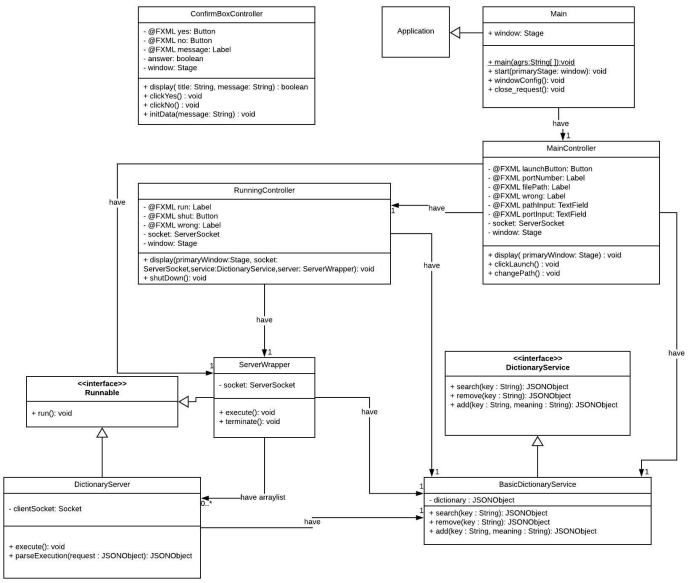

Figure 9: server UML

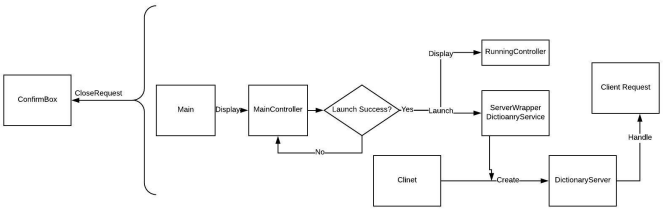

Figure 10: Server Interaction

entry utility level functions. The purpose is to prevent some data inconsistency, like *one client requested to search "apple" while another one requested on delete "apple"*. Different users can still manipulate different entries concurrently, like *one client requested to search "computer" while another one requested on delete "apple"*. The advantage is keeping both simultaneous operations and data integrity to some extent. However, the trade-off is that some data inconsistency may still happen, for example: *one client requested to add "bacon" with some meanings at the same another one requested on add "bacon" as well with another meaning.* Since the *bacon* entry was not in the dictionary before, the concurrent operations will be allowed, resulting some unexpected bad issues. The disadvantage also includes some user inconvenience. For example,*two users can not search "apple" at the same time.* The design is at halfway between

synchronizing the whole dictionary and not synchronizing at all.

- Thread Architecture: The Thread-per-connection architecture provides flexibility compared to worker-pool architecture and less overhead cost constructing thread compared to Thread-per-request architecture in a quite long-term service. However, when clients just run the application and do nothing, the threads resources are still be occupied. The disadvantage causes resources inefficiency and may affect the server's scalability. It is hard to compute the inefficiency cost compared to its benefits with the subject's knowledge. The justification for choosing the Thread-per-connection architecture is the intuition-kind assumption that typical dictionary users would like to make multiple times of operations once connected just like online shopping orders - once connected, many orders. And with the *60 seconds silence out rule*, the system could shut some zombie clients and recall some threads resources in some extent.
- User Interaction: When connection lost, clients will got error messages and some suggestions when trying to make operations. However, more active approaches could be implemented, like automatically returning to login interface when connection lost. The active approaches may need notifier-listener design pattern and have some bad consequences like losing page information as well.

5 Conclusion

In the project, a single server - multi client online dictionary system was implemented with explicit use of sockets and threads. For interaction, the application uses TCP protocol and JSON object. For threads architecture, the system uses Thread-per-connection architecture. Both server and client application are interacted with GUI. The decisions of concurrency handle and thread architecture have their own benefits and limits, which remain huge improvement potential.