

**CSCU9V5 – Concurrent & Distributed Systems**

Student ID: 2520796

The network interface provides a channel for interaction between the nodes and the database. There is a connection to the same port on the network for multiple clients (nodes), but only one can connect at a time. The server socket is waiting for requests to enter the network, processing the request and granting access by establishing a connection at a time with one of them, performing the relevant actions and then “returning the connection” to the requesting node by closing it.

For the purposes of the underline project a socket-based unified DME must be built in order to pass a unique token from the coordinator to every single node. When the specific token is issued, the essential segment can be performed by a node and then the token will return to the coordinator. Each node will operate on a different host and network whilst the coordinator will run on the same host. This will be achieved by forwarding the protocol and port node to the coordinator (e.g. on port 7000).

In the next stage, the node waits until the coordinator produces the token and executes the critical session while prints significant messages marking the beginning and the end of the critical session. After the exit of the critical region the token returns to the coordinator (Coordinator listens to port 7001).

The C\_receiver class listens to the node requests and spans a thread (C\_connection\_r class) that receives the IP and Port number and stores it in buffer in the C\_buffer class.

The C\_mutex class acts like a semaphore, but, with less threads. It receives threads, takes in requests and puts them in a queue to be executed. Moreover, it assigns the token to the requesting node for each query through a quick connection to the specified Port number and waits for a synchronization to retrieve the token (on Port 7001).

The class diagram below illustrates the network and the connection between the classes:

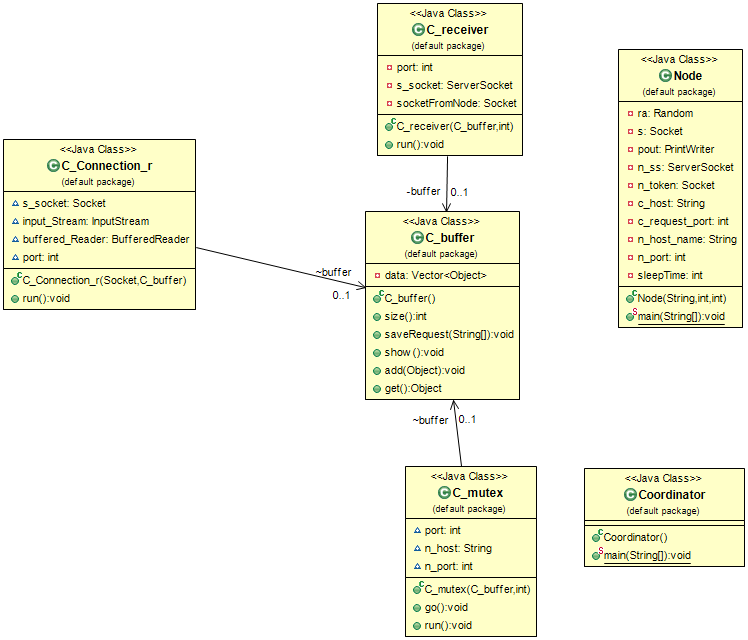


Figure 1: Project Class Diagram

Further to the above, the figure below illustrates the program running in real-time. On the left side, the Coordinator starts first acting as a server and uses the Ports: 7000 – 7001 to send and listen the requests from the Nodes, hence, those ports cannot be used on the Nodes.

The Coordinator starts using the command “java Coordinator” and waits for a request from a node. Then, in order to start a new node, another window shall be opened, and the following command shall be used: “java Node (port) (milliseconds)”. Moreover, any port can be used except 7000 or 7001, as the underlined project employed 4 nodes with ports 1000, 1001, 1002, 1003 respectively.

The Node requests a token from the coordinator to the port 7000 and then waits until the coordinator opens the connection and sent that token to the node. Once the token is been received from the node, then it enters and exits the critical session, while, it prints suitable messages during the process. Finally, the token is returned to the coordinator and the socket connection is finally closed.

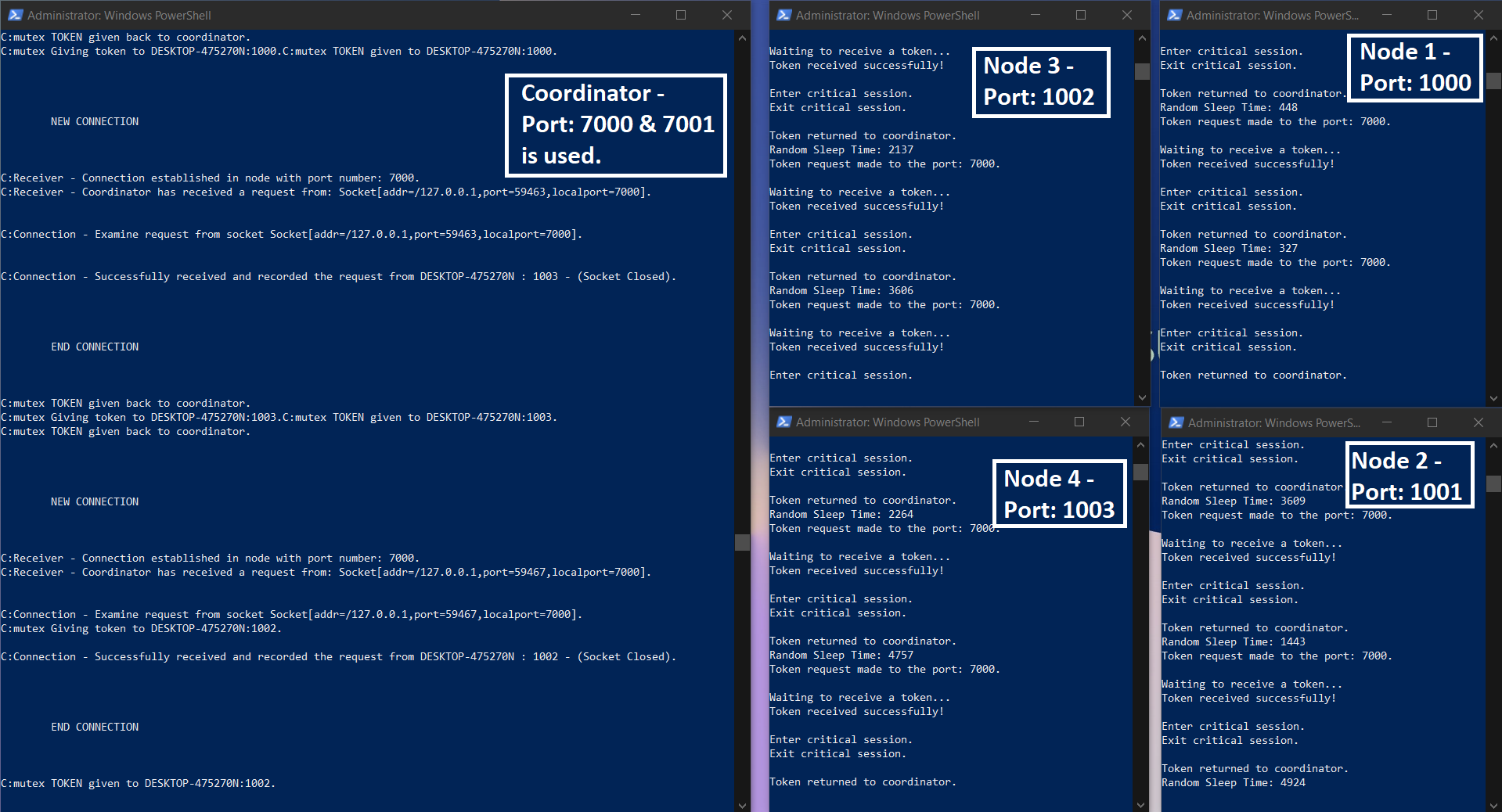


Figure 2: Project Real-Time Running screenshot