6.005 Project 2: Collaborative Editor

Design Documentation

Yonglin Wu, Yunzhi Gao, Shu Zheng

Last Revised: 11/27/2012

Contents

I. General Design

0. Overview

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

1. View-Model-Controller Design Pattern

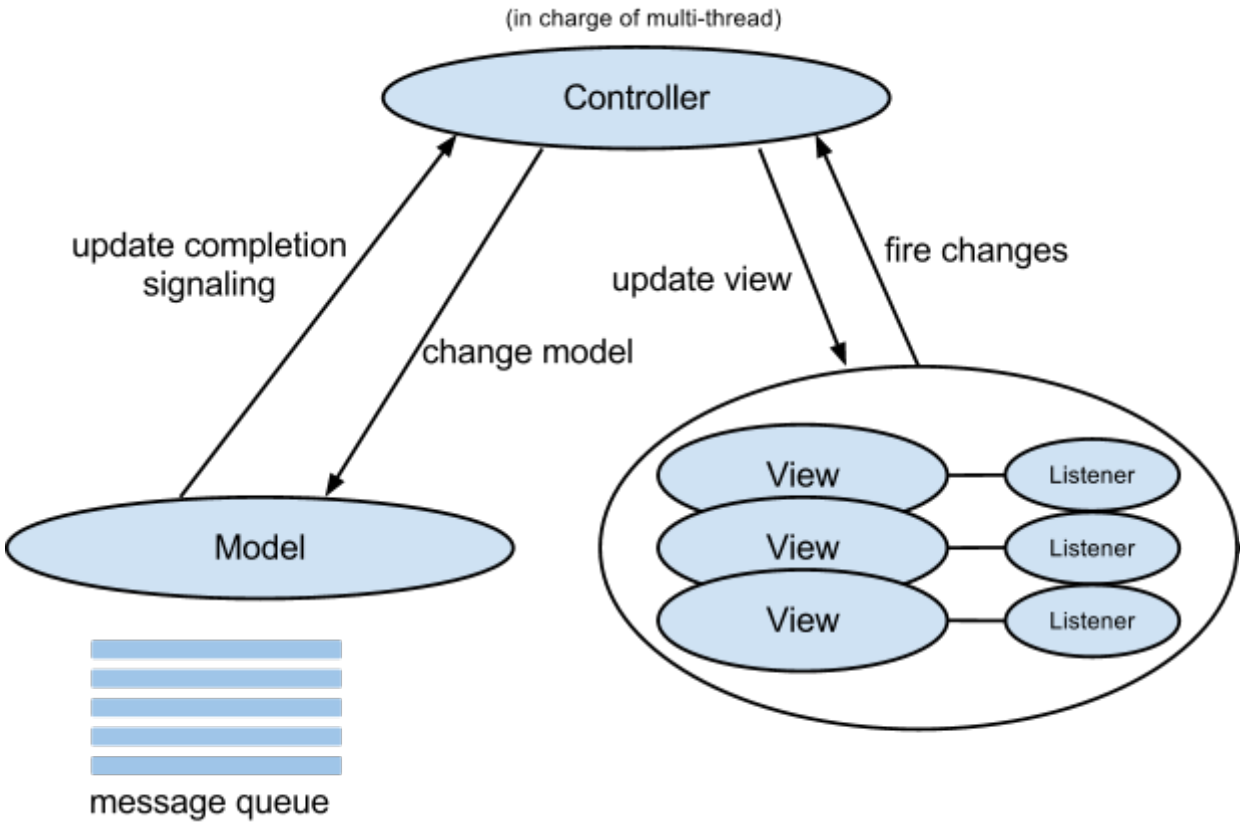

Figure 1 View-Model-Controller

* The file being edited (i.e., the *document*) is stored in Model
* A new GUI View is created for each new editor/client connected; one thread per client
* Controller is in charge of multithreading/concurrency
* The message passing between Controller and Model is handled in a queue

II. Design Choices

* *what set of editing actions are provided?*
Cut, copy, paste and select text.

** how documents are named and accessed by users?*
When the document is first created, it will have a default name ("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 on central Server. Specifically, it is stored inside the field "model" on the server.

** what guarantees are made about the effects of concurrent edits?*
First in- first out (FIFO)
Whenever a user edits the document, the listener on the GUI listens for the change and fires the change to the controller. The controller then send the change to the model. The change on stored on a queue in the model. The model handles the change one by one and returns the effect of the change back to controller, which then sends back to different GUIs.

II. Model

1. Datatype
The only data that the users will access is the text in the textfield of the editor. We store the data a field called doc inside the model, which will be represented as an object of Java's default AbstractDocument. A document listener can be added to AbstractDocument to listen for edits on the document.

II. Multithreading & Concurrency
AbstractDocument supports multiple reader and one writer. When multiple users try to write on the document at the same time, controller will send an action to model for each edit; model will

put them on a queue and do the edit one by one using the “first in, first out” principle. After each edit, model will return the current doc back to controller and the controller will update view for each GUI.

III.Controller

The Controller listens to and handles action events from the client, and update both the view (i.e., the front end) and the model (i.e., the back end) based on the action it receives.

We specify that each client has his/her own View. When an editing action has been performed on the GUI text field (by a certain client), a new thread will be created for both the back end and the front end. Controller will then compare the *document* in the Model with the contents in View’s text field, and update the Model accordingly if they no longer match. The updated *document* in Model will in turn be registered in Controller and get passed into all the Views on the client side.

In this way, each editor will be able to see the instantaneous change made on the working *document* whenever any of the co-editors has modified the text, hence achieving the collaborative functionality of the program.

IV. Graphical User Interface Design

2. Tentative Editor View and View Tree

The view facing the each client is shown in Figure 2.1. A scrollable text field displaying the content of the *document* is at the top half of the frame; a log of editing history (action, position, time modified, etc.) is shown at the bottom half.

(continue into next page)

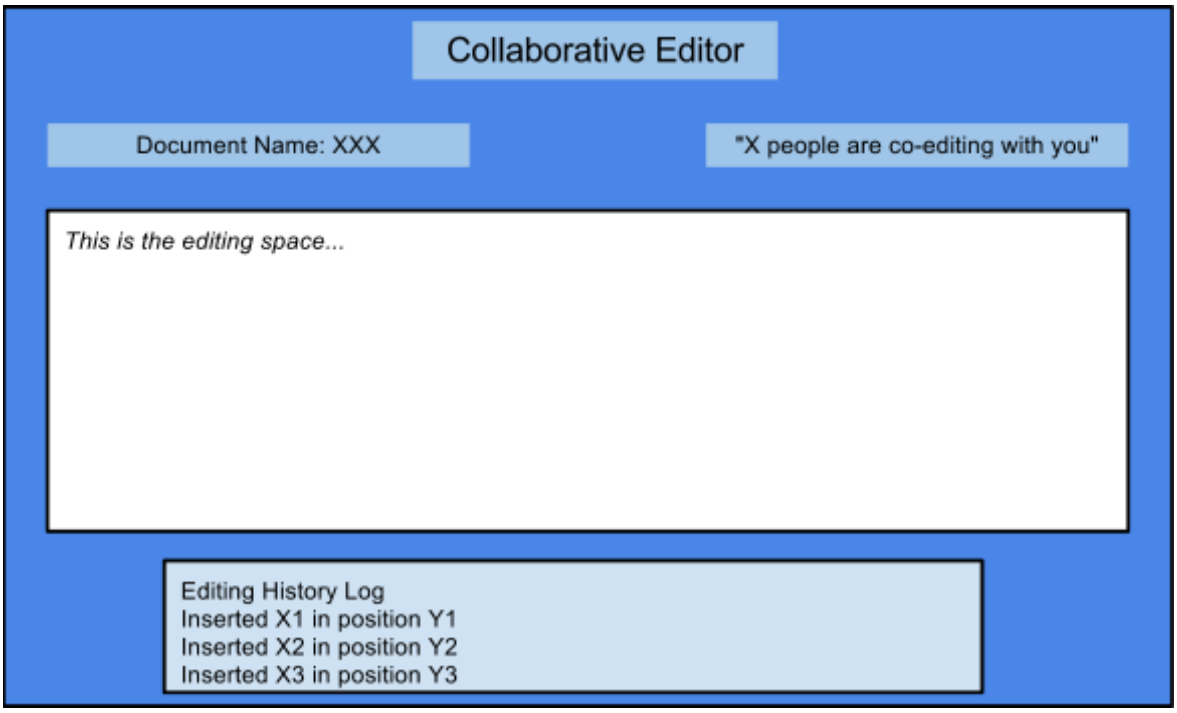

Figure 2.1 Editor View

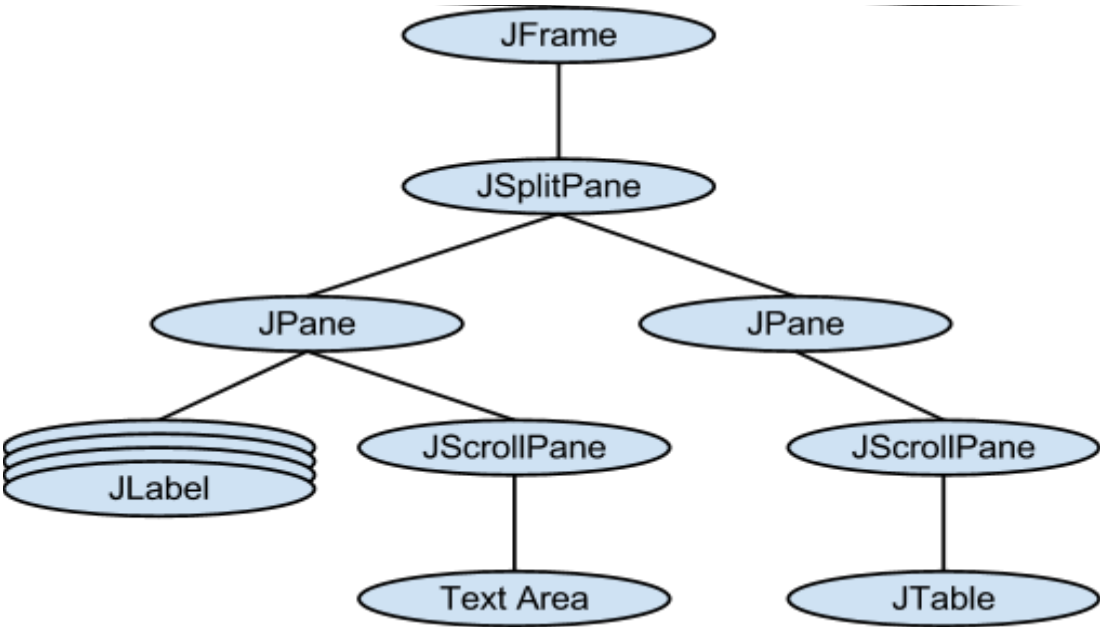

Figure 2.2 View Tree