COMP 1549 Report - C.L.I.C. SYSTEM

Anna-Mariya Mineva Christopher Torrington Jasmine Bourne Tyroan Gordon

COMP1549: Advanced Programming

University of Greenwich

Old Royal Naval College

United Kingdom

Table of contents

Abstract –

We introduce C.L.I.C, a lightweight application facilitating communication between members on a distributed client-server network. C.L.I.C performs similar tasks to applications such as Skype and Slack.

This report covers topics such as the design and implementation, functionality and testing, classes, methods and attributes of a command line based distributed communication applications. Key results include unit testing using Visual Studio Code's (VS Code) testing framework, the use of inheritance, sockets, threading, command line interface for text input and output, client coordination using commands and time delays, as well as abstract classes.The combination of key results and topics forms the application C.L.I.C.

Key words: Client-server communication, Python Sockets, Python Multithreading, Inheritance, Abstract classes

I. Introduction

Our goal for this project is to provide a user-friendly interface with simple commands to allow members to connect as a group or to interact privately with each other in a terminal-like environment. We provide the users with a flexible and straightforward method of communication, with some form of regulation by the coordinator, who has a small set of commands to monitor the activity within the chatroom. It is important to note that the program is entirely written in Python programming language, and the use of various Python modules such as the threading module, socket module, sched module, and pickle module are necessary for achieving the objective. The scope of this project involves creating a multi-threaded application that allows the code to execute and perform various operations simultaneously (threading — Thread-based parallelism — Python 3.9.2 documentation, 2021).

C.L.I.C is a basic command line interface application that offers versatile control over the connected clients along with simple client to client communication, scheduled timeouts when detecting inactivity, and an administrator-like role, called the "Coordinator", designed to overlook the chat room environment and update information accordingly. C.L.I.C runs in the command line and requires no supplementary application installation for a user interface.

We address the designing of this project in a modular and agile way, using object oriented patterns and principles, and focusing on test-driven development through various stages . These are all discussed further and evaluated in the Design/Implementation and Analysis/Discussion sections of the report.

II. Design/implementation

Visual Studio Code unittest is a python extension which is a built-in framework that supports testing with unittest, pytest and nose (Code, 2019). Using the built-in discovery tool to enable the test framework in VS Code. This function discovers any tests that have been built, as well as having the ability to run the discovery tool on the fly. Once run, it enables testing features that allow you to debug and run tests simultaneously for all new unit tests written(Code, 2019). Visual studio code also supports the use of sharing, automation testing, shutdown and aggregation of collected test code, plus easy reuse of code.

A unit is a piece of code within the program that needs to be tested. These consist of methods and functions of the program's classes. A Unittest is a second piece of code that is used to exercise the unit code, testing its functionality, boundaries and edges of that unit (Code, 2019). Unittests do not consider the implementation and are only concerned with

the interface, these the arguments and the return values of the unit. This permit's the running of tests without the need for dependencies because the test code is a low resource cost.

Running unit tests is quick and often helps catch regression in the code, this is completed by following the process of invoking a functions argument and asserting the expected outcome. The framework code compares the output of the unit against the asserted output. If true, the test will return a passed test and if false, return a failed test. The use of numerous variations of strings, null string, null input, SQL inject and HTML, ensures that in-depth testing is completed. In addition, it is best practice to run unit tests before committing the code to the repo (Code, 2019).

We take an object oriented approach for this project by applying modularity in our design. Beginning with the basic structure of our code, the class implementation can be split into individual parts consisting of the ClientServer class, the ServerClient class, the ClientCoordinator class and the DataTransfer class. Each of these classes, or modules, can be flexibly adjusted to fit the requirements needed to change the capabilities of the program. The program is built with the thought in mind that if more features are to be added, the modules that are already in place can easily be customised or readjusted to accommodate. When designing the program, we assemble the code by starting small, with just the ClientServer and ServerClient classes. The ClientServer class is the class which represents the server of the chat room. This class accepts the connections to the chat room via sockets and handles the sending and receiving of messages to the client side. ServerClient is the class which handles all of the client functionality, including setting up the user and handling various message types and commands. As the code grew based on requirements and features for the scope of the project, we incorporated more classes for balance based on the needs. When approaching issues such as sending assorted information across, we created the DataTransfer class. Our DataTransfer class is the main class used for the sending and receiving of data across the chat room. Apart from sending data across as simple "messages", we also need to send additional content such as information about the members joining, how many members are in the server, and whether they are trying to send regular messages across or are trying to send commands. The DataTransfer class is also inspired by the TCP (Transmission Control Protocol) network protocol. This network protocol structures the data into segments and includes a TCP segment header. By following the design used in TCP for data transmission, we structure a message header dictionary (header_dict) with variables based on what is included when we are passing data along sockets (socket — Low-level networking interface — Python 3.9.2 documentation, 2021). Finally we have our ClientCoordinator class, which we construct for running the scheduled timeouts due to inactivity within the server. This is passed onto the client and handled by the member who is appointed as "coordinator". Class implementation for the project requires the use of inheritance within the classes to tie in the functionality of the code. The DataTransfer class is inherited by ClientServer, ServerClient, and ClientCoordinator. The reuse of the DataTransfer class allows for us to manage the way data is handled without having to reiterate ourselves throughout the code. By simply calling the inherited class methods within methods in other classes, we can extensively implement the appropriate functions without duplication or redundancy.

Our objective when developing this program was to create code that would maintain operating status if some functionalities were to fail. Throughout the evolution of the source code design, the idea of testing as we go along is heavily emphasised for many reasons. One example is within the ClientServer class, the server's "receive" method was discovered to have a fault when collecting different data components sent over from the client side. As a result, this was one of the factors to contribute to the creation of the handle

method. As soon as the receive method "breaks" so to speak, we handle the error by either calling the handle method or disconnecting the client. This shifted our focus when developing further implementations within the code from whether there is a fault in the code to when there is a fault in the code, and what our approach will be for handling it. Consequently, we integrate try-catch exceptions as an effective solution to convert errors into exceptions when applicable. Every function that can fail is inevitably handled with exception handling. This is also remarkably useful for debugging potential errors when merging different design ideas and components of the code.

Roles for the components are defined based on their function. We want to limit the number of operations for each method, and refactoring plays a big part in doing this. Externally, the code functions the same way but by refactoring methods into smaller components we can have better control over unit deployment. When specifically looking at the Client Coordinator role, we create a method for checking whether a client is set up as a coordinator. We figured the first person to join is to be assigned the coordinator role. By organizing a check on the client side, we can ensure that a coordinator is always appointed at the inception of the chat room whenever clients are present. It removes the need to send a message saying "this person is the new coordinator" when every member already knows the person at position 0 in the members list will always take the role of coordinator.

Our test-driven deployment was an effective way to single out issues in the design as we were building the program. By testing a component immediately after it was created, we were able to improve aspects of the code as well as plan for the future development of the implementation of the design. After running a test on a new component, we can see what conditions are needed to implement the feature without failing the test. This also involved refactoring code as we went along with the test-driven development.

A UML diagram or Unified Modeling Language diagram is a way to help visualise how pieces of code relate to each other while also keeping track of their hierarchy within a piece of software. Our UML diagram will help to understand how each of the different classes are structured and how they interact with each other (Appendix A). The classes ServerClient, ClientCoordinator and ClientServer all inherit from the DataTransfer class, this is shown with the arrows and extend tag within the diagram. The reason for each of the main classes to inherit from DataTransfer is that it allows for each individual class to communicate with each other, for example the DataTransfer class allows for the ClientCoordinator class to communicate with the ServerClient class when the timeout warning is needed to warn a client that their connection is about to be closed if they don't interact with the program before the time is up. Within the UML each of the classes are split into three sections which are the name of the class, any attributes the class has and finally any methods within the class. By using this diagram we are able to see how each of the classes within a program are set up and what is contained within them in a clear and concise manner. The minus sign at the start of the attributes and methods show that they are considered private which means that they are not used elsewhere in the code.

III. Analysis and Critical Discussion

The logic behind the test-driven development of the code was to resolve any issues detected during testing of components as quickly as possible. The results revealed that for some of the methods within our classes to work, we needed to incorporate multi-threading, serialisation and deserialisation, and event scheduling modules for python. When we tried to encode different data types to send across sockets, we ran into a problem. This is when

the python pickle module was introduced to our code in the DataTransfer class (pickle — Python object serialization — Python 3.9.2 documentation, 2021). Pickle allows for serialising and deserialising Python object structures by using the pickle.dumps() and pickle.loads() functions. By converting the data to a byte stream on one side, and "unpickling" the data on the other side, we were able to successfully transfer the data.

Running unit tests on the code through its various stages of development was essential to catch bugs and logical errors. The core components of the program were tested to ensure that they worked as expected. The program passed all of its unit tests. The testing for DataTransfer().send() involved ensuring that the data received matched the data that needed to be sent. DataTransfer().recv() worked much the same way - ensuring that the data sent matched the data needed to be received. The program is only expected to have a maximum of three active threads at a time, however it is difficult to conduct a standalone test because it requires the full interactivity of the program. Thus we decided to test for it by printing the active thread count whenever it would be changed while the program was running. The active thread count never exceeds three.

From the initiation of the project, we knew we would need to involve threading so the program can execute multiple functions simultaneously. The Client role is expected to perform multiple things at a time, such as receiving messages at all times while still being able to send messages. However, it wasn't until we introduced event scheduling to the ClientCoordinator class for inactivity timeouts that we discovered we would need to incorporate another thread to handle the scheduler.run() blocking method (sched — Event scheduler — Python 3.9.2 documentation, 2021).

The coordinator is not chosen when needed but rather assigned to the first person in the list of active members. Every client will always have the same list of active members and because of that it is a simple check to see whether a member is the coordinator or not. This means that having an up to date list of the active members is essential and that is why we tested to ensure that members are removed correctly. To test member deletion the code from the relevant ServerClient() method was imported to the test method and ran on a simulated list of members. The list was then checked to ensure that the member to be removed had indeed been deleted. Member deletion passed the unit testing.

Coordinator reassignment is tested by ensuring that the first person in the list of active members becomes the coordinator. This is achieved through a method called whenever the list of active members is updated, which could change the active coordinator. The testing imports the code from the ServerClient() and checks the members type before and after the first member in the list is removed. In the test the first member is removed, the second member moves to the first position and becomes the new coordinator.

Inactive members will be removed from the server and so an inactivity warning was created to inform them that they will be removed if they do not become active. Testing this involved instantiating a ClientCoordinator object, giving it a member to warn and checking if that member received the correct warning after the appropriate amount of time had passed. The member did receive the warning after the correct amount of time.

If the member does not become active within the given time period then they must be informed that they have been removed for inactivity. To test this the same structure was used as with the testing for the inactivity warning however this time the test continued

until the inactive removal time was met. The member receives both warnings about inactivity and removal.

IV. Conclusion

Managing this project in an agile and object-oriented way, we were able to construct an easy-to-use CLI chatroom with abundant functionality. Through principles such as factory method patterns, refactoring, unit testing and exception handling we reduced the redundancy and dependency in our code.

In the system's current strage, it is a simple and operational command line interface application that offers flexible control over the connected clients along with simple client to client communication. There is potential for growth in the future of this program in which we can apply additional features that expand the range of services offered. One way to attain this would be by distributing the program across the internet. *This needs to be run on a router or on a machine connected to a router with port forwarding to the machine*. Adding an SQL server for storing data being shared amongst clients and keeping logs of the communication between clients and their roles would directly impact productivity of the system while reducing raw memory usage and helping maintain the lightweight aspect of the application. This would be a level of horizontal scaling which can greatly improve the efficiency of the system as it grows. Another way we can broaden the capacity of the system is to incorporate a Graphical User Interface for user-friendly navigation and additional accessibility to the clients. To further develop the system, the use of commonly practiced design patterns and unit testing implementations which were used during this stage of the project would still be relevant and effective.

References

Code, V. (2019) Testing Python in Visual Studio Code, *Code.visualstudio.com*, [online] Available at: https://code.visualstudio.com/docs/python/testing (Accessed 17 March 2021).

pickle — Python object serialization — Python 3.9.2 documentation, (2021) *Docs.python.org*, [online] Available at: https://docs.python.org/3/library/pickle.html (Accessed 28 March 2021).

sched — Event scheduler — Python 3.9.2 documentation, (2021) *Docs.python.org*, [online] Available at: https://docs.python.org/3/library/sched.html (Accessed 15 March 2021).

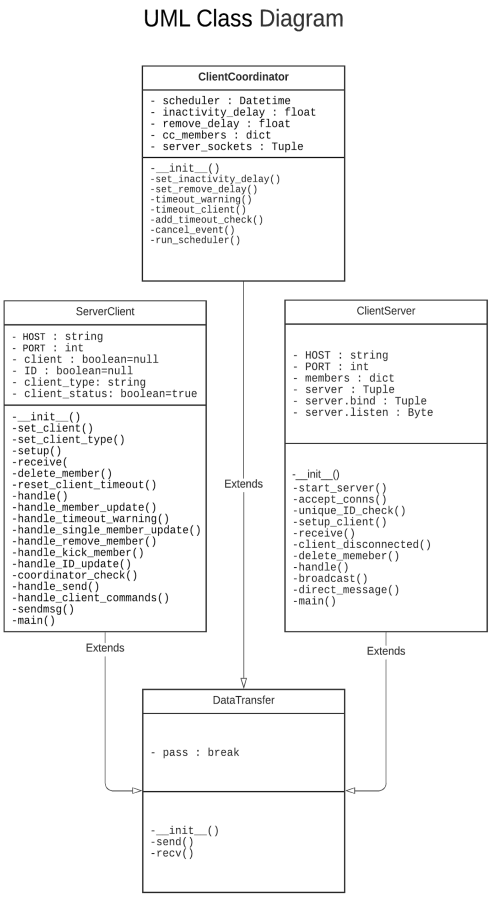
socket — Low-level networking interface — Python 3.9.2 documentation, (2021) *Docs.python.org*, [online] Available at: https://docs.python.org/3/library/socket.html (Accessed 20 February 2021).

threading — Thread-based parallelism — Python 3.9.2 documentation, (2021) *Docs.python.org*, [online] Available at: https://docs.python.org/3/library/threading.html (Accessed 25 February 2021).

Transmission Control Protocol - Wikipedia, (2021) *En.wikipedia.org*, [online] Available at: https://en.wikipedia.org/wiki/Transmission_Control_Protocol (Accessed 28 February 2021).

unittest — Unit testing framework — Python 3.9.2 documentation, (2021) *Docs.python.org*, [online] Available at: https://docs.python.org/3/library/unittest.html (Accessed 5 March 2021).

Appendix A:



Appendix B: