6.005 Project 2: Collaborative Editor

Design Documentation

Yonglin Wu, Yunzhi Gao, Shu Zheng

Last Revised: 12/04/2012

Contents

I. General Design

1. Overview

A collaborative editor allows multiple users to work on a single document simultaneously, across a network.

2. Graphical Summary of Design

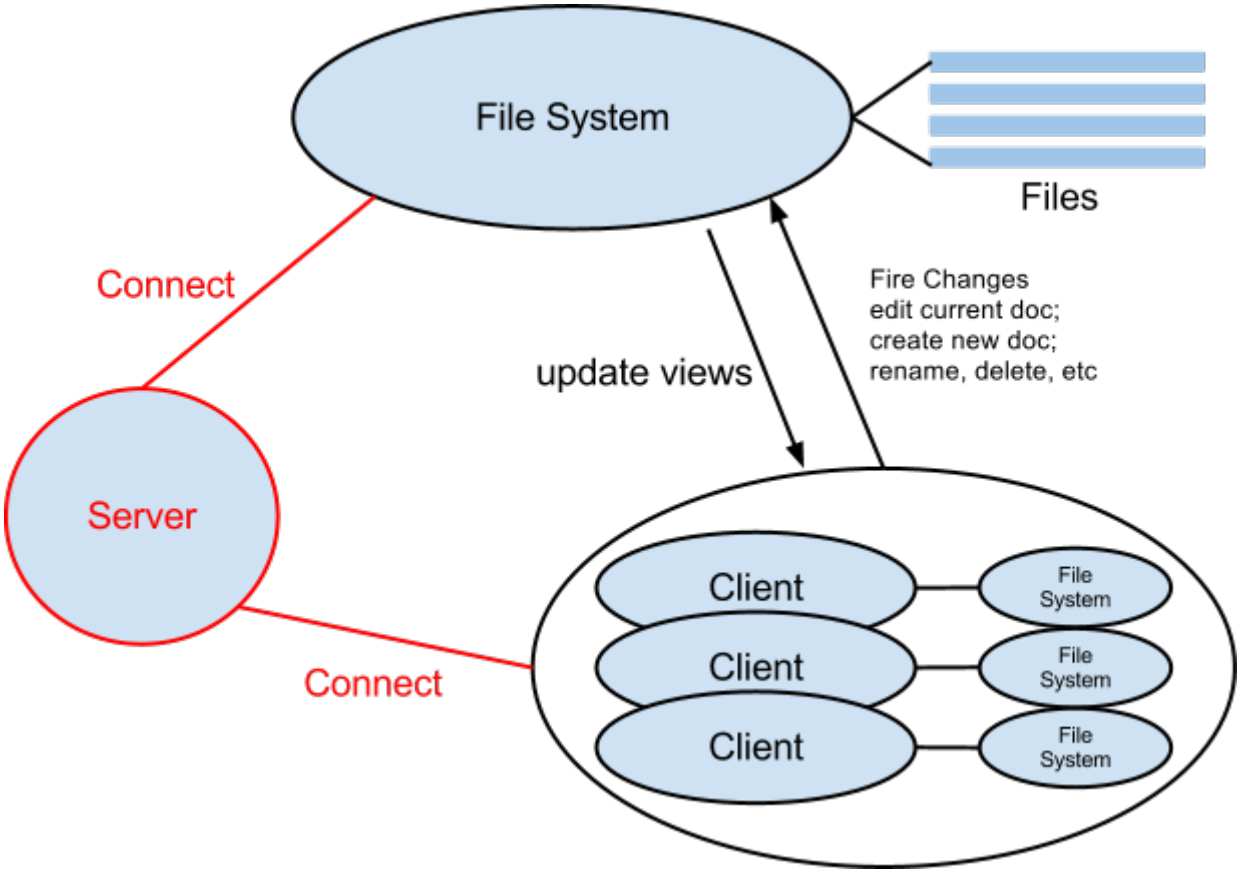

Figure 1 A

* To support multiple document, we design a File System to store all the documents created so far.

The central server and each client keeps a copy of the File System. When a client makes an edit, create a new file, or insert/delete file, the client will send a message (an EventPackage, a String, or a FilePackage) to the server. The server updates its File System accordingly. The server will then send the same event package back to all the connected clients, and update the state of their respective local File Systems. Finally, the change of local File System will be reflected on each of the local GUIs.

In this way, each client/editor will be able to see the instantaneous change made on the working document (as well as the File System) whenever any of the co-editors has modified a single document or the multi-document set, hence achieving the collaborative functionality of the program.

* A new GUI View is created for each new editor/client connected;

3. Datatype

The primary data that the users will access is the text in the textfield of the editor. We represent the text as an object of Java's default AbstractDocument. A document listener can be added to AbstractDocument to listen for edits on the document.

4. Multithreading & Concurrency

AbstractDocument supports multiple reader and one writer. When multiple users try to write on the document at the same time, relevant listener on the view will send an update message to the File System. The state of the File System (e.g., individual file name and content, ordering of files, insertion and deletion of files) will be instantly updated according to the message received from the client side. We shall use synchronized methods to preserve the atomicity of File System states and prevent interleaving.

5. Message & Protocol Descriptions

Our message will be in the format of the EventPackage, a built-in message format supported by Java. Specifically, we shall implement *java.awt.event* with the following three classes.

(1) EventPackage: contains editing information for single document.

Fields:
a) docNum: index/numbering of this particular document
b) eventType: record the editing type (insertion/deletion) of this event
c) inserted: the string inserted if the editing type is insertion.
d) offset: the position of this insertion
e) length: the length of this editing. How many characters are inserted/deleted

(2) FilePackage: contains information for a single file, used when adding files
Fields: a) file: contains filename information
b) content: contains the current content of this file

(3) FilenameChangePackage: contains information for filename change, used when change file names
Fields: a) docNum: index/numbering for this particular document
b) newFileName: new filename

As is in Problem Set 3, we shall again be using Telnet for interacting with our text-based protocols.

- On the client side, one of the client made some change to the documents, then fire an appropriate message to the server (on Telnet).
- Server receives the message and knows the client who fired this change. Then, it distributes the message to all other clients.
- All other clients receive the messages, and make corresponding changes to their documents.

II. Design Choices

* *what set of editing actions are provided?*
For an individual document: change caret position, cut, copy, paste, select text, select all;
For multiple documents: insert/delete/rename files, save file to local disk.

** how documents are named and accessed by users?*
When the document is first created, it will have a default name (e.g., "Default Document"). Any user can change that name in GUI at any time. Same document name is shared among users.

** where documents are stored (e.g. at a central server or on clients)?*
Documents are stored both in the File System of the central Server and in the File System of the clients.

** what guarantees are made about the effects of concurrent edits?*
When a user edits the document, the attached document listener will fire the change to the File System on the server. The changes will be ordered (e.g., via a queue) in the server, which then handles the changes one by one and return the latest state of the File System to each client. In this way, valid concurrent multiple edits would be guaranteed.