Final Design

6.005 Project 2 - Collaborative Editor

Deepak Narayanan, Todd Cramer, Victor Pontis

TA - William Ung

Important Questions/Overview

- **What is an edit?**
  An edit is a character removal or insertion. We do **NOT** support multiple character insertions or deletions. This includes **NO** copy/paste support.
- **What set of editing actions are provided?**
  Users are allowed to input and delete characters. Users cannot remove or add multiple characters at a time such as with highlighting then copy and pasting.
  No changes involving multiple characters are reflected on the server; the client is resynced with the server in such a case; in order to prevent future errors.
- **How the document is structured?**
  A document is stored internally as a string of text. It also has some other meta data such name, last edit time and version number for bookkeeping purposes.
  A document also has a list of changes (each version number corresponds to a change) which establish how a document was constructed, and is useful to merge client changes on the server.
- **How documents are named and accessed by users?**
  Documents are named by the user. Every time, a user logs in, he sees a document table which contains all available documents. The user can choose any document within this table to edit. A user assigns a unique name to a new document. Documents CANNOT be renamed or deleted.
- **Where documents are stored?**
  The documents are stored on the central server. This makes it easier to facilitate collaboration. Local copies are opened on the clients, when a client opens a document.
- **What guarantees are made about the effects of concurrent edits**
  All edits go through and modify the document. We cannot guarantee that an edit will go through before other people editing the document. If multiple people are editing the document and insert characters before the position you want to delete, you may end up deleting one of their inserted characters and not the character you wanted to delete. But if you are the only one editing the document or if other users aren't editing your same section, your edits will go through and properly modify the document.
  We will use a queue on the server to handle multiple changes. When one of the clients changes something they add a change request which is added to the end of the queue.

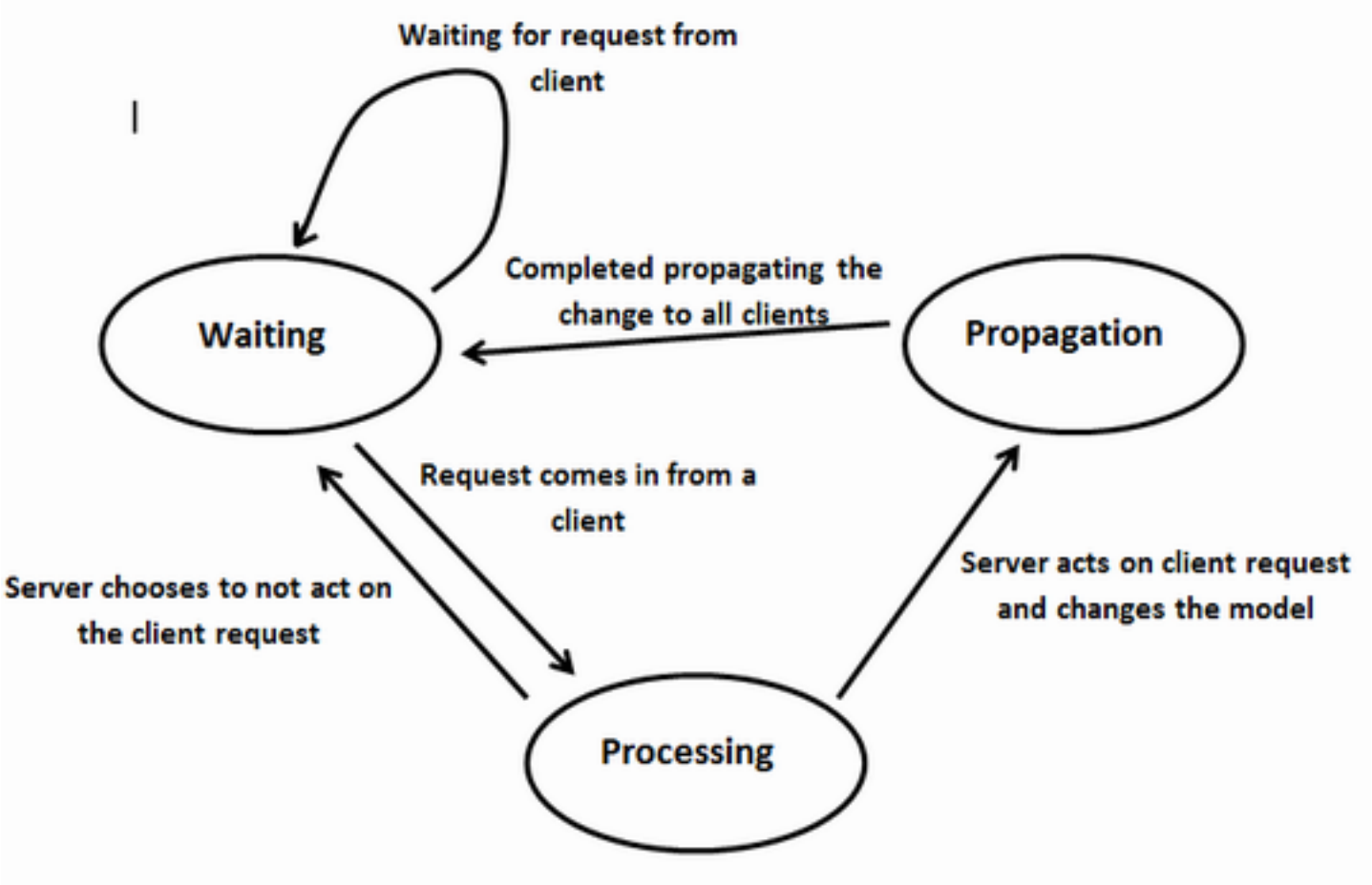
Architecture

We used a server - client architecture. Each client sends requests to the server. The server processes these requests; and then sends messages back to the clients. After receiving messages from the server, the clients update themselves. The messages between the server and the controller are translated into objects on both sides. These objects contain maps of all of the key-value pairs in the requests. Both the client and the server have queues which hold their requests in chronological order of time received.

States used on the server side

Our server has a queue and a series of states. The states are waiting, processing, and propagating. In the waiting state, the server is waiting for a new message from the client. In this state, the server is static. It continuously reads the queue until something is added to the queue. When the server finds something in the queue, it enters the processing state. It looks at the client request and decides what to do. It could do two things, either act on the request and modify the model or not act on the request and not modify the model.

If the mutation occurs, the server performs the mutation. Then if the client's request change's the model, the server switches to the propagating state and broadcasts the change to all affected clients. It then returns to the waiting state, and waits for the next request from a client.



State diagram of the server

Client-server protocol

We set up our protocol similar to HTTP get requests. Each of our client to server and server to client requests basically encoded a dictionary and the name of that dictionary. Each request started with the name/type of the request followed by an ampersand. We had different key fields which were followed by an equals sign, the value of the field, then an ampersand. For example an example request from the client to the server could look like this, “opendoc&docName=document1&userName=VictorPontis&id=21&”. When the client or server received this message they converted it into an object, a ControllerRequest or ServerRequest respectively, and parsed the request with regexes. The advantage of this was that we did not have to hardcode the fields, could easily add more fields and we did not have to worry about the order of the fields. The ControllerRequest and ServerRequest objects used regexes to convert the url into a dictionary. We escape the ampersands and equal signs in the text to prevent the user from injecting code, accidentally or maliciously, into our protocol.

When we outline our protocol we will give the name of the request and then the arguments that it requires. We will not show how each request is exactly formatted (because they are all

formatted exactly the same) and it does not matter what order we give the arguments. Both of these are argued in the paragraph above.

Shared grammar

NEWLINE ::= “\n”
TAB ::= “\t”
COLON ::= “:”
COMMA ::= “,”
INTEGER ::= [0-9]

Username ::= [A-Za-z]+
Docname ::= [A-Za-z]+
Position ::= INTEGER+
Length ::= INTEGER+
Version ::= INTEGER+

Hour ::= INTEGER INTEGER //Hour must be between 1 and 12
Minute ::= INTEGER INTEGER //Minute must be between 00 and 59
AM_PM ::= “AM” | “PM”
CurrentMonth ::= INTEGER INTEGER //CurrentMonth must be between 1 and 12
CurrentDay ::= INTEGER INTEGER //CurrentDay must be between 1 and 31

Client to server protocol

MESSAGE ::= (LOGIN | LOGOUT | NEWDOC | OPENDOC | EXITDOC | CHANGE | CORRECTERROR) NEWLINE
//logging in and out
LOGIN ::= “LOGIN” Username
LOGOUT ::= “LOGOUT” Username

//accessing different documents
NEWDOC ::= “NEWDOC” Username Docname
OPENDOC ::= “OPENDOC” Username Docname
EXITDOC ::= “EXITDOC” Username Docname

//make change
CHANGE ::= (INSERTION_CHANGE | DELETION_CHANGE)
INSERTION_CHANGE ::= “CHANGE” InsertionType Username Docname Position NewText Length Version
DELETION_CHANGE ::= “CHANGE” DeletionType Username Docname Position Length Version
CORRECTERROR ::= “CORRECTERROR” Username Docname

InsertionType ::= “insertion”

DeletionType ::= “deletion”

LOGIN - mutates the onlineUsers field within the server. User with same username cannot login to the editor. The server will either reply with IncorrectLogin if a user with the same name is already on the server or DocTablePage. If the user gets rejected, they stay on the login page and a little message pops up asking them to enter another user name. If they get accepted, they get forwarded to the document table page where they can create and edit documents or logout.

LOGOUT - mutates the onlineUsers field within the server. The server replied with a Logout message. This query attempts to logout whichever user the message specifies. This requires that the user is on the document table page. A user cannot logout from any other page.

NEWDOC - mutates the server. The server creates a new document and adds it to the list of documents saved on the server. The user can open this document and make changes to it. The new document must have a unique name, otherwise a new document is not created, and a notcreateddocument message is sent back to the user.

OPENDOC - does not mutate the server. The server sends information about the document with the specified document name to the client. The user can open this document and make changes to it. Note that only a document that has already been created can be opened.

EXITDOC - does not mutate the server. The document editor is closed and the DocTablePage is displayed.

CHANGE - may mutate the server. The client adds a request to the server queue.
The change may be either an insertion change or a deletion change.
Insertion Change - An insertion change request consists of the name of the document to which the insertion must be made; the position in the document at which the change is to be made and the length of the change. The change request also has a version number to keep track of the version of document the client made a change to.
Deletion Change - A deletion change request consists of the name of the document to which the deletion must be made; the position at which the deletion must be made and the length of text to be deleted.Delete change requests also contain version numbers to keep track of the version of the document the client made a change to.

CORRECTERROR- does not mutate the server. This request is sent by the client to the sever when the client is out of sync with the server. The server responds to this request by resetting text in the client to the text stored in the document on the server.

Server to client protocol

MESSAGE: ::= (CREATED | NOTCREATED | OPENED | LOGGEDOUT | EXITEDDOC | CORRECTED | CHANGED | UPDATE | NOTLOGGEDIN | LOGGEDIN | ID | ENDDOCINFO | DOCINFO) NEWLINE

CREATED ::= “created” userName docName date
NOTCREATED ::= “notcreatedduplicate”
OPENED ::= “opened” userName docName docContent collaborators version colors
LOGGEDOUT ::= “loggedout” userName
EXITEDDOC ::= “exiteddoc” userName docName
CORRECTED ::= “corrected” userName docName docContent
CHANGED ::= “changed” changeType userName docName change? position length version color
UPDATE ::= “update” docName collaborators colors
NOTLOGGEDIN ::= “notloggedin”
LOGGEDIN ::= “loggedin” userName id
ID ::= “id” id
ENDDOCINFO ::= “enddocinfo”
DOCINFO ::= (“docinfo” docName date collaborators)+ ENDDOCINFO

NOTLOGGEDIN - This gets sent back to the user when they enter in an invalid username or try a username that is already in the server.

LOGGEDIN- Sent back to the clients when the user with the given username is successfully loged in to the system.

CREATED- Sent back to the clients when a document with given document name is successfully created on the server.

NOTCREATED- Sent back to the clients when a document with given document name has already been created on the server.

OPENED- Sent back to the clients when a document is successfully opened by the user with the given username. Contains the content of the document stored on the server,and also all previous collaborators of the document, the new version number of the document, and the colors associated with the legends of each of the collaborators of the document.

LOGGEDOUT - This message confirms that the user has been logged off the server. To access documents, the user needs to log back in.

EXITTEDDOC- This message confirms that the document in the user with given username has successfully been exitted, and that the user should now see the document table.

CORRECTED- This message is sent to the clients when the server receives a CORRECTERROR message from the client. Contains information about the username of the user whose document needs to be corrected. Also contains the server’s version of the content of the document with given document name whose content needs to be updated.

UPDATE- Message sent to all clients when a new user joins a document, to update the collaborators on the documents of all other clients.

ID- Handshake message sent to a client when it connects for the first time to the server.

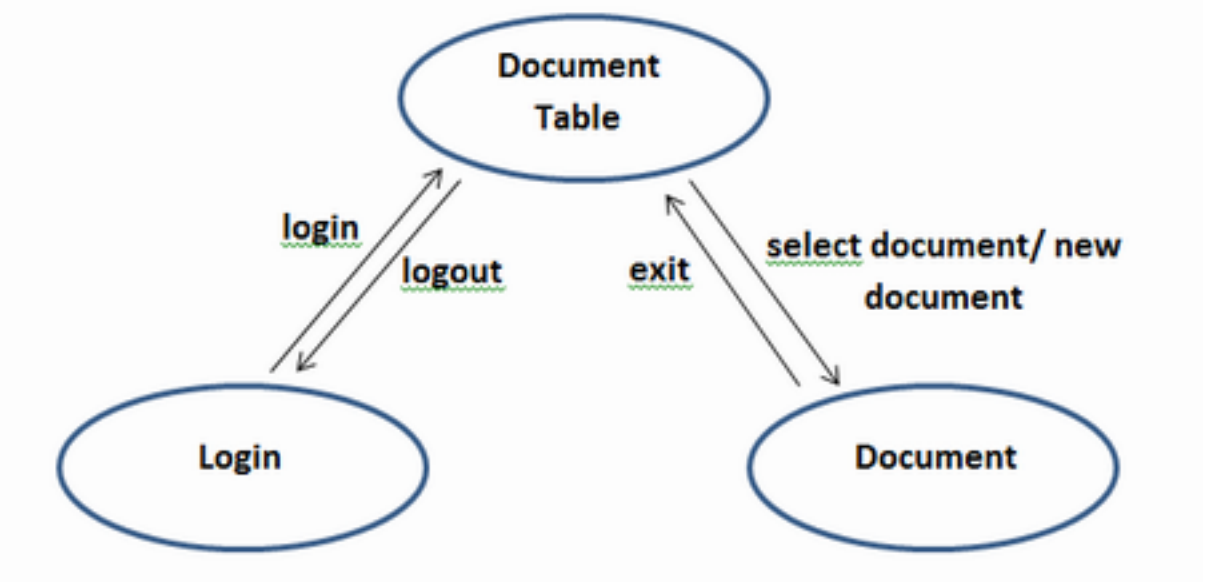
CHANGED- Message sent to all clients to propagate change made to the document on the server by a client. Contains the username who made the change, the document name of the document on which the change has to be made, the change, its length and position, and the new version number of the document.

DOCINFO- Message sent to the client, containing information contained in the document table. Contains information about every document on the server, the list of users who have opened the document, and the date/time at which the last change was made.

Classes Overview

GUI Classes

- **Login Page -** initial screen that asks a user for their name. The user logs in and gets redirected to the document table page.
- **Document Table -** list of documents and details about the documents, this is the page where the user can choose to access a document or create a new document. This is the splash screen that shows up after login. The user is able to logout which redirects them to the login page.
- **Document Editor -** page to edit a selected or new document. It has a title bar at the top. Under the title bar is another bar which displays the other users currently editing the same file and a button that allows the user to exit and go back to the document table page.



State diagram of the UI

Server Classes

- Server - this class sets up and gets the server running. It allows multiple users to connect to the server and modify the server's information. Each client connects to the server through a new socket, i.e., each client has its own socket. Each client and the server runs on a separate thread.

Model Classes

- Document - this class represents a document on the server. A document has a unique ID, a unique name, a list of collaborators and the last time the document was edited. The document represents the text by holding a list of paragraphs that correspond to this document. The document name and ID are at this point unique and final but we are looking into the option of allowing a user to change the name of a document.

Operational transforms

We use operational transforms to merge updates from multiple clients at the server. Each change is associated with a version number. When a change request is received by the server, the server looks at the version number of the incoming change. The server looks at all changes made since that version number, and updates the position of the change to be made in the document.

LaTeX

Our LaTeX does not support wrapping. It also does not support all functionality offered by LaTeX. At the same time, it does offer most of the core functionality. All LaTeX "quantities" are delimited by double dollar ($$) signs.

Decision Making Process

Ideas we rejected

During the initial brainstorming sessions, we discussed many ways to structure our implementation. Here are a few ideas we discussed but rejected in our final design:

1. We initially thought about having 4 GUI classes instead of 3. The forth one would be for adding a new document versus editing a document that already exists. We then realized it would be much easier and user friendly, to just have a single document table GUI that could handle both editing and document, or creating a new one.

2. We also considered different options for when a user tries to connect to the server with the same username as someone already on the server. We considered having both the users able to connect to the server, or automatically logging the first person off and allowing the second to connect. We finally decided on rejecting the second user's attempt to login. This makes the whole process more safe and more bug-free. Also when thinking if our app was used in the real world, it would not make sense to have two people connected under the same username, or for

someone to be logged out of the application in the middle of editing a document.

3. We considered having users “own” certain documents, allowing them to have more control over the document then just random users who access the file. We decided against this because we feel it is not necessary to have a good collaborative editor.

4. We also considered having clients store the documents. We felt it would be better to have them stored on the server, so that collaboration is easier to handle.

5. For editing, we originally thought about having the entire page update for every edit. We realized this approach would involve too many overheads, and was not necessary. If we implemented it like this the speed of updates would be much slower than necessary. To solve this problem we implemented edit to only update the paragraphs that were changed speeding up the saving process, and allowing users to see changes much quicker, and at the same time enhance parallelism.

6. You can also compare our initial design and our final design. We went through a decent number of refinements to reach the current design.