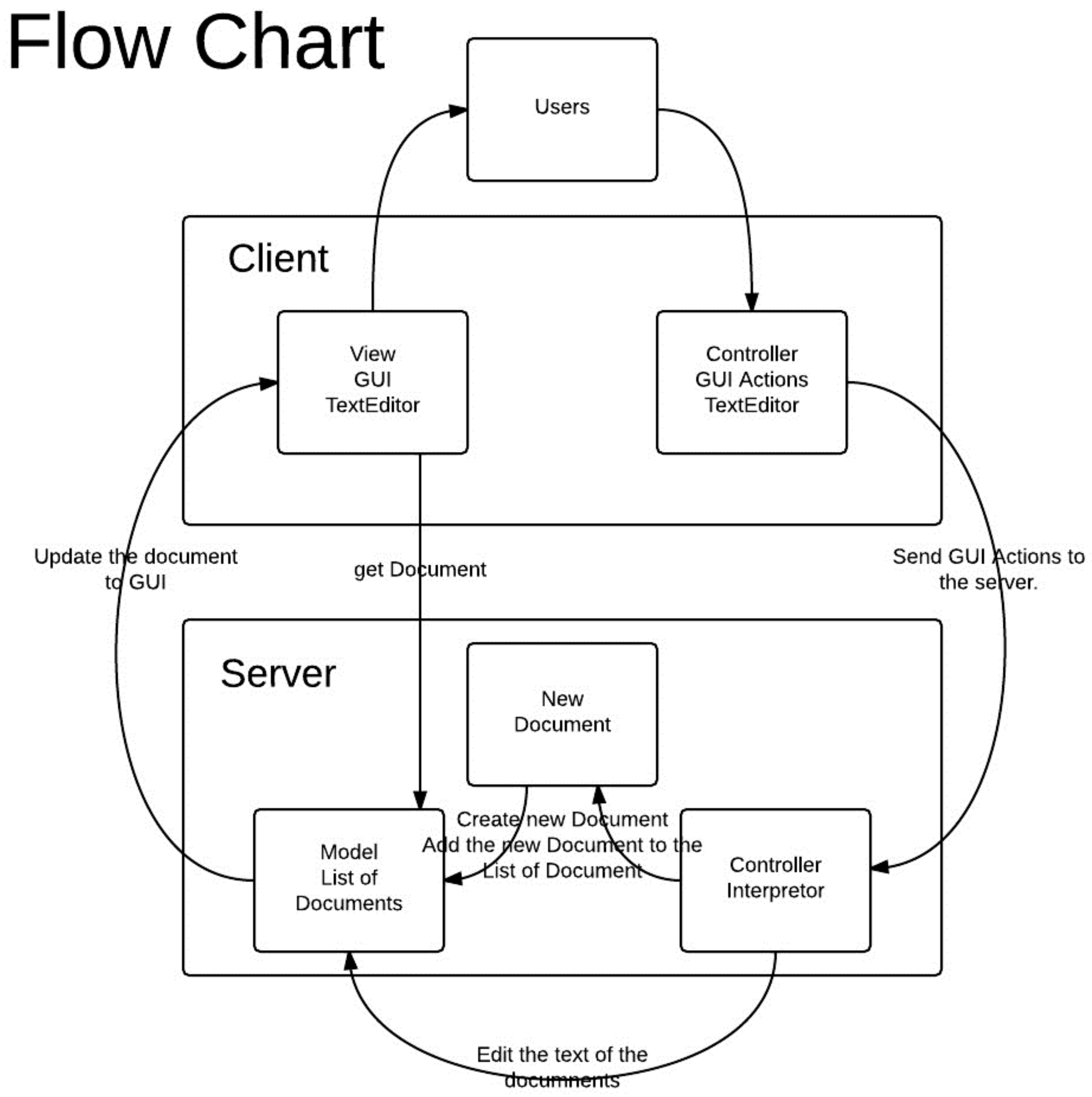
Project 2 Design Decisions

ericemer, salazarm, mpan1218

Client => View - Controller Pattern
Server => Model - Controller Pattern
Client - Server => Model - View - Controller Pattern



Client:

Fields:

ID - ID of document being edited

Methods:

We will have Swing listeners for different events such as highlight-paste, deleting entire sections, etc. They will appropriately handle how many INSERT and DELETE commands we need to send. There will be a main SEND command which sends text to the server via the output stream.

Design Choice:

1. All Changes are going to be handled by the central server directly. Even the person making the change will not see the update until the server has handled the edit and returns the updated version of the document (We believe the lag will be minimal and unnoticeable enough for this to work).. We withheld the design of storing document in all clients because for the initial design, we want to keep things simple and easy to implement. However, our choice of design might result in latency of updates if lots of edits are made in a short time.

Server:

Fields:

ArrayList<Document> Documents - List of documents on the server.

Map<ID, Queue> queueMap - Maps each document to a queue.

Methods:

String get(ID) - returns Documents.get(ID).toString() and calls Documents.addActiveUser(Socket).

void handleConnection(COMMAND) - Gets the ID from the COMMAND and adds it to the appropriate document's queue.

class Document:

Fields:

Queue<COMMAND> queue - A Queue of Commands to be handled.

ArrayList<char> doc - An ArrayList of characters representing the document.

Map<Socket, PrintWriter> activeUsers - A Map of users editing this document and their output streams.

Methods:

void addActiveUser(Socket) - Opens an output stream and adds the socket and the stream to the activeUsers map.

void insert(INDEX, CHAR) - Inserts CHAR at position INDEX of the doc ArrayList.

void remove(INDEX, CHAR) - Removes CHAR at position INDEX of the doc ArrayList.

void toString() - returns a string representation of the document.

void handleCommands() - This is always active in a background thread. It executes commands in the Queue and calls updateActiveUsers() after every command.

void updateActiveUsers() - Goes through all open sockets in the activeUsers Map and sends the result of document.toString() through the output stream.

GRAMMAR:

```
COMMAND  ::= NEW | INSERT | DELETE | GET
NEW ::= “NEW” NAME
NAME ::= [.]+
DELETE::= ID “DELETE” INDEX
INSERT ::= ID “INSERT” INDEX  CHAR
GET ::= “GET” ID
ID ::= [0-9]+
INDEX = [0-9]+
CHAR = [\\s\\S]
DOCUMENT ::= CHAR*
```

Assumptions:

1. All user edits will be something that can be broken down into inserts and deletes.
2. Server queue will be able to handle requests in a timely manner without debilitating visible lag.
3. A user who makes edits and then rapidly leaves the document will still have his or her requests posthumously handled by the server.
4. A single server/queue will be capable of handling an unlimited number of users.