Karan Katnani
Assignment - 1
COMP90015
5 September 2018

Multi-Threaded Dictionary Server

The focus of this project is to implement a dictionary system, with a client side application to manipulate the dictionary data. The system supports multiple concurrent users by means of multi-threading, where communication is handled using Java Serialisable Objects and JSON. The client and server side have user friendly GUIs to aide usability, while the backend implementation includes data structures like HashMaps to accelerate real-time performance.

Brief Description

An application which uses the basic client server model, which leverages multi-threading to support multiple clients is required, where all communication takes place over sockets, and implements a message exchange protocol. The application focuses on the basics, where multiple clients (client side application) can connect to a dictionary server at a time, and search, insert and delete words and their meanings in the dictionary. The application also pre-loads a text file which contains a dictionary of its own, essentially preloading files in the dictionary, to be made available to the user. Both, the client and server are able to specify the port numbers they run on. The client and server both run on Java, and the server may implement any multi-threading architecture.

Introduction

The application revolves around a real-life client server system, where multiple clients are able to communicate with a single server concurrently. This allows multiple users to share a common resource running on the server. The server is generally is a high performance computing unit which is able to handle large workloads, and run a large number of concurrent processes, to ‘service’ all the clients with minimal lag time.

The server needs to establish multiple connections at one time, which is possible using ports on the socket. But, the multiple sockets cannot be broadcasted to the client. This issue is taken care of by using sockets, where the server is listening for client connection requests only on a single port, and then opens another port to service the clients request. This is essentially an abstraction layer, to reduce complexities of the architecture. A new thread is created for every request, which is attached to a socket or

port for communication. All data transactions take place using JSON, which has been implemented using the Java Serialisable Object Class, and a 3rd party ‘gson’ library.

The system uses a reliable connection oriented protocol called TCP over which the sockets run. It supports full duplex communication over an established channel, but has minor overheads for ensuring the reliability.

A thread per request architecture has been built, where connection is established only until the given request is completed. A new connection is established every time the client sends a new request.

Components of the System

Client Application:

The application which a user opens, to access and manipulate the dictionary server. The client can enter the port number in the application itself, allowing easy access to multiple dictionary servers in case they exist. This consists of 2 classes, where one provides the UI, while the other acts as the middle-man, establishing communications with the server, providing the server with inputs from the UI, and giving back the responses from the server, to the UI.

Server UI:

The server also has a UI, used for loading the dictionary file, entering the port number, and starting and stopping the server.

Dictionary Server:

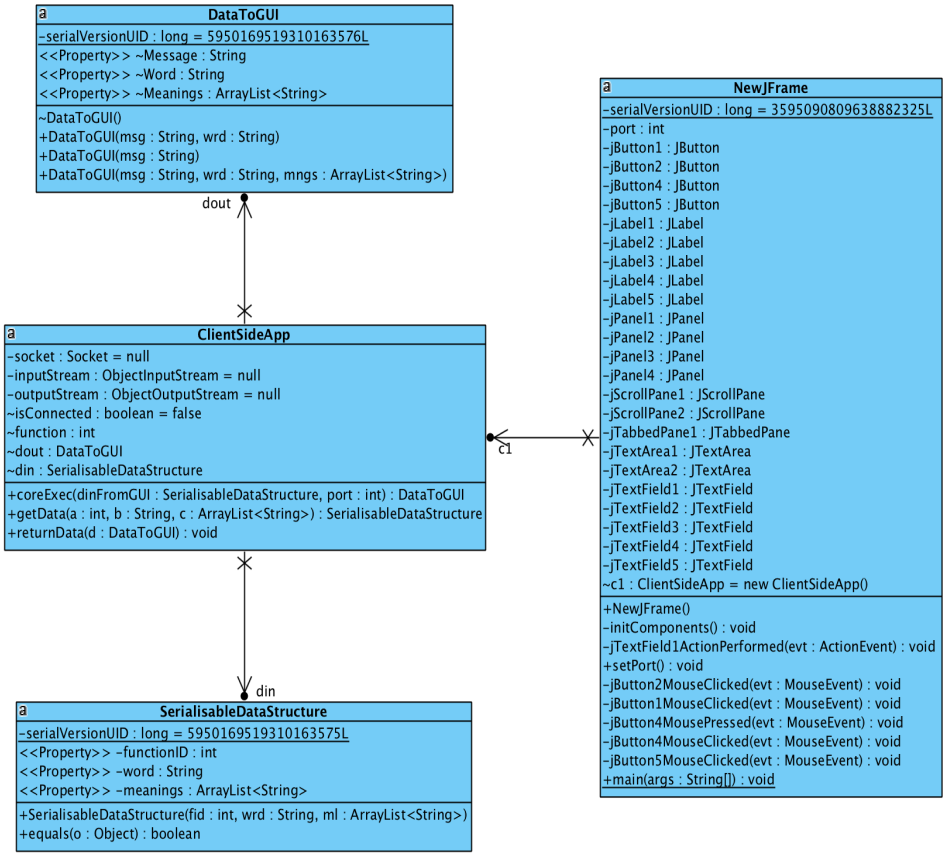
The server has been configured using multiple classes, where one class is responsible for maintaining the data: loading the external file and also maintaining the dictionary data. Another class provides the add, search and delete functions, as selected by the client. The other 2 classes are generic hosts, where one starts and stops the server, while the other maintains client interaction, and makes the functionality available to the client over sockets.

UML Diagrams:

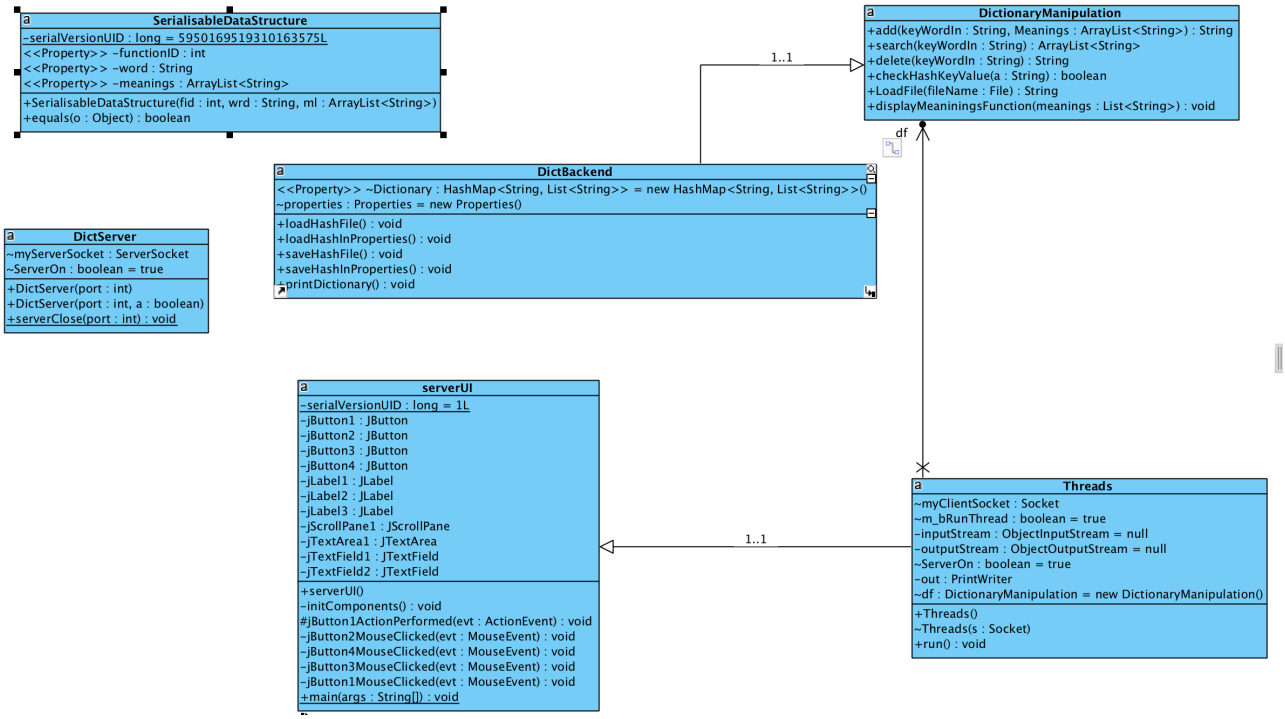
Class Diagram:

This represents an architectural overview of the source code, showing the design of code at a higher level.

Client Side Application:

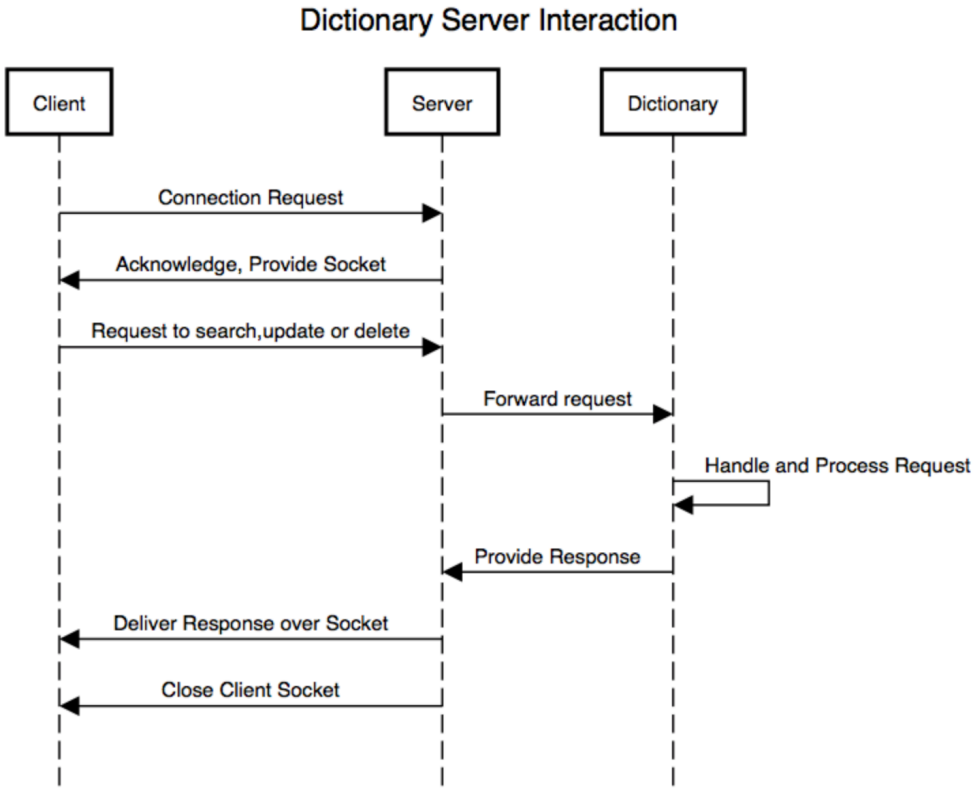


Server:



Sequence Diagram:

This diagram shows a highly abstracted view of how the system functions, by showing the sequence of actions.



As shown in the diagram above, a client initiates a request, which is then acknowledged and connected, and the user is provided a socket for individual communication. The client then sends the data and function to be called, encapsulated in JSON. This is forwarded by the communications server to the backend, which then parses the message, calls the equivalent functions, and manipulates the data in the data structure. The backend generates a response, again encapsulated in JSON, which is delivered by the server to the client, over the established socket.

Critical Analysis

The current system can be compared to an alpha release, which has not been tested, and may have run-time bugs. There may be errors which have not occurred in ideal developer testing, and have thus not been handled. The UI is rudimentary, and provides just enough for the user to be able to perform the specified tasks.

The system uses a TCP base with a thread per request architecture, which may have unnecessary overheads, since such UDP is much more apt for the system, but has been skipped to reduce complexity.

The system is extremely basic and has been designed and run within test environments only. Due to this, load and response testing of the system is not possible.

There is scope for addition of features, such as implementation of user accounts to know who is manipulating data, and handling errors in a more user friendly manner.

Conclusion

The given problem of designing a client-server architecture based multi-threaded dictionary, with functionality to add, search and delete words with one or multiple meanings has been accomplished, using sockets over TCP for communications. The system is inherently reliable due to the use of TCP, but requires more work to become actually usable.

Bibliography

[1] https://en.wikipedia.org/wiki/Client–server_model.
[2] Davidson, J., 2012. An introduction to TCP/IP. Springer Science & Business Media.

[3] Calvert, K.L. and Donahoo, M.J., 2011. TCP/IP sockets in Java: practical guide for programmers