COMP90015 Distributed System
Assignment 1 Report
Chenyang Dong
1074314

Problem Context

The aim of this assignment is to implement a multi-threaded dictionary server using a client-server architecture. In such distributed system, it needs to allow multiple clients to interact with the client and perform operations concurrently. All communication between the server and clients must occur through socket, ensuring the reliable connection. It is important to have proper error management on both server and client sides. This involves implementing exception handling to address errors such as input parameters, network communication, I/O to and from disk and so on. It is noteworthy that the application should be able to gracefully exit, ensuring that the termination process is smooth and orderly.

The dictionary encompasses several essential functional requirements, including queries for the word meanings, addition of new words, removal of existing words and update the meaning of existing words. These functionalities should be integrated into a user-friendly graphical user interface (GUI) within the client program. The system must also be able to handle any potential errors that may arise, ensuring that users are informed promptly while performing the operation.

System Component

Server Component

The server implements on a worker pool architecture, which is a core aspect of the system design. Upon server initialisation, the initial dictionary data from a file containing the initial list of words and their meanings will be loaded. Then, a set number of worker threads are generated to manage the client requests sequentially. These threads are systematically processed, one by one, as each thread will remain in a waiting state until it receives a notification. This activation takes place when a client forwards a request to the server, prompting the thread to execute its task. This method ensures efficient handling of incoming client requests and optimal resource utilization.

Client Component

The client acts as the bridge for sending and receiving data between the user interface and the server. Every time the user performs the operation, for example searching the word or update the meanings of a word, the client establishes the connection with the server. After the server processes the request and transmits the response back to the client, the connection will be closed.

TCP/IP Communication via Sockets

The client-server communication relies on the TCP/IP protocol facilitated by sockets. When a client initiates communication, a socket is created to establish a connection with the server to ensure reliable and ordered data transmission.

JSON Message Exchange Protocol

To facilitate meaningful and structured data exchange, JSON is chosen to serve as the message exchange protocol between the server and client. When the client sends the request, the data is structured into JSON objects, such as the word, meanings, and method. The server processes these JSON requests, executes the required actions, and responds using JSON-formatted messages. By using this protocol, it simplifies the data handling and allows clear communication between the client and server.

Example:

```
33 New connection assigned to worker: Thread 4
34 Request from client: {"method":"search","word":"
   example"}
35 Response: {"method":"search","response":{"data":["A
    representative instance!","A sample or specimen
   !"],"status":"success"},"word":"example"}
36 Thread 4 returned to the pool.
37 ************************************************
38 New connection assigned to worker: Thread 5
39 Request from client: {"method":"remove","word":"
   example"}
40 Response: {"method":"remove","response":{"status":"
   success"},"word":"example"}
41 Thread 5 returned to the pool.
42 ************************************************
43 New connection assigned to worker: Thread 6
44 Request from client: {"method":"add","word":"
   example","meanings":["A representative instance","A
    sample or specimen"]}
45 Response: {"method":"add","response":{"status":"
   success"},"word":"example","meanings":["A
   representative instance","A sample or specimen"]}
46 Thread 6 returned to the pool.
47 ************************************************
```

Figure 1: JSON Protocol Communication Example (Server)

```
 4 Sent Data: {"method":"remove","word":"example"}
 5 Received Response: {"method":"remove","response":{"
   status":"success"},"word":"example"}
 6 Sent Data: {"method":"add","word":"example","
   meanings":["A representative instance","A sample or
    specimen"]}
 7 Received Response: {"method":"add","response":{"
   status":"success"},"word":"example","meanings":["A
   representative instance","A sample or specimen"]}
 8 Sent Data: {"method":"update","word":"example","
   meanings":["A representative instance!","A sample
   or specimen!"]}
 9 Received Response: {"method":"update","response":{"
   status":"success"},"word":"example","meanings":["A
   representative instance!","A sample or specimen!"]}
10 Sent Data: {"method":"search","word":"example"}
11 Received Response: {"method":"search","response":{"
   data":["A representative instance!","A sample or
   specimen!"],"status":"success"},"word":"example"}
```

Figure 2: JSON Protocol Communication Example (Client)

Class Design

Server Class Structure

Presented in Figure 3 is the UML Diagram depicting the structure of the *Server* program. The core components of the server include the ***DictionaryServer***, ***Dictionary***, ***WorkerPool***, and ***WorkerThread*** classes. Each instance of the *DictionaryServer* process initializes a *WorkerPool* responsible for managing multiple ***WorkerThread*** instances. The ***Dictionary*** class plays a pivotal role, facilitating synchronized operations on both the dictionary JSONObject and the dictionary file.

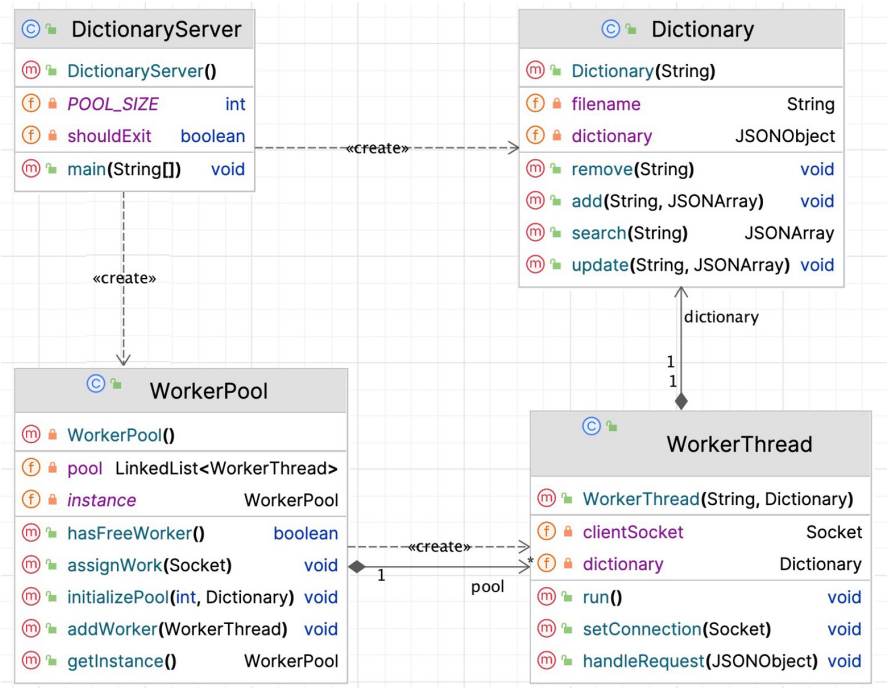


Figure 3: UML Class Diagram (Server)

Client Class Structure

In Figure 4, the UML Diagram is presented to illustrate the structure of the Client program. The key components include the ***DictionaryClient*** and ***GUI*** classes. ***DictionaryClient*** class effectively manages communication with the server by constructing and dispatching JSON-based requests for various operations. The ***GUI*** class provides users with an intuitive interface. As users input data and interact with the graphical elements, the ***DictionaryClient*** class coordinates necessary actions for interacting with the server.

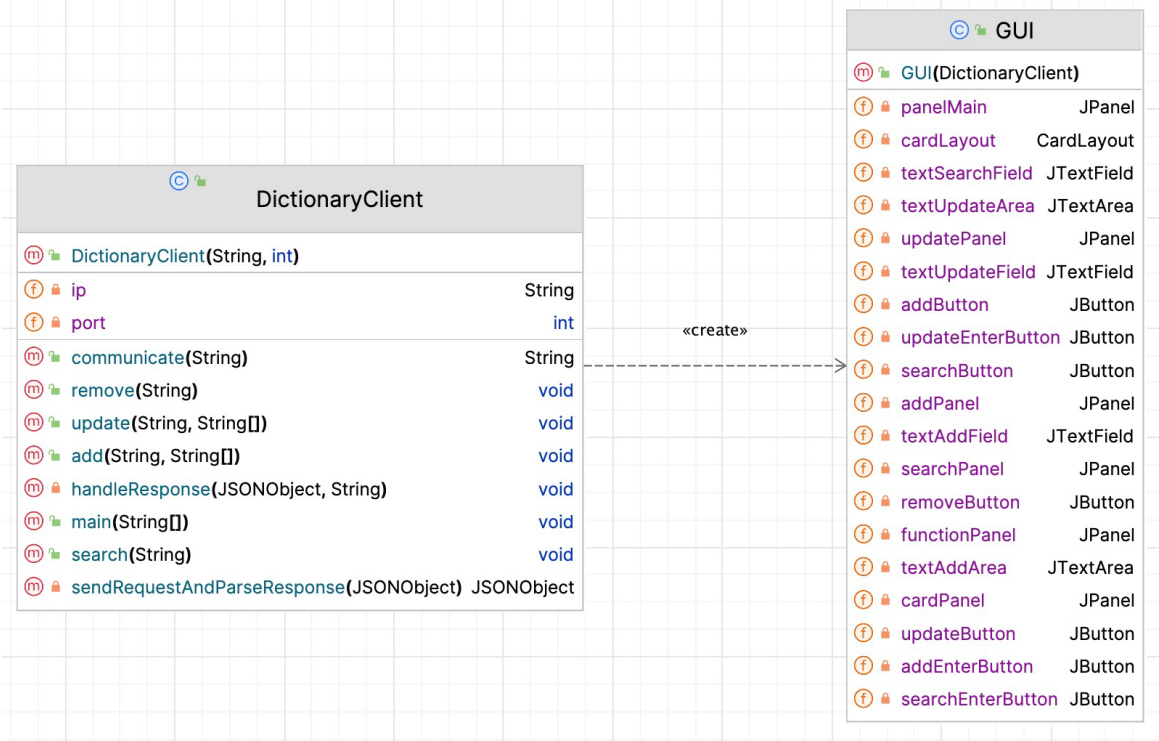


Figure 4: UML Class Diagram (Client)

Interaction Diagram

The interaction diagram shown in Figure 5 illustrates how the server responds to an incoming request from the client after a connection is established. The ***DictionaryServer*** assigns the work requested by the client to the ***WorkerPool*** which is further allocated to the ***WorkerThread*** by order. Once the request has been handled and processed to the Dictionary, the response will returned to the client.

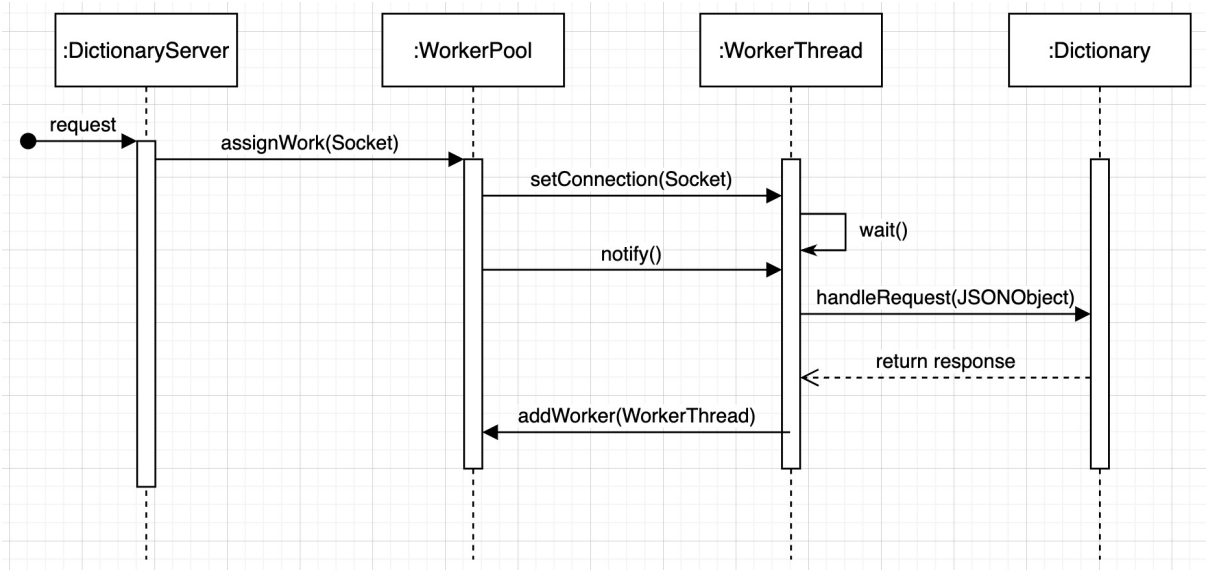
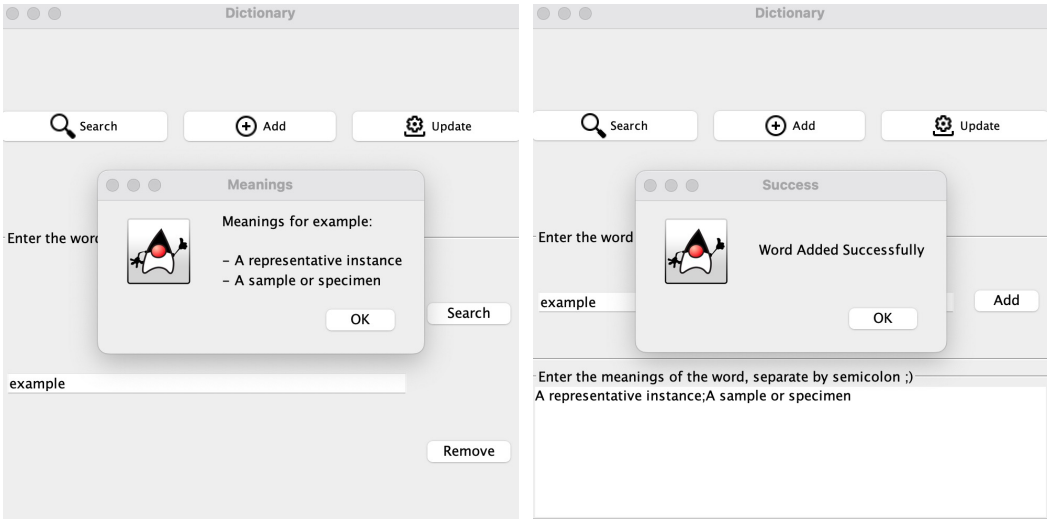


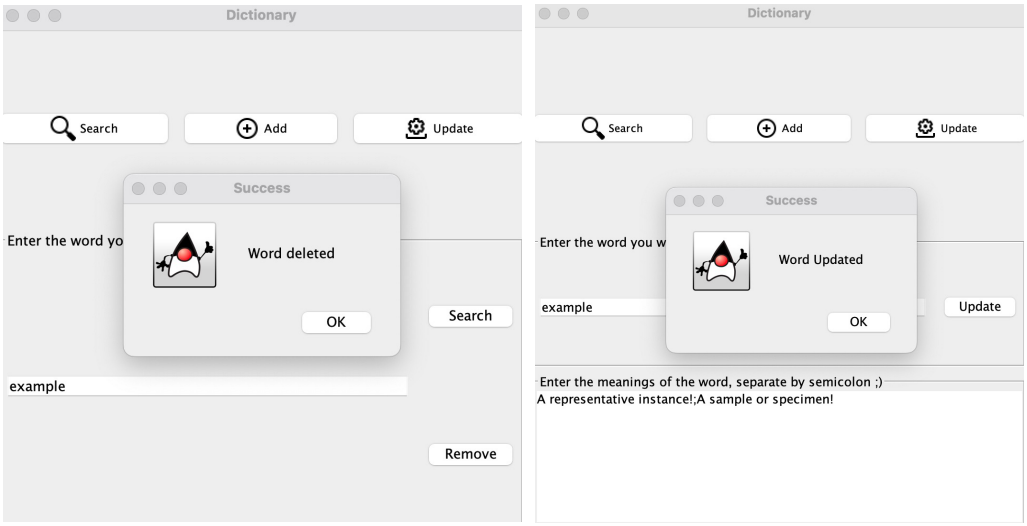
Figure 5: Interaction/Sequence Diagram

Critical Analysis

Functionality

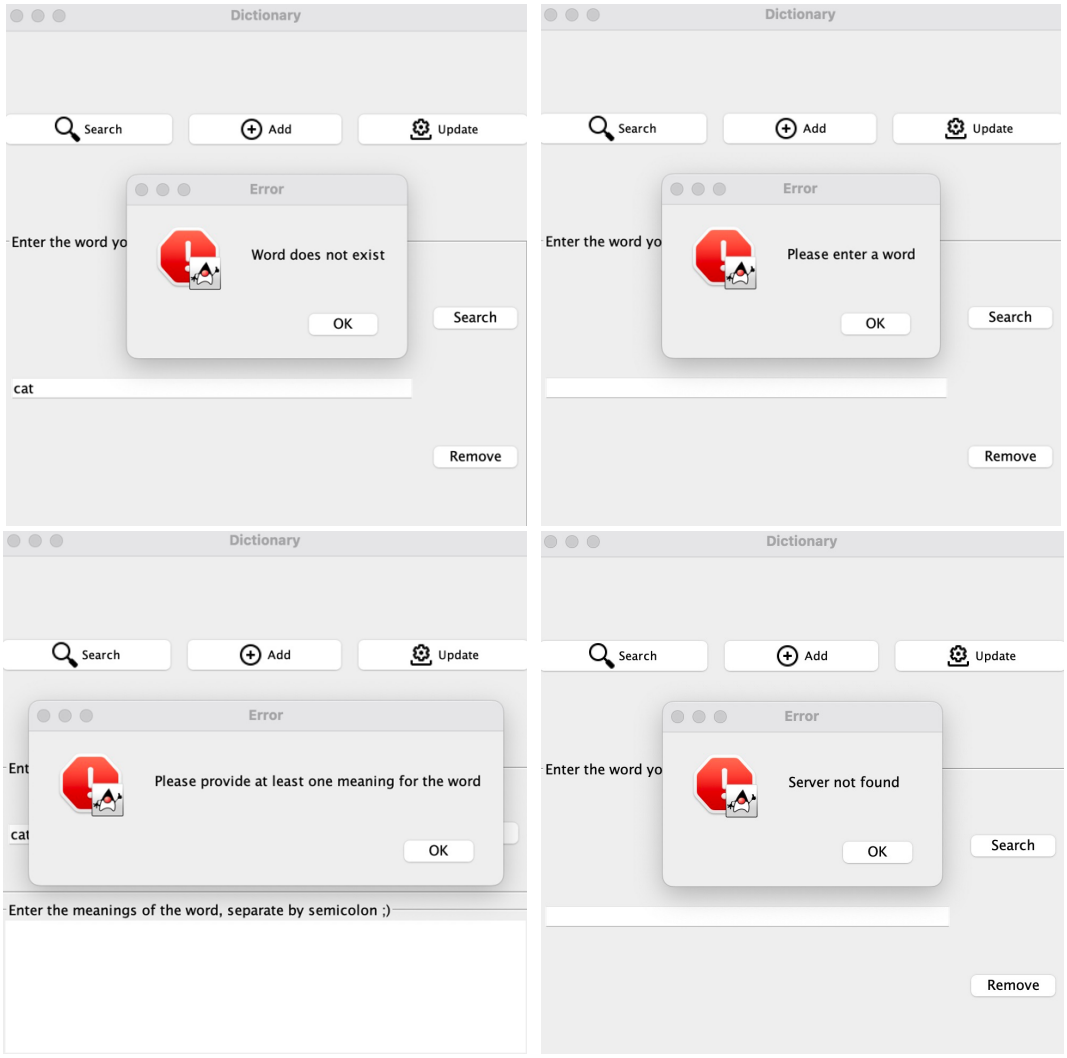
All functional requirements have been satisfactorily fulfilled as shown in graphs below. Through the systematic implementation of various features and components, the system effectively meets its intended goals.





Error Handling

As the examples shown below, whenever a user initiates an incorrect request, the server promptly responds with an informative error message shown in a dialog, guiding the user towards the correct usage. This enhances the user experience by minimizing confusion and facilitating correct interactions.



Advantage of System

The utilization of TCP sockets allows for reliable and ordered data transmission between the client and the server. The integration of a worker pool architecture ensures that all operations related to shared objects are synchronized, which further enhances the system's capabilities by facilitating proper concurrency management. This allows that multiple client requests can be processed simultaneously, improving the system efficiency and responsiveness. Error handling mechanisms have been thoughtfully implemented on both server and client sides.

Future Improvement

More advanced scalability strategies could be implemented in the future to allow dynamic scaling based on the incoming request load. Also, to help prevent overloading of specific resources and enhance overall system performance, load balancing mechanism can be introduced. Besides, database could be integrated to persistently store data for better data management and retrieval.

Conclusion

In conclusion, the system's architecture relies on a worker pool for effective task handling. Communication between the client and server is achieved through TCP using sockets, facilitating reliable interaction. JSON is employed as a clear and structured format for message exchange. Effective error management is implemented on both sides to handle issues like input errors and network problems, ensuring a smooth and orderly termination process. The functionalities of the dictionary including queries for the word meanings, addition of new words, removal of existing words and update the meaning of existing words are integrated into a user-friendly GUI within the client program, allowing prompt user notification in case of errors or issues during operations.