COMP90015 Project 1 Report

Haonan Chen 930614 haonanc1@student.unimelb.edu.au

1 Introduction

The project requires to implement a multi-threaded dictionary system using client-server architecture. In this system, each component should be either a client or a server. No component can be in both roles. The dictionary system allows clients to query an existing word, add a new word and remove an existing word. The word and its definitions are stored as type "String". The system should allow clients to query, add or remove a word concurrently, without raising any problems of interleaving. Each action of clients should be atomic, which assures the consistency of data in the dictionary. The communication between clients and the server is via TCP sockets. Clients should be informed whenever errors happen, or the server is offline. Also, the status of server should be presented so that the administrator can know whether the server is operating normally. Message exchange protocol chosen in the project is JSON. The request and response in communications are written in JSON files.

2 System Description

2.1 Architecture

The system implements a two-tier single-server multi-client architectural model. Each connected client runs as a Java thread to send the request to the server and use dictionary service. Each client thread has the same priority to be served. Once connected, each client can send more than one requests sequentially via connection. To make the system more efficient and suitable for the highly concurrent situations, this system implements a Worker Pool architecture. Compared with Thread-per-request architecture, the Worker Pool architecture reduces the cost of creating new threads and increases the reusability of existing threads. Compared with Thread-per-connection architecture, Worker Pool can have a better performance in the highly concurrent situations. When there are a lot of the concurrent requests, Worker Pool has a queue to manipulate the requests from clients. In this way, we can reduce the load pressure of the server and ensure the high availability of the server. Thread-per-connection architecture may force the server to serve requests from many clients simultaneously, which increases the load of server and may cause the server to crash, losing the availability of the system. Although Worker Pool architecture may be more difficult to implement than other architectures, this issue can be well handled by professional software engineers.

2.2 Concurrency

In this system, the shared resource of all candidates is the dictionary. Any request from one client that changes the content of dictionary (such as adding a new word or removing a word) should be reflected to other clients. Access to shared resource should be synchronized at the entry level. The atomicity of requests from clients should be guaranteed. No concurrent operations on the same

dictionary entries are allowed. If there are two requests from clients that implement "conflicting" actions on the dictionary, such as one "remove" request and one "query" request on the same word, these two requests cannot be handled simultaneously. Concurrent should still be allowed on different dictionary entries.

2.3 Service Operations

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

Query: Search the meaning of a word.

Parameter: the keyword to be queried

Output: the status of operation (“Success” or “Failure”). The definitions of keyword when the operation succeeds. The reason of failure when the operation fails, eg: “Cannot find the word”.

Add: Add new word and its definitions to the dictionary. The word cannot already exist in dictionary.

Parameter: the keyword to be added and its definitions.

Output: the status of operation (“Success” or “Failure”). A message shows the new word and its definitions when the operation succeeds. The reason of failure when the operation fails, eg: “Word already exists, or No definition is found”.

Remove: Remove an existing word from the dictionary.

Parameter: the keyword to be removed

Output: the status of operation (“Success” or “Failure”). A message shows the word is removed when the operation succeeds. The reason of failure when the operation fails, eg: “Cannot find the word”.

2.4 Interaction

All interactions between clients and the server happen through Java socket interface using TCP protocol. This system must use TCP protocol because the clients care about the arrival of response from the server. Using UDP protocol cannot guarantee the arrival of response. The socket-based programming can be easily implemented for general communications, and it allows the dictionary service to cover a board range of clients.

This dictionary system uses JSON as the format of information exchange because the conversion between JSON and String is simple. Also, JSON object is like the HashMap data structure in Java, the cost of retrieving the value of a corresponding field in request JSON is only O(1), which is efficient.

The request JSON from clients has at least two keys: “Task” and “Key”, which corresponds to the type of action (“Add”, “Query” and “Remove”) and the keyword the action performs on. If the action is to add a new word, the request JSON should also include a field “Value”, which corresponds to the definitions of the word. The definitions of the word should be stored in an array of String. For example:

{Task: “Query”, Key: “Apple”}

The server accepts the request JSON via socket, and then parses it to String, making corresponding operations. The response from the server is also in JSON format. The response JSON includes three fields: “Action”: the type of task; “Status”: the status of the task; “Message”: show information about the results if success or the reason of failure when the task fails. For example:

{Action: “Query”, Status: “Success”, Message: “[A Fruit]”}

2.5 GUI

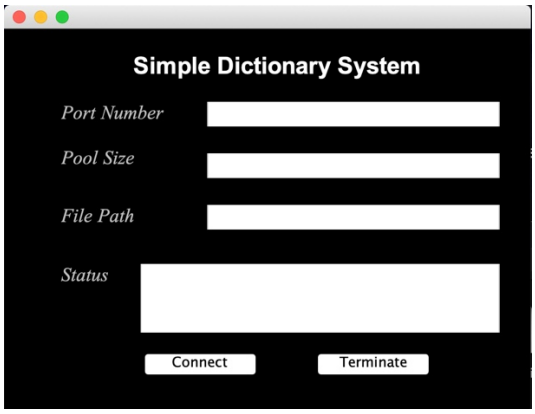
The system provides GUI for both client and server using Java Swing Window Builder tool. The background is designed as black for a taste of minimalism. There are in total three pages of the system GUI, one for the server and two for the client.

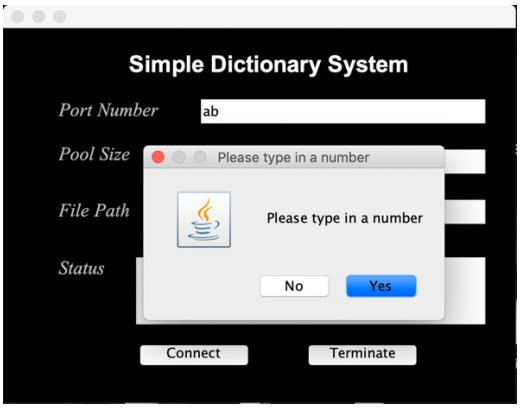
1. Server Login Interface

The administrator can use this page to launch the server. There are three input fields in this page: port number, pool size and file path. The port number indicate the port that the server should listen to. The pool size indicates the size of worker pool associated with the server. The larger the pool size is, the more efficient the system will be in the highly concurrent situation.

The port number and pool size must be typed in number, any incorrect input will trigger a pop-up window, warning the user to type correct input. The file path must be an absolute file path in the local machine, which indicates the location of original dictionary JSON file.

The status is an output field which shows the status of the server. If there is any exception happens, the exception messages will be presented in this field.

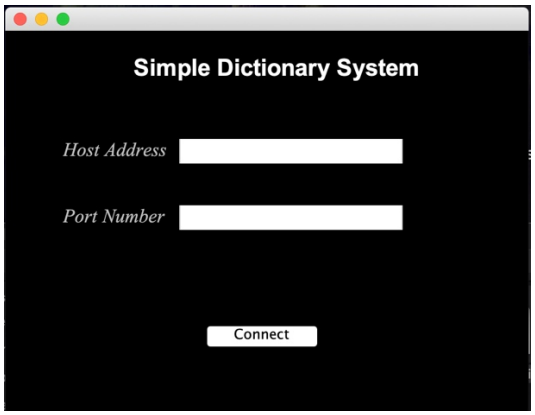


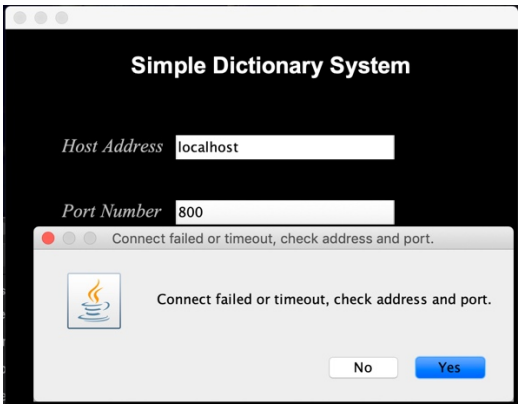


2. Client Login Interface

The client can use this page to launch the client. There are two input fields in this page: port number and host address. The port number indicate the port that the client sends the request.

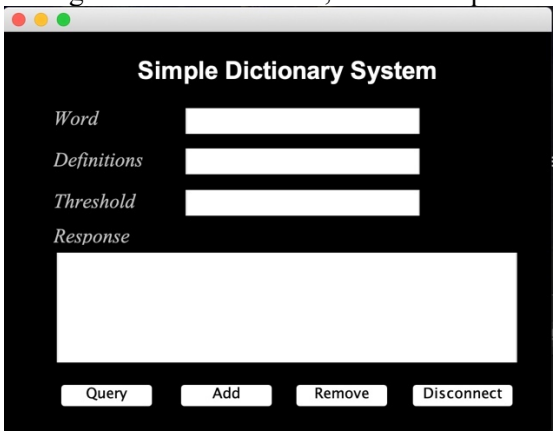
The host address must be in correct format or String “localhost”. The port number must be typed in number. Any incorrect input will trigger a pop-up window, warning the user to type correct input. Also, fail to connect to server will trigger a pop-up window to inform the user.





3. Client Operation Dashboard

Once connected successfully, clients will have operation dashboard which he/she can choose the dictionary functional services or disconnect the client and logout. The Response field can show the client the content of response from the server or reasons of errors when the operation fails. This interface also allows the user to set a time threshold for the request, if the response time of the server is longer than the threshold, then the request is treated as “failure”.



2.6 Errors

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

1. AbnormalCommunicationException

Invoking Issue: Errors occurred when the request time between client and the server exceeds the time threshold set by client user.

Interaction Message: “Request timeout, check the connection.” The error message will be caught and trigger a pop-up window on dashboard to inform the user.

2. ConnectException

Invoking Issue: Errors occurred when the connection between client and the server is interrupted or the server is shutdown.

Interaction Message: “Connect failed or timeout, check address and port.” The error message will be caught and trigger a pop-up window on dashboard to inform the user.

3. UnknownHostException

Invoking Issue: Errors occurred when the client cannot connect the server defined by client user, or the host address is incorrect.

Interaction Message: “Unknow host, check the host address.” The error message will be caught and trigger a pop-up window on dashboard to inform the user.

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

1. CloseFailureException

Invoking Issue: Errors occurred when the server socket cannot be closed, and the server cannot be terminated normally.

Interaction Message: “Connect close the socket.” The error message will be caught and shown in the “Status” field of Server Login Interface.

2. ParseRequestException

Invoking Issue: Errors occurred when the server parses the request JSON from the client. There might be something wrong in the format of request JSON.

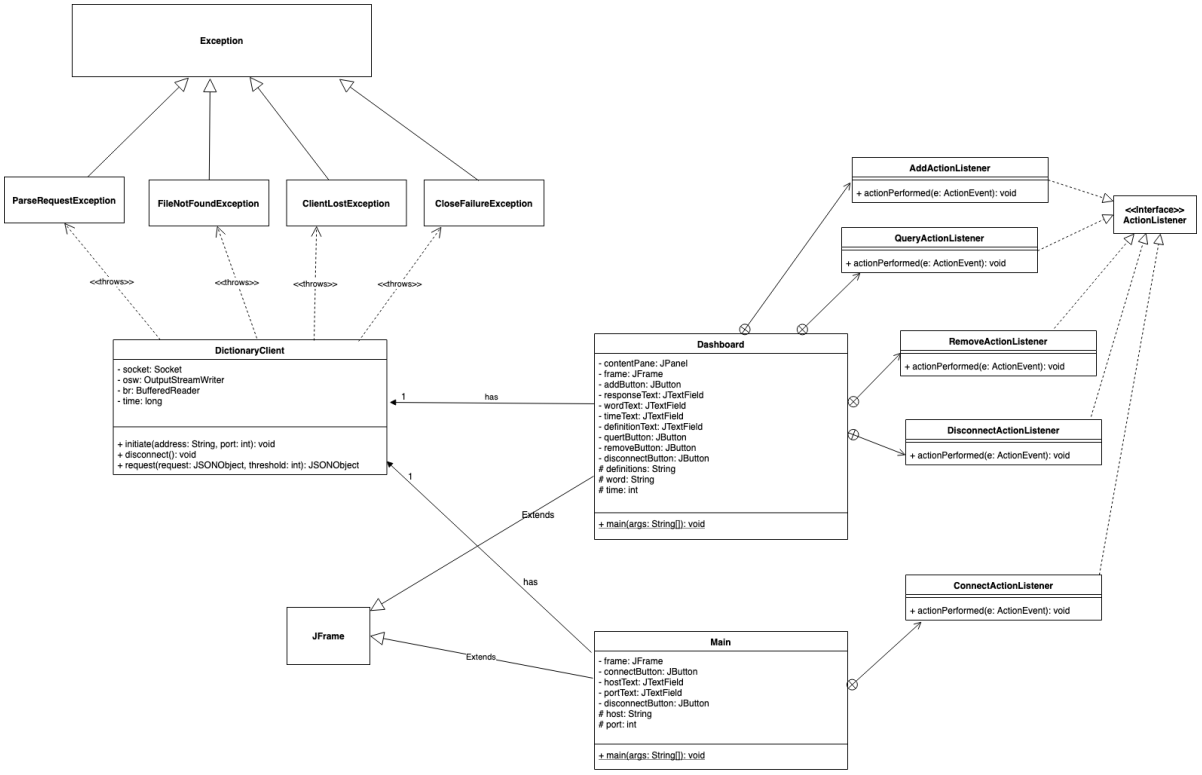
Interaction Message: “Errors happen when parsing the request json.” The error message will be caught and shown in the “Status” field of Server Login Interface.

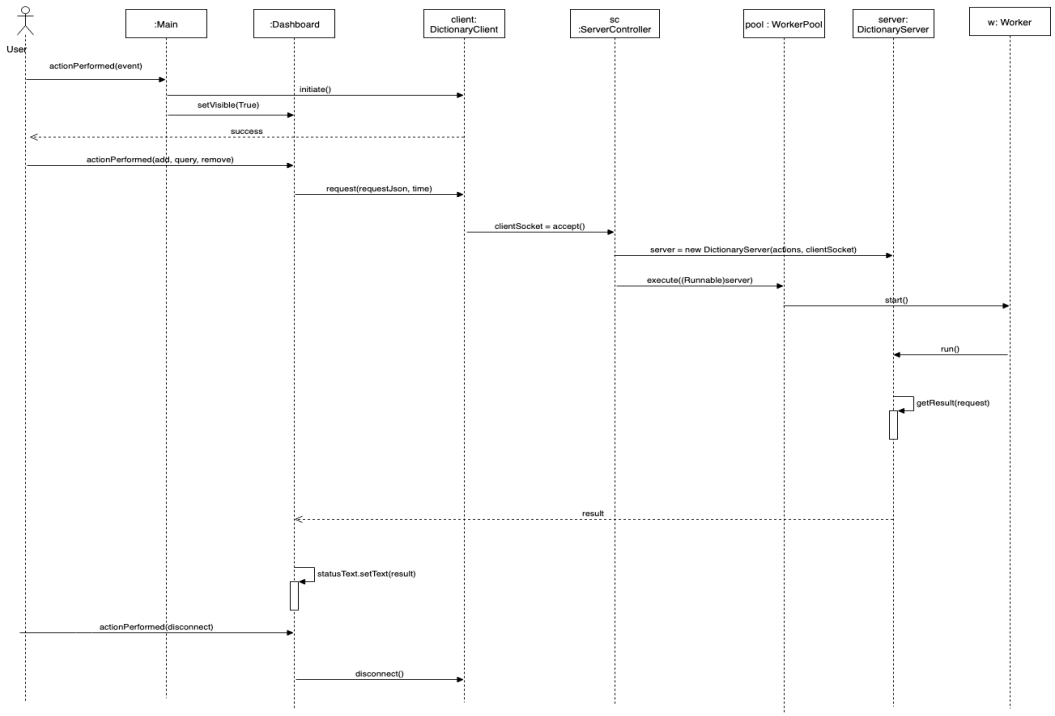
3 Class Design

The overall class design follows Java Swing framework, which encapsulate the distributed functionalities into graphic user interface.

3.1 Client

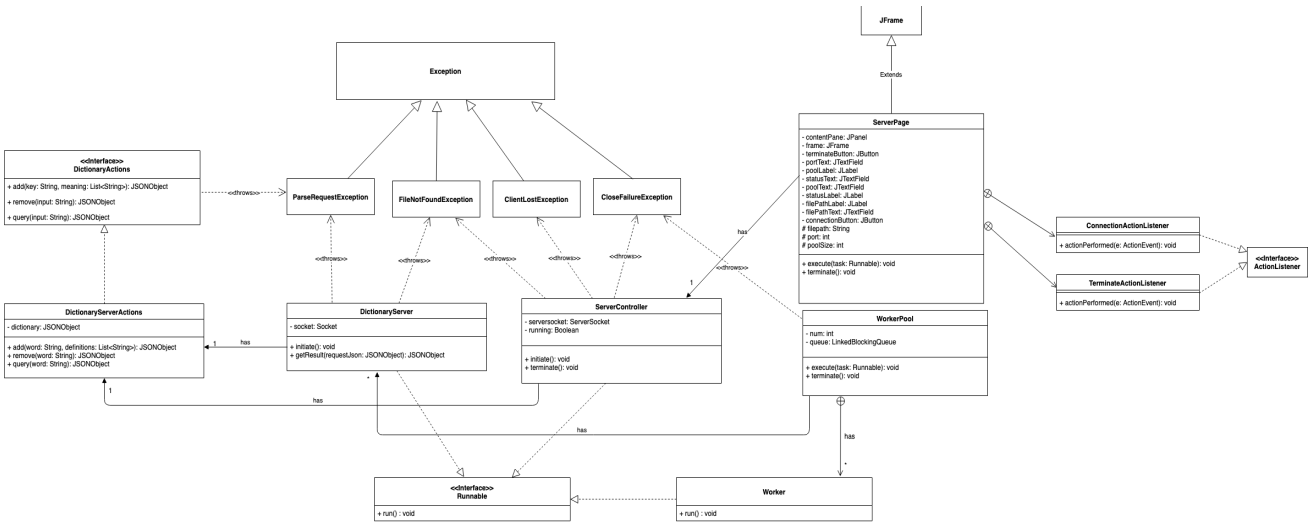
Main class contains a DictionaryClient object. When the user types in the information and click “Connect” button, the DictionaryClient object will be created if the connection succeeds, and Main class will create a new Dashboard object to jump to Dashboard page. In Dashboard page, the client user can customize the request and see the results of request in “Response” field.

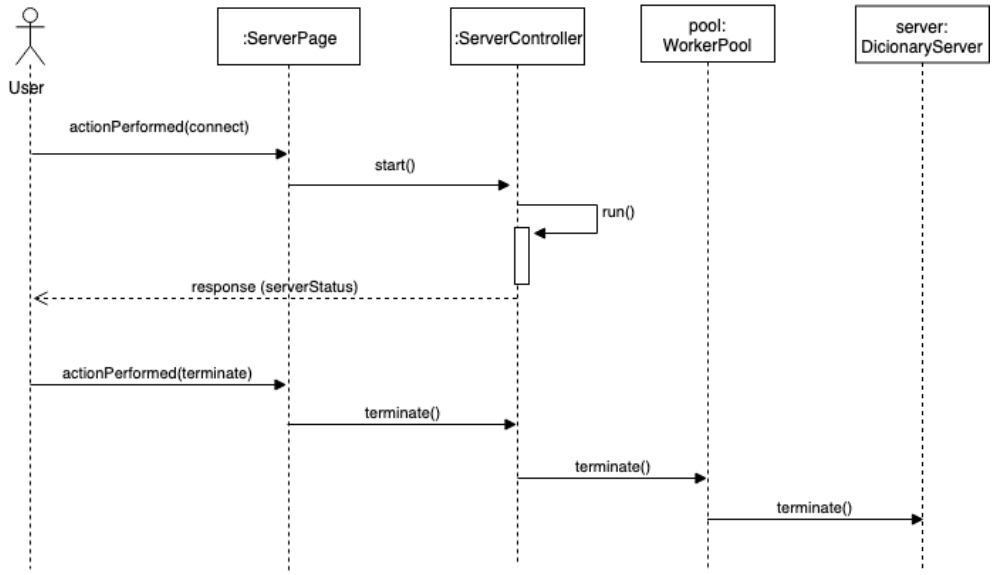




3.2 Server

DictionaryServer and ServerController classes implement Runnable interface. The thread of ServerController is separated from the thread of ServerPage. A ServerController accepts socket from clients and distributes the workload to DictionaryServer threads through Worker Pool. The DictionaryServer thread deals with the client request and sends response to the client. The dictionary service is encapsulated in DictionaryServerActions class.





4 Critical Analysis

Concurrency Management:

Multiple clients can access the shared dictionary resource concurrently, however, they are not allowed to access the same dictionary entry at the same time. The synchronization of access to the shared resources is handled by Java “Synchronized” methods. In this way, the system can be prevented from data inconsistency raised by concurrency access of multiple clients. For example, there is a word “apple” in the dictionary, two clients send requests concurrently. Client A sends a “remove” request and Client B sends a “query” request on the same word. The synchronization of data access protects the data integrity. If the system handles the “remove” request firstly, it will not allow Client B to “query” the same word at the same time. After the word gets removed, the “query” request from Client B will return “Failure” and indicate that the word doesn’t exist in the dictionary.

However, there are other issues cannot be controlled in this way, such as Unrepeatable Read. For example, when client A add a new word “Pear” into the dictionary with meaning AA. Then, the client B sends a request to remove the word “Pear”, and then client C add “Pear” with another meaning BB. In this situation, when client A sends a request to query the word “Pear”, he will find that the meaning changes surprisingly. This problem can only be handled at a database level and out of scope of the concurrency management.

Thread Architecture:

The Worker Pool architecture increases the reusage of threads and save a lot cost of creating threads, making the processing of requests faster and efficient. However, when the number of requests of clients is very low, the existing threads in the Worker Pool cannot be fully utilized, which is a waste of resources. Therefore, for applications of low concurrency situations, Worker Pool architecture may not be the first choice. Sometimes It is hard to tradeoff between the waste of resources and gained efficiency of Worker Pool architecture. The choose of Worker Pool architecture in this project is based on the assumption that many clients send requests to the server concurrently and we would like to reduce the instantaneous load and assure the high availability of the server.

Client Connection:

When connection is lost, clients will be informed by error messages. However, other approaches can be implemented to optimize the system, such as “60 seconds silence out rule”. The system can shut down the connection between server and zombie clients (who didn’t make any request in last 60 seconds) to free up resources for other clients. This implementation can improve the efficiency and effectiveness of the system.

5 Conclusion

This project establishes a single server - multi client online dictionary system, with Worker Pool as the thread architecture. The protocol of interaction between client and server is based on TCP and JSON object. This system also provides users of both ends with GUI which can inform the status of the connection. This system has its own pros and cons in the current design and implementation. Further improvement can be done based on the actual application circumstance.