Initial Design

6.005 Project 2 - Collaborative Editor

Deepak Narayanan, Todd Cramer, Victor Pontis

TA - William Ung

Important Questions/Overview

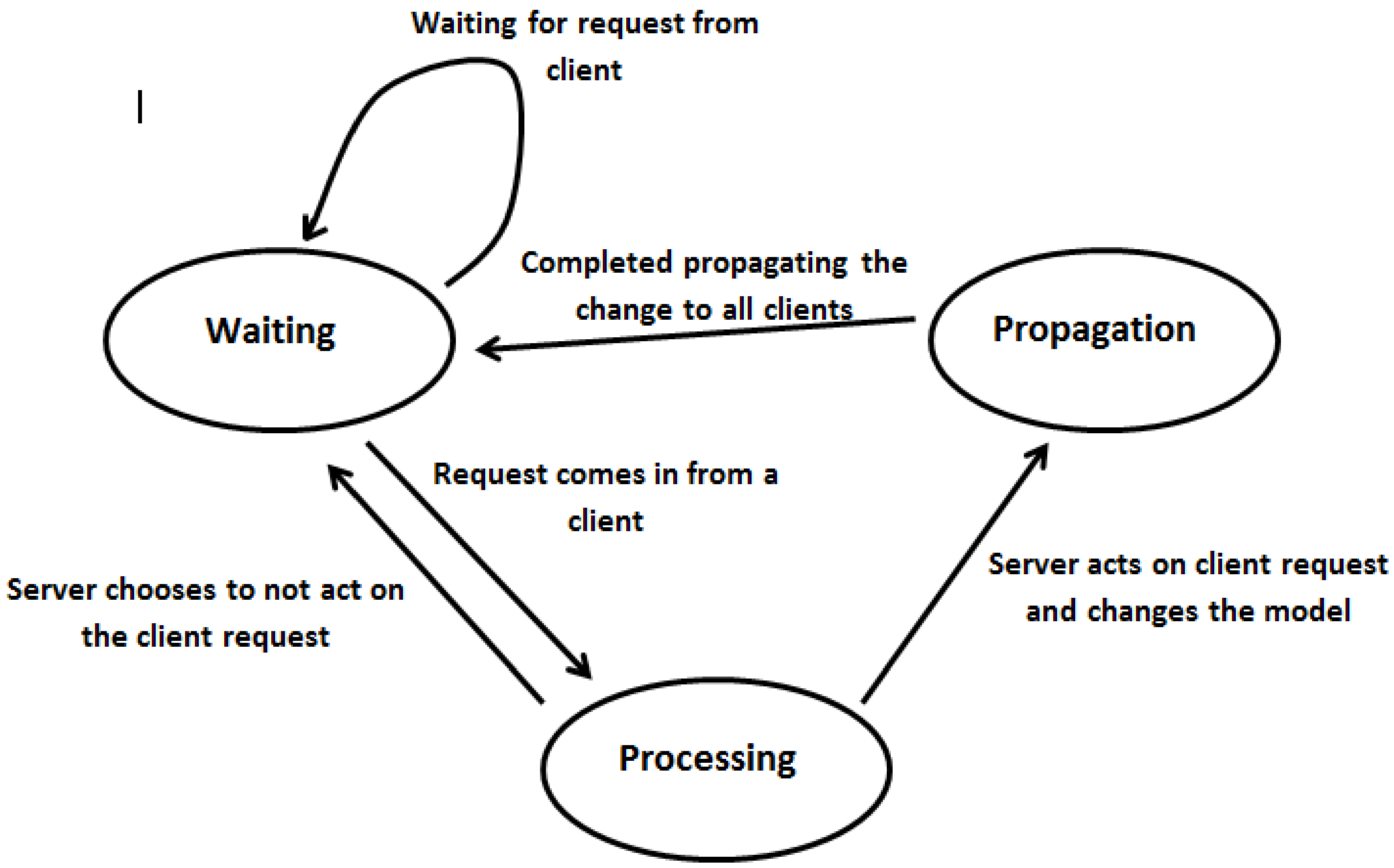
- **What is an edit?**
  An edit is a character removal or insertion. Right now, we do not support multiple character deletions. This may be changed in the future.
- **What set of editing actions are provided?**
  Users are allowed to input and delete characters. Users cannot remove or add multiple characters at a time such as with highlighting then copy and pasting. Also, the pound character is a special symbol and users cannot enter in the pound symbol.
- **How the document is structured?**
  A document is a list of paragraphs. It also has some other fields such as ID, name and version number for bookkeeping purposes.
- **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.
- **Where documents are stored (e.g. at a central server or on clients);**
  The documents are stored on the central server. This makes it easier to facilitate collaboration.
- **What guarantees are made about the effects of concurrent edits**
  - When a user is the only one editing a paragraph, all of their changes will persist.
  - When two users are editing the same paragraph, their changes might not persist.
  - 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. We are considering using different queues for different documents / paragraphs, to maximize parallelism.

Architecture

We are planning to use 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.

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.


Waiting for request from client
Waiting
Completed propagating the change to all clients
Propagation
Request comes in from a client
Server chooses to not act on the client request
Server acts on client request and changes the model
Processing

State diagram of the server

Client-server protocol

Shared grammar

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

Username ::= [A-Za-z]+
Docname ::= [A-Za-z]+
Docversion ::= [0-9]+
ChangeID ::= INTEGER+
ParaID ::= INTEGER+
Paragraph ::= ParaID “#” [^\# ^NEWLINE]+ “#” //the #’s delimit the text, it is an illegal character in our documents

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) 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 ::= “change” Username Docname Paragraph Docversion ChangeID

LOGIN - does not mutate the server. 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 giving them their error message. If they get accepted, they get forwarded to the document table page where they can create and edit documents.

LOGOUT - does not mutate the server. The server replied with a Logout message. This query attempts to logout whatever user the message specifies. This requires that the user is on the document table 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.

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 server may mutate the paragraph text to contain new text. The document version number is updated to reflect the change. If the user exits the document and reopens it, he will see this change reflected in the document.
The server may also choose to ignore the change; if this change interferes with another change that was made concurrently. In this case, the document version number is not updated; and the user does not see his/her change reflected in the document. Note that document version numbers indicate an implicit queue representation, because changes are made to documents only in increasing order of document version numbers.

Server to client protocol

MESSAGE ::= INCORRECT_LOGIN | DOCTABLEPAGE | DOCUMENT | CHANGEACCEPTED | LOGOUT | CHANGEDECLINED NEWLINE

INCORRECT_LOGIN ::= “User already on server”

DOCTABLEPAGE ::= [DOCUMENTINFORMATION]*
DOCUMENTINFORMATION ::= Docname TAB TimeMod TAB COLLAB NEWLINE

COLLAB ::= ([Username] COMMA?)* //list of users that have edited that document

TIMEMOD ::= Hour COLON Minute AM_PM COMMA CurrentMonth SLASH CurrentDay

DOCUMENT ::= Docname COLLAB [Paragraph]*

LOGOUT ::= “Thank you for your time, we have logged you out.”

CHANGEACCEPTED ::= “change accepted” ChangeID Docname Docversion
CHANGEDECLINED ::= “change declined” ChangeID Docname Docversion

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

DOCTABLEPAGE - This is the information for the document table page. It contains a list of documents that is on the server with some corresponding information for the documents. The information is the last time the document was modified along with a list of users who have edited the document.

DOCUMENT - This is the information for a document. This is what the GUI needs to display a document. There are only three pieces of information for this: document name, a list of paragraphs, and a list of collaborators. The GUI displays the text by looking through the paragraphs and aggregating the text for each.

CHANGEACCEPTED - This message confirms that a certain change has been accepted. It also gives the user the new version of the document with the document name.

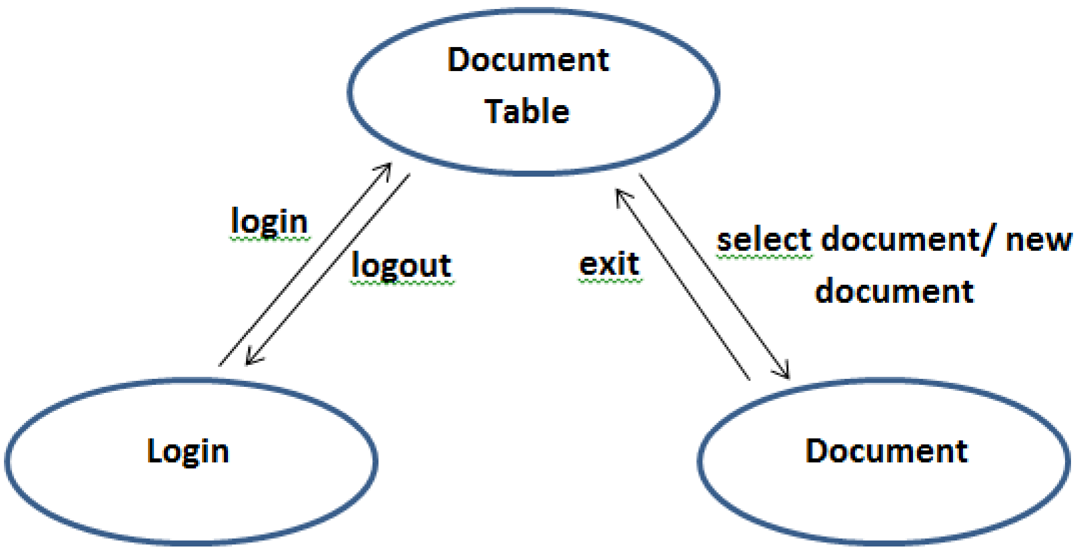
CHANGEDECLINED - This message confirms that a certain change was NOT accepted. The version of the document is not updated ,i.e., its value on the server is not changed.

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

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.
- Paragraph - this class represents a paragraph unit in a document. We define a paragraph to be a continuous stream of text ending with a carriage return (\n). Paragraphs have unique IDs and a string that represents the text they contain, not including their carriage return.
- User - this class represents a client that has accessed the server. The user has a unique name. At this point, any user can access the server by just typing in their name but if a user of the same name is already on the server, they get rejected. We are going to look into making a more fleshed out user system where users have passwords and we have a list of verified users.
- Document/Paragraph decision argument - We decided to have a document consist of a series of paragraphs because we reasoned that in the average use case people are not editing the same paragraph at the same time, usually people work on different parts of the documents. The breaking up of a document into different paragraphs allows us to only lock one paragraph at a time and allows for easier synchronization and

collaboration.

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.