1 Introduction

Surfin' Editor is a collaborative editor with a server that

- Stores files locally. (Hence files are never deleted on server shut-down)
- Supports multiple documents being edited at once.
- Supports multiple users editing the same document.
- Supports cut, copy, and paste into the document.

Local Storage

When a user creates a file the server automatically creates a local file to hold the user's document. This local file is stored inside of a local directory called the `documents` directory; the server will automatically create this directory if it does not already exist. The server stores the document titles and file paths in a file called `server.cfg`. This file uses the following grammar to store this information:
`TITLE FILEPATH`
This file is written to when new files are created and is used to rebuild the server every time the document starts up. Rebuilding the server means that we are going to create a `document` object for every document in the `server.cfg` file. During the building phase the server will not handle connections, thus if a very large number of files were on the server there would be some delay until the server can actually serve. This trade-off is unnoticeable for our purposes.

Multiple Documents

Each document has its own thread and queue for commands to be executed on the document. This means that instead of commands being executed in the order that they arrive to the server, they instead are executed in such a way that because each document handles its own commands, each document is executing roughly the same number of commands per unit time on average, meaning that if document A has 1000 commands sent to the server and document B has 200 commands sent to the server, and even if they arrive very close together with document A's commands arriving first, document B would be much more likely to have its 200 commands completed first.

Multiple Users

The server keeps 1 version of the document that is broadcasted to all users viewing/editing the document whenever a user edits something. This is ensures that all users are updated. Because of the nature of this, our editor relies on low latency connections to behave well.

Cut, Copy, and Paste

To allow for cut, copy, and paste we used KeyListeners to break up cut and paste actions into separate individual delete and insert commands. The Cut action sends delete commands for each character in the highlighted text. It is triggered by either `Ctrl+x` or the Cut button in the GUI. The Copy action works as a normal copy action, it needs not send any commands to the server. The Paste action sends insert commands for each character in the sender's clipboard. It is triggered by either `Ctrl+v` or the Paste button in the GUI.

2 Design Decisions

2.1 Abstract Design

We chose to use a Model-View Controller Design Pattern. In our implementation, we defined the following relationships:

```
Client ::= View - Controller Pattern
Server ::= Model - Controller Pattern
Client - Server ::= Model - View - Controller Pattern
```

Figure 1 represents our implementation of this architecture to Surfin' Editor. In this design, there is a central server that stores all the documents for the users and is connected to by the clients. The server will function as a Model - Controller Pattern. The Controller in the server will edit the design according to the action made by the server on the Client.

When a Client sends an action to the Server, the central server will handle the action and make an update to all the Clients are the same time. When multiple actions are received from different Clients, the Server will handle action by action and update to Client whenever it finishes handling an action.

Surfin' Editor is a realtime collaborative editor. This means that it supports multiple clients interacting with the same document. In order to create this sense of collaboration, Surfin' Editor users interact with the client. The client sends actions to the server, and the server parses these actions and sends updates to the other clients. Figure 2 expresses a potential chain of events that can be carried out using Surfin' Editor.

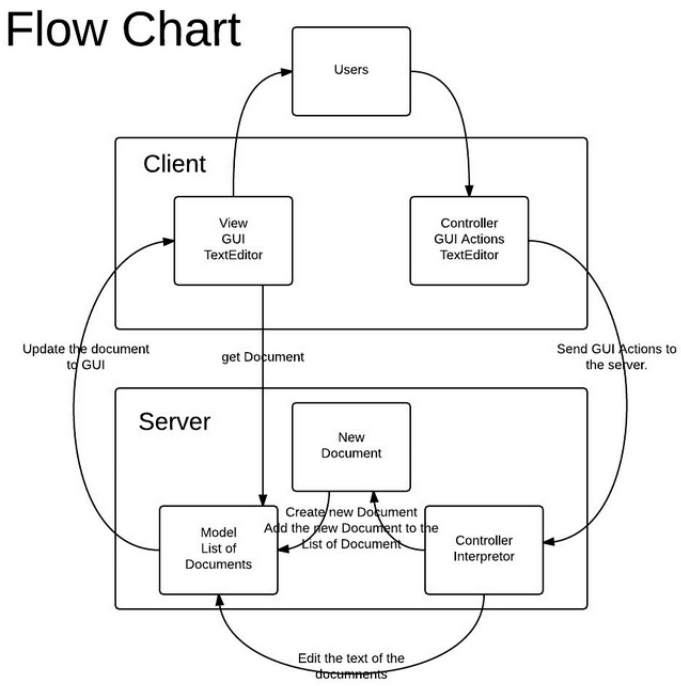

Figure 1: Flow chart representing our implementation of the MVC Design Pattern architecture.

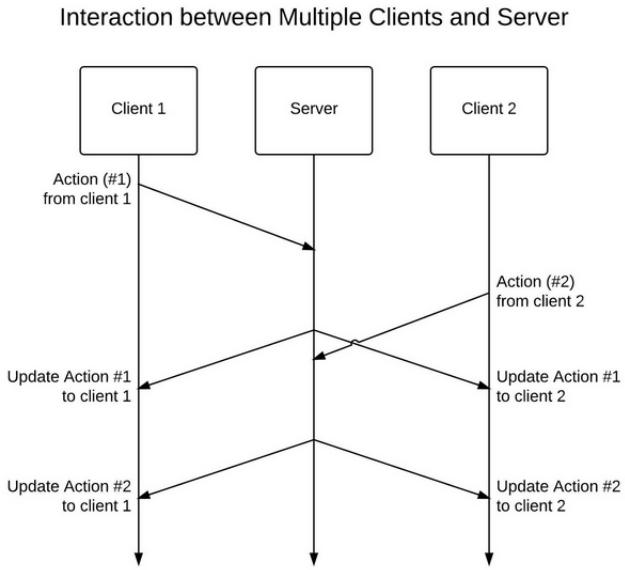

Figure 2: A possible interaction between multiple clients and the server. The server reacts to actions from the client and passes along updates to other clients.

2.2 Real Implementation Decisions

1 We decided that if one user has text highlighted and another user starts editing within that text, that the first user will highlight the text that is already highlighted and also the text that the other user adds.

2 We decided to use a KeyListener and CaretListener to determine the characters that are typed by the user and to track each users caret position.

3 We decided to make the cut, copy, and paste keyboard shortcuts incompatible with Mac OS. Mac users must use the cut, copy, and paste buttons in the JToolbar in order to perform these clipboard actions.

4 Only 2000^2 documents can have the same title.

5 Documents cannot be deleted.

3 Grammar

3.1 Client to Server

```
COMMAND ::= NEW | INSERT | DELETE | GET
NEW ::= "NEW" NAME
NAME ::= [.]+
DELETE ::= "DELETE" ID INDEX
INSERT ::= "INSERT" ID INDEX ASCIICODE
GET ::= "GET" ID
ID ::= d+
INDEX ::= d+
ASCIICODE ::= d+
CONNECT ::= "CONNECT"
```

3.2 Server to Client

```
MESSAGE ::= (ID TITLE)* ID ::=
d+ TITLE ::= [.]+ FILE ::= ID | "A" | DOCUMENT
ID ::= d+
DOCUMENT ::= "A" | ASCIICODE
ASCIICODE ::= d+
```

Figure 3: `ClientLoader`, which is the user interface users see when they launch SURFIN' EDITOR.

Figure 4: `ServerDocumentListLoader`, which is the user interface users see after they have connected to a host and port using SURFIN' EDITOR.

4 GUI

The SURFIN' EDITOR GUI consists of three user interfaces. `ClientLoader` is an interface where the users can connect and manually input a host and port. This user interface is a JFrame window with JPanels where users can enter the host and port to wish they would like to connect.

The second interface is the `ServerDocumentListLoader` where the users are allowed to open an existing document or create a new document. This user interface is a JFrame window which displays the available documents, or allows users to create a new document.

The third interface is the `TextEditor` where the users can edit the file collaboratively. This user interface contains a JTextArea where users can write and edit text. It also has a JMenuBar with File and Edit menus. SURFIN' EDITOR includes a JToolBar which includes buttons for Open, Cut, Copy, and Paste actions.

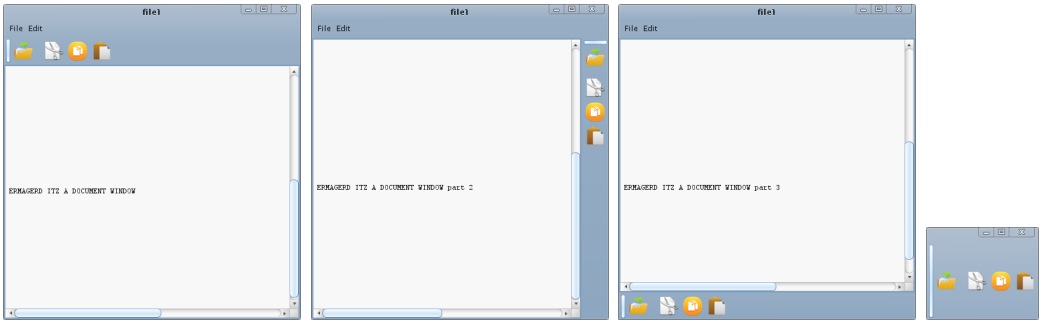

Figure 5: `TextEditor`, which is the user interface users see when they are ready to edit their document either collaboratively or alone. The JToolBar can be popped out of the window or placed on any edge of the main window. It can also be popped back in.

5 Assumptions

- All user edits will be something that can be broken down into inserts and deletes.
- Server queue will be able to handle requests in a timely manner without debilitating visible lag.
- A user who makes edits and then rapidly leaves the document will still have his or her requests posthumously handled by the server.
- A single server/queue will be capable of handling an unlimited number of users.
- Race condition: Offline editing would cause problems of two people editing the same thing and trying to merge the two documents.
- Client can receive ASCII code and translate it into string text.
- The user will not try to input anything that is neither an action key nor a character which can be converted to ASCII.

6 Revisions

1 We decided to use ASCII code to communicate between the server and client. We did this in order to make it easier for the server to comprehend line breaks. Because of this decision, we did not have to make special cases for line breaks.

2 We decided to change the grammar so that a document ID is sent to the client. This allows the client to have multiple documents open at once, and for the server to send the document to the correct `TextEditor` instance.