Report on ‘Shipper’’ Application

Patryk Wos, Second Year Student of (BSc Hons) Software Engineering

Staffordshire University

Author Note

*Includes User Guide, Design Treatment, Explanation of Strategy and Critical Evaluation*

Table of Contents

1. A word from the developer (page 3)

2. User Guide (page 4)

3. Design Treatment (page 5)

4. Explanation of the strategy (page 7)

5. Critical Evaluation (page 8)

6.Appendix (page 10)

…

**Report on ‘Shipper’**

This report is written fully to the needs of Software Engineering module   
‘Concurrent Network Applications’. And should not be considered professional product manual.

# User Guide

# Login into the server- to login into the server enter your nickname and click login button. To disconnect temporarily choose ‘disconnect’ option from options menu, to reconnect choose ‘connect’ option from option menu. To disconnect permanently close the window or choose ‘quit’ option from option menu. If you try to log in using already taken nickname you will not be able to connect to the server, login must be unique.

* **Basic chatroom -**to use chatroom functionality you need to be logged in and connected. Everyone connected to the server will be able to see messages in the chatroom. To send a message in the chatroom type in your message in textbox and press Send button or click Enter. To play higher or lower type in ‘HigherOrLower X’ (X being your chosen number)
* **Private messages -** to send a private message double-click a person nickname or avatar in clients list and type in your message in direct message window then click send button (arrow shaped button) or click enter.
* **Shipper -** to user original shipper functionality double-click anywhere on empty space in your main form, it is going to make a little shipper logo sail across the client on every single client connected.
* **Change avatar-** to change your avatar choose this option from options menu then choose desired avatar by double-clicking it or choosing it and then clicking Choose button.
* **Paint –** to use Shipper’s shared canvas Paint, choose it from Games menu. You will be able to see everything any other user painted on there from after you’ve joined paint room. To just hold LMB on white canvas and drag your mouse. To change the color or shape of your brush use buttons above the canvas. To clear everyone’s canvas click X button.

# Design Treatment

**List of functionalities implemented**:

* **Chatroom messaging**- Everyone connected to the server will see the messages  
  Everyone can start a game of higher or lower if it hasn’t been started already
* **Login into the system** – Every client can log into the system by typing in their nickname and clicking login.
* **Direct messaging**- Only 2 clients will be able to see their direct messages
* **Shipper game**- Everyone connected to the server will see the ship swimming through their client once activated
* **List of clients with avatars**- Everyone connected to the server will see the list of clients with their avatars
* **Change of avatar**- Everyone connected to the server can change their avatar
* **Shared canvas**- Everyone connected to the sever can open a shared canvas and paint whatever they like in there/ clean the canvas

There are few threads running on both server and clients that help keep the application working. Thread responsible for accepting incoming connection requests is running all the time, it’s the main server thread that opens other threads if needed. Server has got 2 separate threads for each client connected to the server, one opens on client start, second opens when UDP connection is needed (paint functionality). Those threads are responsible for reading packets incoming from clients and sending data across to the clients or just one client (TCP and UDP). Each client has got one thread responsible for reading incoming TCP packets from server stream and sending appropriate packets back to the server, another thread responsible for graphical part of the application, which is the shipper game and one more thread to read incoming UDP packets, process them or send them back to the server. My client class is also UI class (See UML Client diagram) which allows me to manipulate data from within the client. For packet classes I use class library which allows me to access different packets easily and change anything whenever I need it. List of clients connected to the server as well as clients who are disconnected but were connected is stored in Sorted Dictionary. Dictionary is a generic class, it allows me to store Sockets for UDP connection and TCP clients on the server. It also Allows me to iterate through collection of clients and send packets through to them using just one loop. I also store ID of avatars of clients on the server. It makes it more efficient than storing entire images on the server and sending them over whenever needed. Server runs 2 threads for each single client connected- one for UDP connection, one for TCP connection. There are 2 appropriate classes created as responders/listeners to UDP packets and TCP packets (see Server UML diagram). My Clients and Servers are Concurrent- they do multiple things at once, they are also synchronized- they do the same things at once on multiple clients (See concurrency diagram and other diagrams).

# Explanation of the strategy

To implement efficient, bug free solution I used mainly Kolb’s learning theory. I researched a lot about libraries and packages I need, I tried them first before implementing them, after learning enough I started implementing freshly gained knowledge into the project. Of course, mistakes were made but after I’ve made a mistake, I learnt from it and I never repeated it after. I tried making my code as clean as possible naming all the variables appropriately, sorting my classes in Class library and using a code cleanup to make things as tight as possible, saving up space and computer memory. After I’ve written the code, I’ve refined it multiple times to make sure my final product is of the best quality. I also stored older versions of working application to compare it to the current product. All those versions are available to download. My mutual exclusions are handled using simple threads, each client runs on a different thread and each client sends/receives just one packet at a time therefore there are no race conditions. To make sure I have enough time to process one packet before another comes, I simply put appropriate thread to sleep which doesn’t affect the application itself, but it gives enough time for it to process data. This solution is not ideal, I could have used confirmation packets or thread locks, but I wanted to avoid a deadlock and in a simple application like this my solution is more than efficient. In case of any buffer errors I throw an empty exception, which resets the buffer, packet loss is minimal, and it happens only with paint application so client cannot tell the difference.

# Critical Evaluation

In this section I will talk about what went wrong, what I’ve learnt and what challenges I have overcome.

Firstly, there is many things that went wrong, and I am not happy with, therefore I will continue to work on the project and application to optimize it and make it better. A simple example would be not being able to send large images across using the Stream my Shipper uses. Server wouldn’t be able to store Hash Table with list of all avatars and users and send it across due to lack of memory in my stream. I have figured out a way around it, however it makes adding custom avatars nearly impossible at this point. The graphical part of application is not really sophisticated and there is not a large amount of functionality to the client. It is purely due to lack of time and stress caused by work and other modules. As I mentioned before I will continue to develop this project in my free time. I have learnt a lot about Networking and concurrency. I now know how to properly set up a server that takes in packets and requests, processes them and then sends out another packets or requests to get more data or use data differently to help clients display what they are supposed to display or do. Once I’ve understood the idea it all became easy and pleasant to program. Time-consuming but enjoyable process. I have learnt a lot about serialization and streams of bytes, the possibilities that this open is nearly endless. When I first started, I did not know what I was doing, I barely knew basic functionality of windows forms and now I know exactly what and where to put in order to make my program work as intended. I became efficient in debugging. This option helped me solve loads of minor and major issues with my code allowing me to implement more functions in shorter time. What I regret is not sorting my code- it is messy, and you must look for specific methods in order to find what you’re looking for. If I was about to code this all over again, I would have planned my code structure first and care when adding new pieces of code. I would have also made the functionality a lot neater and use more design patterns to make it clearer and more efficient.

With time I will add some games that work online so clients have access to more functionality.

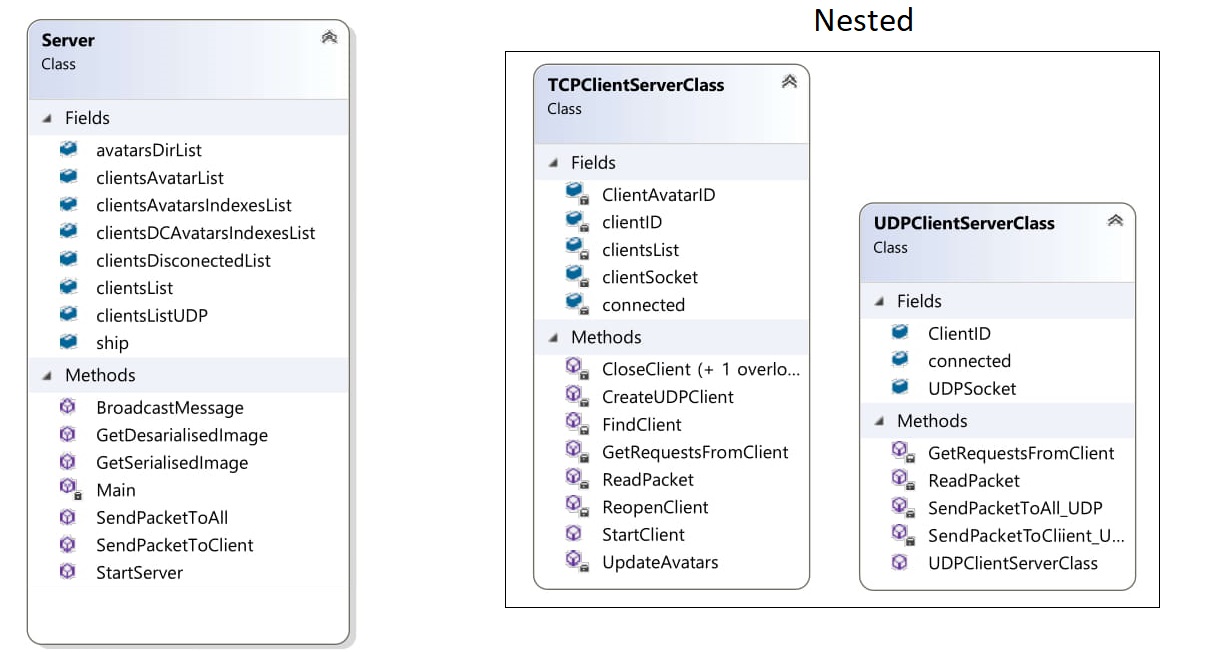
I wanted to refer to Kolb’s experimental learning cycle here because that learning theory helped me the most. When I first started nothing made sense to me. I didn’t know how to connect two clients together, or even how to use windows forms. But slowly, step after step I learnt. I was trying new things, coding everything I thought was right and everything I wanted my program to have. Then I was reflecting on the experience I had, researching more about the problems I faced. It helped me a lot because I did not make the same mistake twice, which saved me a lot of time, so I learnt from the experience just like in Kolb’s theory. Then obviously I would implement my freshly gained knowledge into my programming routine, making it better and better each day I was working on this project. Simple example might be usage of Hash tables instead of simple Lists or usage of Packets instead of distinguishing bytes using specific ascii signs.

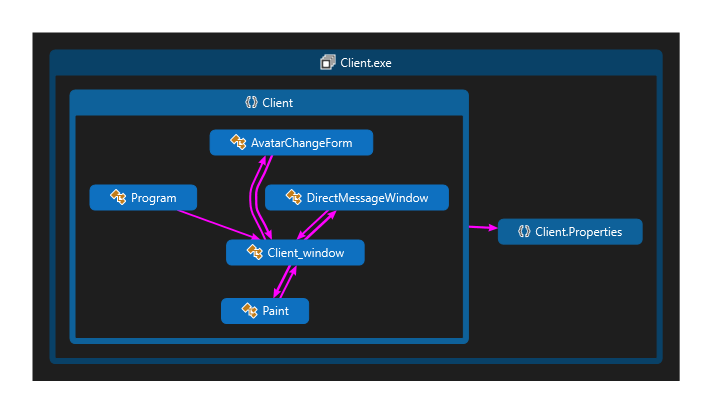
Another thing that’s worth mentioning is Rolfe et al.’s Reflective Model. It helped me a lot along with previously mentioned theory. With each problem I had I would just stop and think. What’s happened? Then I would think about what to do about this occurrence. That and a little bit of research would lead me to next steps, possibly problems or new solutions in my application.

It is really similar to Gibbs Reflective Cycle but in my opinion much easier to remember. Gibbs breaks it down to smaller problems- to your personal feelings, evaluation of the problem, making conclusions and action plants. With this small application it is not necessary in my opinion. I can learn a lot without making an action plan for each error I encounter. However, I do see its usability in large projects, especially with agile methodology and working as a group.

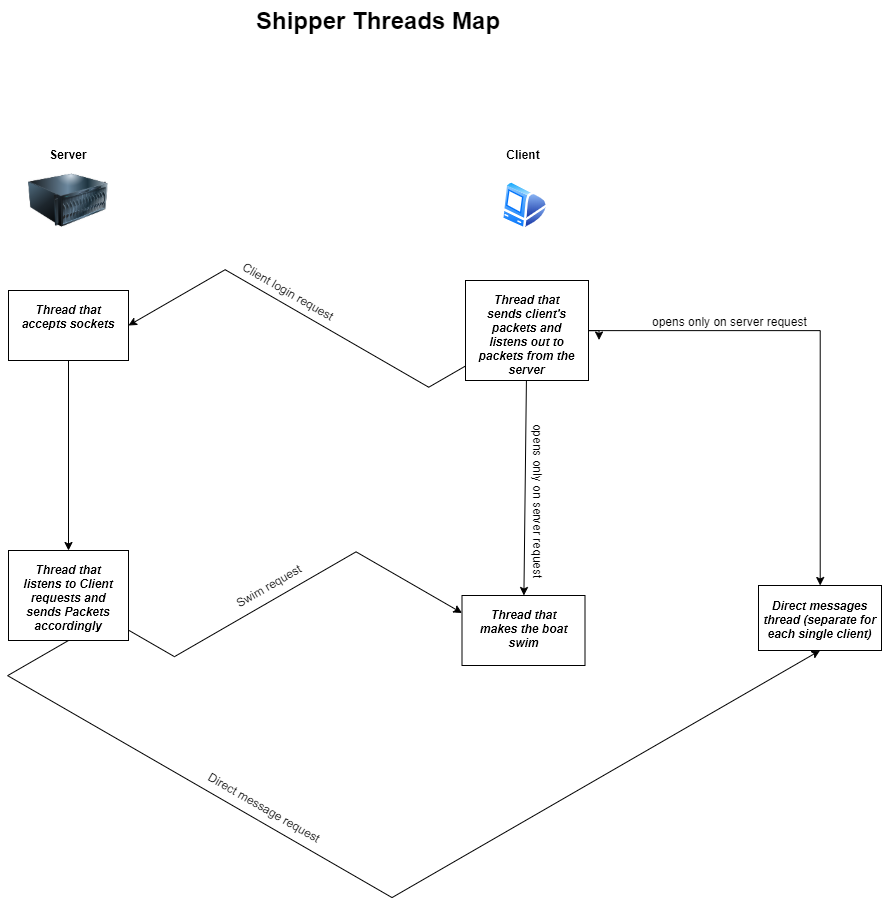
APPENDIX

) Client UML

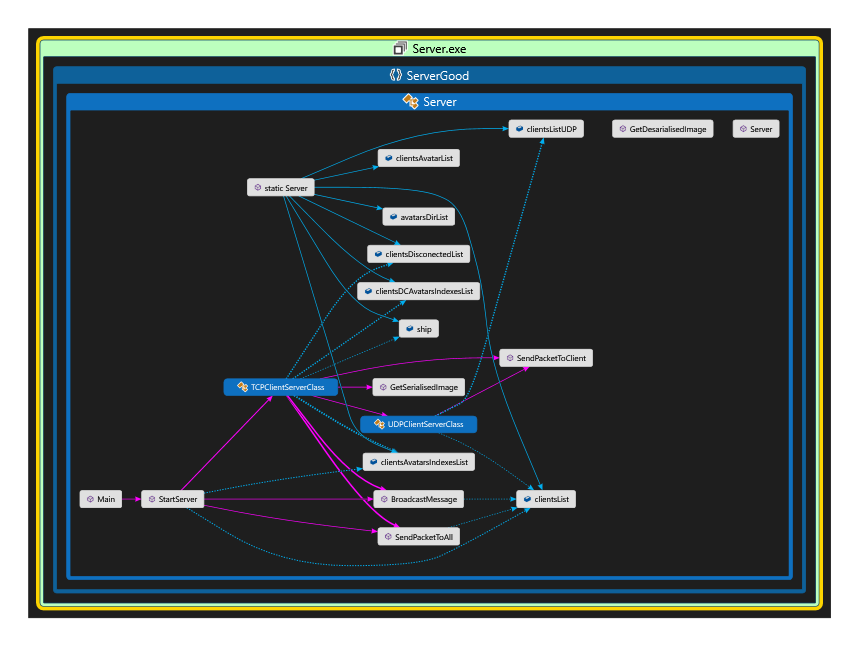


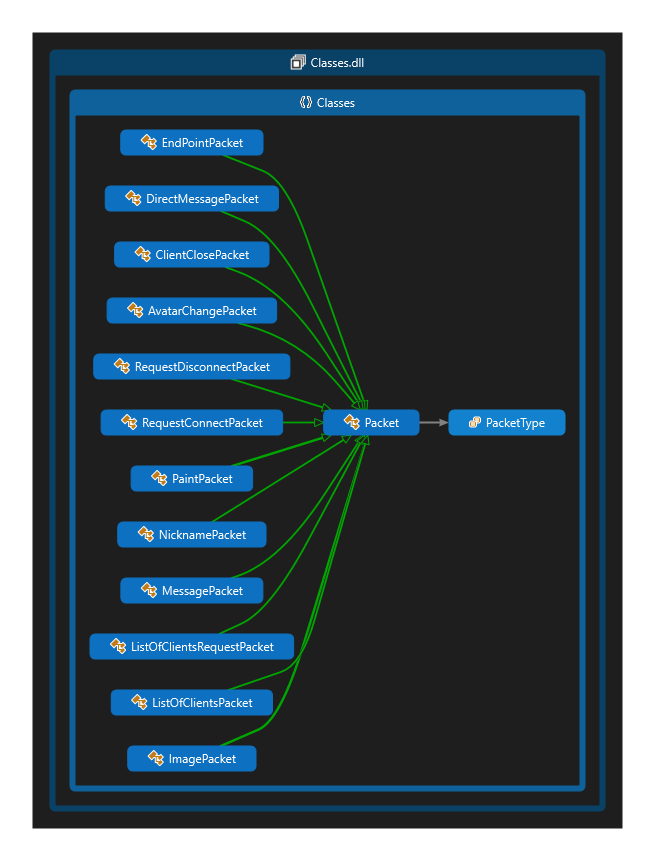
)Server UML

)Client Diagram



)Concurrency diagram

e)Server Diagram



f)Classes Library