Algonquin College Logo

# SCHOOL OF ADVANCED TECHNOLOGY

### ICT - Applications & Programming

### Computer Engineering Technology – Computing Science



A31

Game C/S Model – Collaboration Diagram

Lab Professor / Lab Session:

[Daniel Cormier / 302]

Team:

[Vijayalakshmi Krishnaraj] - Id: [041026347] - [Priya Patel] - Id: [041084491]

TM Model C/S Proposal

***This template is suggested (not mandatory) to answer A31 Specification.***

|  |  |
| --- | --- |
| **Part**  **1** | **C/S Architecture** |

* 1. **UC Model (1pt)**

🛠 Define the UC diagram and the UC table (actors and functionalities).

**A screenshot of a diagram

Description automatically generated**

**Fig. 1** – C/S Model for Chat[[1]](#footnote-1)

**UC Diagram** (change this diagram to accommodate the actors and functionalities to be used):

|  |  |
| --- | --- |
| **Actors** |  |
| User | The user interacts with both the client and the server.  Initiates actions such as entering a username, server name, port number, connecting to the host, closing the connection, configuring the server, starting, and ending the server, finalizing the server, showing messages, entering the TM model, validating the TM model, sending the TM model, receiving game configuration, and running the TM. |
| Admin | Represents the system itself, which includes the server and client components.  Handles the internal processes such as configuring the port, starting, and ending the server, finalizing the server, showing messages, validating the TM model, and managing the game execution. |

**UC table** (example):

|  |  |
| --- | --- |
| **Use Cases** |  |
| Configure Port | User can define the port to be used for the server. |
| Start Server | The user uses the start button to **start** the server application |
| End Server | User can explicitly end the server. |
| Show Messages | User can view all messages (received from clients or logs) in a text area. |
| TM | User enters the TM model (string) for the game. |
| Validate Model | System validates the entered TM model. |
| Connection | User initiates an initial connection to the host using the provided hostname and port. |
| End | User closes the connection to the server. |
| User | User enters a username (string). |
| Server | User enters the server’s name (string). |
| Port | User enters the port number (integer). |
| Send | User sends the TM model to the server. |
| Receive | User receives game configuration from the server. |
| Run | User runs the TM in a separate frame/window. |

* 1. **ClassD (2pt)**

🛠 To draw the diagram, you can use tools (ex: <https://app.diagrams.net/>) or desktop applications (ex: Visio / Powerpoint) or simply take photos from drawings.

**Class Diagram** (change this diagram to accommodate the actors and functionalities to be used):

A diagram of a server

Description automatically generated

**Fig. 3** – ClassD for a chat[[2]](#footnote-2)

**Class table** (example):

|  |  |
| --- | --- |
| **Server Class** | **Order** |
| Inner Fields[[3]](#footnote-3) | * serverStatus: A string that indicates the current status of the server (e.g., "Running", "Stopped"). * connectedClients: A list that keeps track of all client objects that are currently connected to the server. * dataQueue: A queue that holds data objects which are waiting to be processed or sent to clients. |
| Relationships[[4]](#footnote-4) | * This is a bidirectional association, as indicated by the line connecting the two classes with the label "Ending Connection." It suggests that the Server class can communicate with multiple Client instances and may control the lifecycle of these connections, including their termination. |
| Methods | * startServer(): Initializes the server and prepares it to accept client connections. * stopServer(): Shuts down the server, disallowing any new connections and possibly closing existing ones. * broadcastData(data: Data): Sends a data object to all connected clients. * handleClientRequest(client: Client, request: Request): Processes a request from a connected client. |

|  |  |
| --- | --- |
| **Client Class** | **Order** |
| Inner Fields[[5]](#footnote-5) | * clientID: A unique identifier for the client, typically a string. * connectionStatus: A boolean that represents whether the client is currently connected to the server. * receivedData: Stores data that the client has received from the server. |
| Relationships[[6]](#footnote-6) | * The Client class has a receivedData field of type Data. This implies an association where the Client is the receiver of Data objects, possibly from the Server |
| Methods | * connectToServer(server: Server): Establishes a connection with a server. * sendData(data: Data): Sends data to the server. * receiveData(): Retrieves data sent from the server. * disconnect(): Ends the connection with the server. |

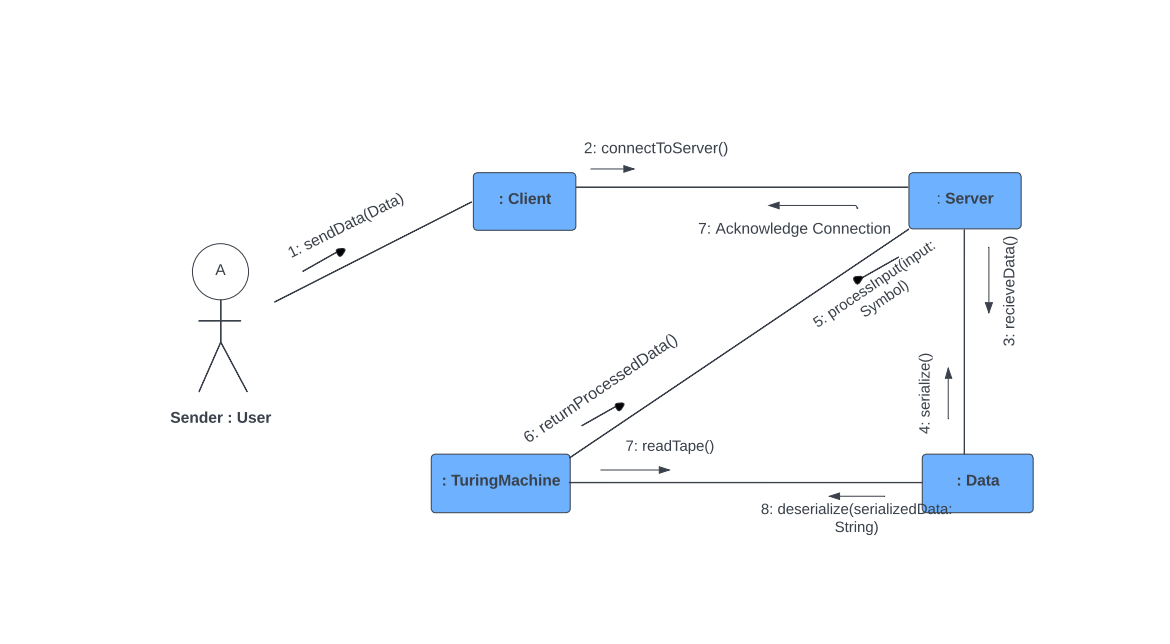
|  |  |
| --- | --- |
| **Data Class** | **Order** |
| Inner Fields[[7]](#footnote-7) | * dataContent: A string holding the content of the data object. * dataType: A string that indicates the type of data contained (e.g., "text", "image"). * timestamp: A DateTime object representing when the data was created or last modified |
| Relationships[[8]](#footnote-8) | * The Server has a dataQueue which is a queue of Data objects. This indicates an association relationship where the Server class uses the Data class. The Server likely enqueues and dequees Data objects as part of its operation, particularly when broadcasting data to clients. |
| Methods | * serialize(): Converts the data object into a string format for storage or transmission. * deserialize(serializedData: String): Converts a string back into a data object. |

|  |  |
| --- | --- |
| **TuringMachine Class** | **Order** |
| Inner Fields[[9]](#footnote-9) | * currentState: Represents the current state of the Turing machine, which determines its operations. * tape: A list representing the tape of the Turing machine, where each element is a symbol. * transitionTable: A mapping that defines the state transitions based on the current state and input symbol. |
| Relationships[[10]](#footnote-10) | * The UML diagram doesn't show a direct relationship line between TuringMachine and Data. However, if we consider that a TuringMachine could process Data or use it as part of its computation, we might infer an indirect association where Data could be the input, output, or state information for the TuringMachine, even though this isn't explicitly shown in the diagram. |
| Methods | * processInput(input: Symbol): Processes the given symbol according to the transition table and changes the machine's state. * moveTape(direction: Direction): Moves the tape in a specified direction (left or right). * readTape(): Reads the symbol at the current tape position. * writeTape(symbol: Symbol): Writes a symbol to the current tape position. |

*Create tables for all classes.*

* 1. **UML Collaboration Diagram**

*Describe the collaboration diagram to be used.*



**Fig. 4** – Collab model for a chat[[11]](#footnote-11)

**References**

Algonquin College

Fall, 2023

1. See <https://www.researchgate.net/figure/System-Architecture-Use-Case-Diagram-Client-side-Functionality_fig2_318502492>. [↑](#footnote-ref-1)
2. See <https://www.codeproject.com/Articles/443660/Building-a-basic-HTML-client-server-application>. [↑](#footnote-ref-2)
3. The inner fields and relationships together are properties from the class. [↑](#footnote-ref-3)
4. In the diagram, you can see relationships (ex: association / aggregation) between classes. In the implementation, these relations imply in the inclusion of other classes as fields. [↑](#footnote-ref-4)
5. The inner fields and relationships together are properties from the class. [↑](#footnote-ref-5)
6. In the diagram, you can see relationships (ex: association / aggregation) between classes. In the implementation, these relations imply in the inclusion of other classes as fields. [↑](#footnote-ref-6)
7. The inner fields and relationships together are properties from the class. [↑](#footnote-ref-7)
8. In the diagram, you can see relationships (ex: association / aggregation) between classes. In the implementation, these relations imply in the inclusion of other classes as fields. [↑](#footnote-ref-8)
9. The inner fields and relationships together are properties from the class. [↑](#footnote-ref-9)
10. In the diagram, you can see relationships (ex: association / aggregation) between classes. In the implementation, these relations imply in the inclusion of other classes as fields. [↑](#footnote-ref-10)
11. See <https://www.researchgate.net/figure/Collaboration-diagram_fig1_268362338>. [↑](#footnote-ref-11)