Design Documentation

TalkBox App Final Version
Group 3

Authors:
Andrew Maywapersaud
Chandler Cabrera
Seo Hyun Jeong
James Kong

Table of Contents

Introduction

Our Talkbox follows a Model-View-Controller design pattern; the controller manipulates the model and updates the view based on user input.

Figure 1.1 – Class Diagram for Talkbox

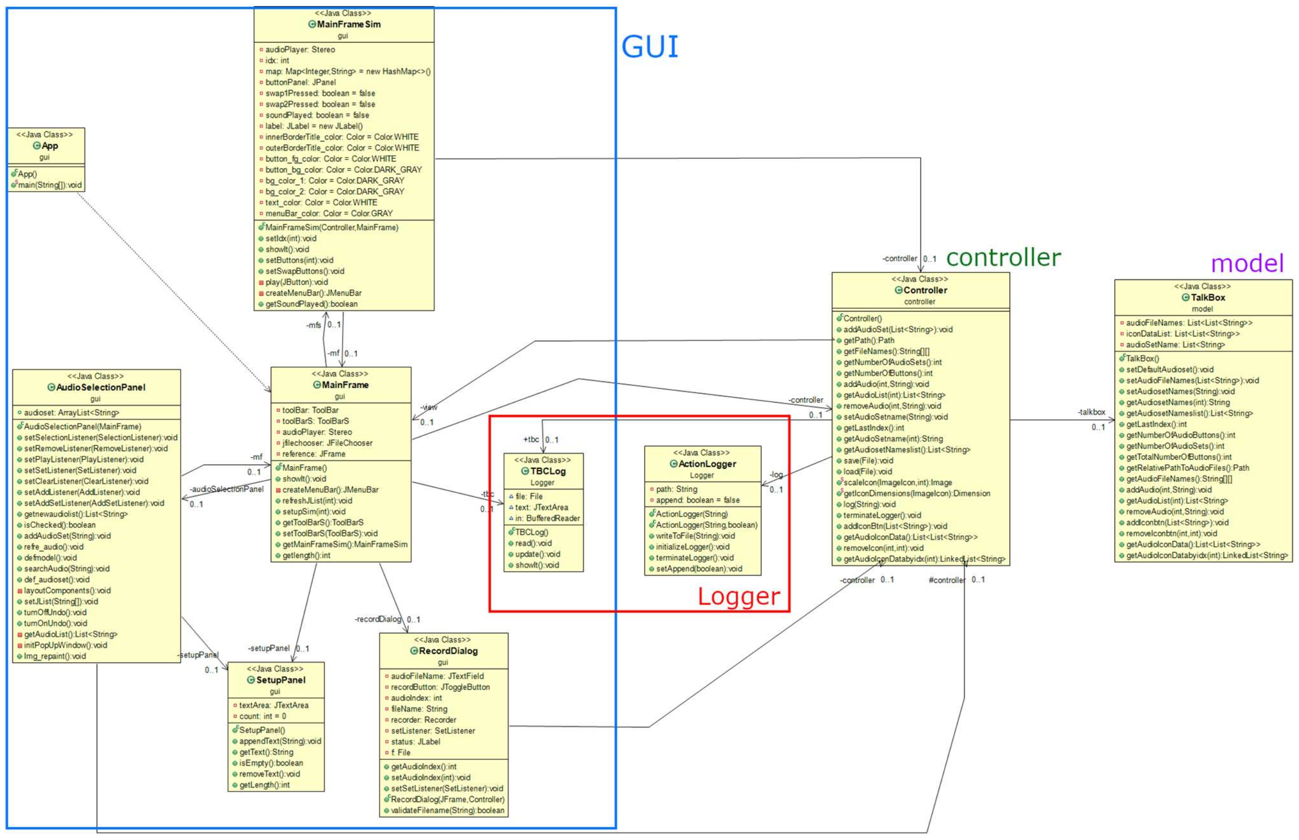
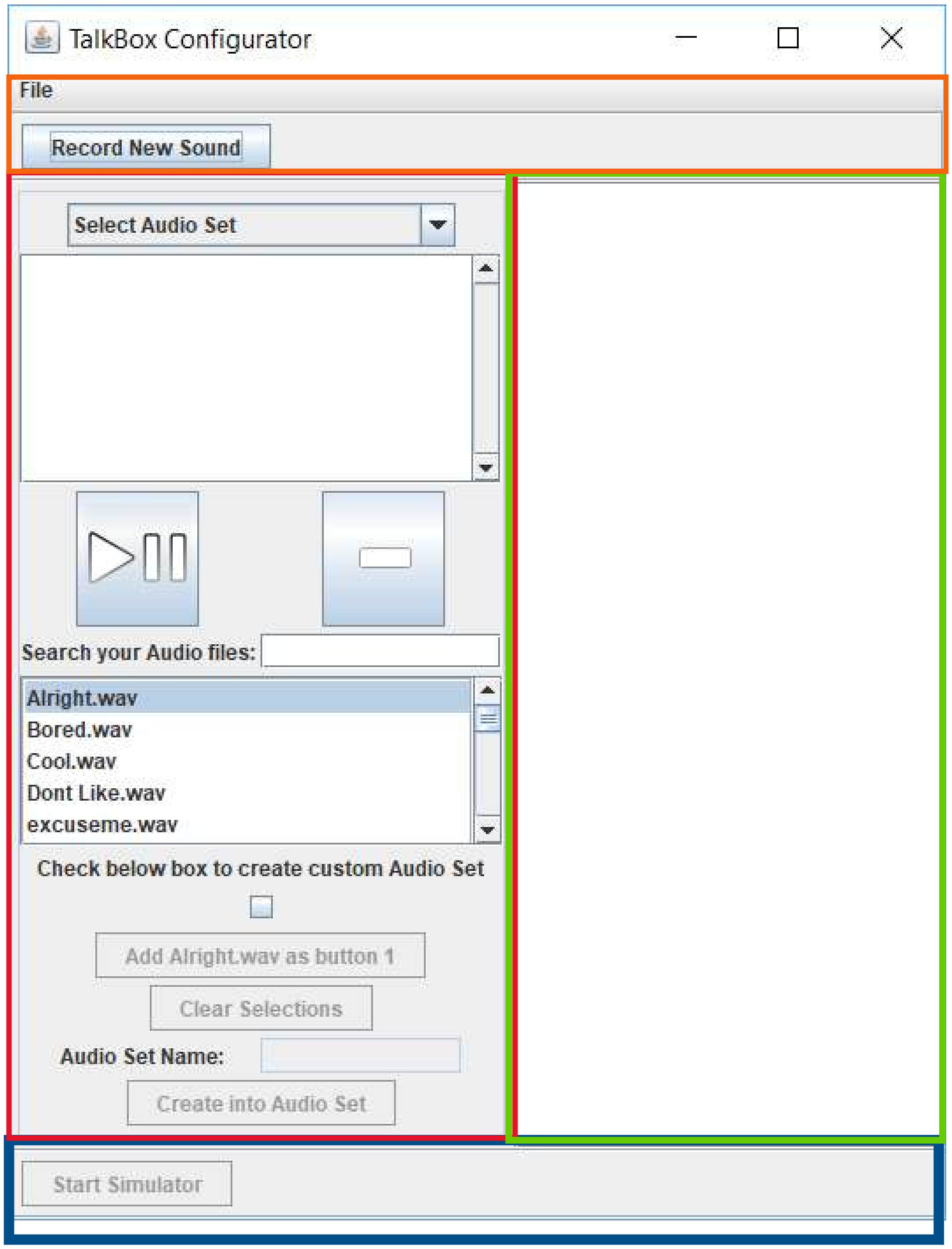


Figure 1.1 shows a high-level structure of the Talkbox app; the user launches the app and mainly interacts with MainFrame.java – like it's name describes, it is the main frame that houses different panels that are handled by other classes. Compartmentalizing aspects of the mainframe that pertain to different functions made it much easier for us to edit panels individually, and kept classes at manageable lengths.

Apart from the Controller and the Model, two classes make up the Logger – ActionLogger.java takes commands from the controller and writes it to a .txt file, while TBCLog generates a real-time view of the logger by using TBCLog.read().

Figure 1.2 – Mainframe and component panels



Pictured: North (orange) is ToolBar.java, West (red) is AudioSelectionPanel.java, East (green) is SetupPanel.java, and South (blue) is ToolBarS.java.

The controller has a hand in most of the other classes in the app, acting as a middle man by receiving commands and dispersing data or manipulating the user interface. For example, whenever an actionListener detects an input from the user, the controller is responsible for both logging the action and manipulating the model/GUI.

On the other hand, the Model (model.java) is seen the least by the other classes, as it only pertains to the general structure of the data. The model stores audio sets and the icons as a list of string-lists, where as audio sets are stored as string-lists. Generally, when the controller needs to manipulate one of these fields, the controller calls a method within the model that handles the data manipulation.

1.2 Runtime

At run-time, the main launcher App.java creates an instance of MainFrame.java. The MainFrame constructs one instance of each of its component panels (ToolBar.java, ToolBarS.java, AudioSelectionPanel.java, SetupPanel.java), and each of these panels generate their required components. MainFrame also constructs an instance of the controller, which each panel communicates through for their required controller methods.

Invisible to the user, the MainFrame also creates instances of TBCLog (the GUI representation of the logger's results, stored in log.txt) and RecordDialog (the dialog box used by the user to record audio). Since these are not immediately required, setVisible(false) is used to hide them until the user prompts them.

At this point, the user may launch the Talkbox Simulator to test it with the default audio set, create their own audio set by recording sound clips, or loading a custom audio set that was created and saved onto their computer.

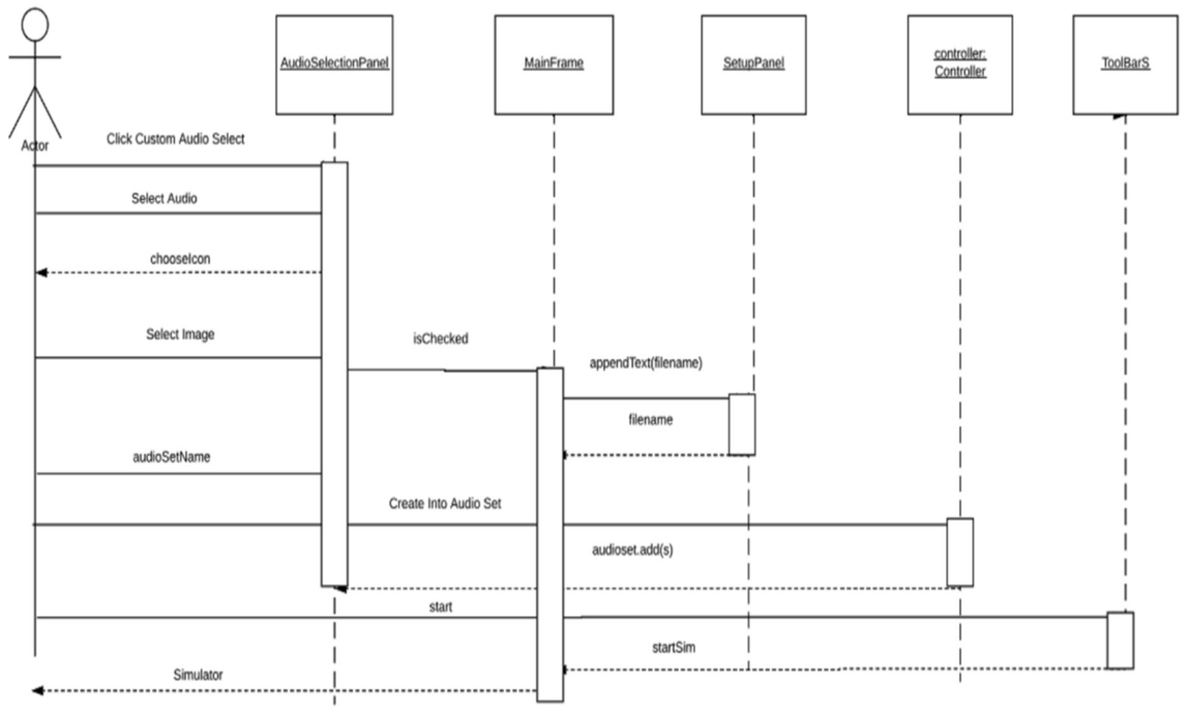
1.3 Talkbox Simulator

When the simulator is launched, the MainFrameSim uses the controller to load the corresponding audio sets (whether generated by the user or default) from the model, and dynamically generates the buttons based on which audio set is active. Each button has the same listener that plays audio (using the Stereo object, audioPlayer) based on the sound file associated with the button.

1.4 Sequence Diagrams

The following sequence diagrams depicts how all the GUI instances interact while on the main JFrame.

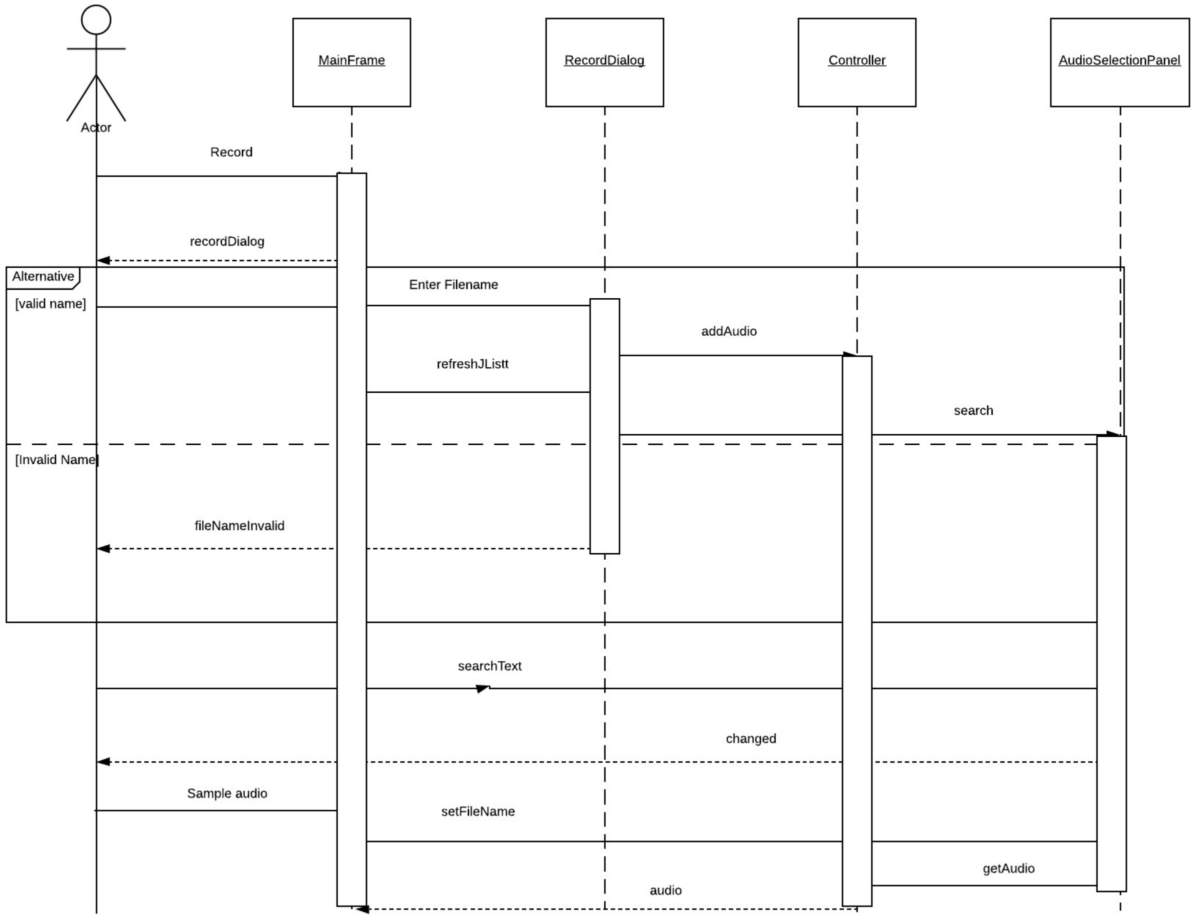
1.) Most frequent use



This sequence diagram illustrates the fundamental use of the TalkBox application. A user first interacts with the AudioSelectionPanel to select a custom audio set and then an audio file. The AudioSelectionPanel returns a request to the user to assign an icon to the button that would be produced if the set were to be instantiated. The panel interacts with the Mainframe to signal that the SetupPanel on the MainFrame should append the name of the selected audio. Finally, the user clicks the create audio set button. The controller instance is responsible for adding the newly created set to the list of current sets. The set is then available on the AudioSelectionPanel .

The user can now click the start button on the ToolBarS (South) located on the MainFrame to initiate the simulator. The Mainframe is responsible for rendering the simulator and returning it to the user.

2.) General user interaction



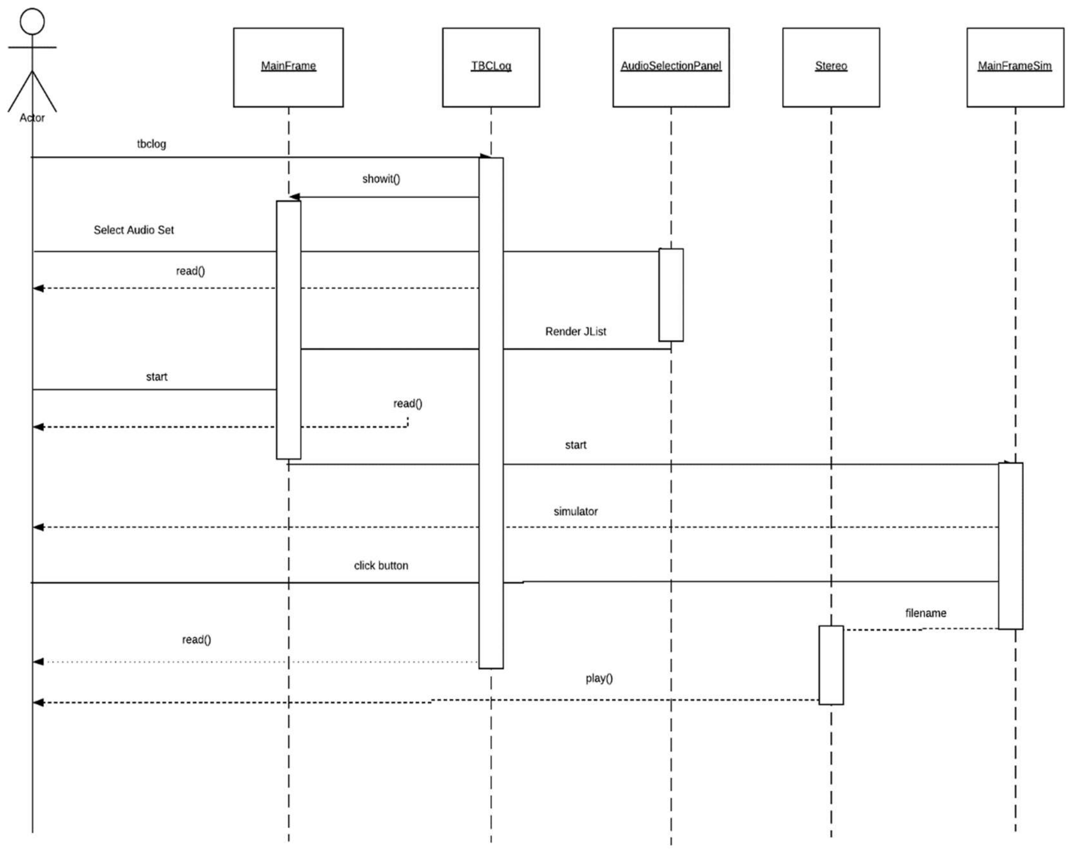
This sequence diagram shows a more general flow that the user can experience while running the application. This flow incorporates recording audio and not necessarily initializing the simulator.

First, the user interacts with the MainFrame to request the record audio option. The MainFrame returns a recordDialog. A conditional event emerges. If the user inputs a valid file name, the recording will be successful. Else, No recording occurs.

If successful, the controller instance will add the audio to the database while the recordDialog sends a message to re-render the JList on the MainFrame.

The user can now interact with the AudioSelectionPanel on the MainFrame. A searchText event will cause the AudioSelectionPanel to filter the audio list to match and return the text. Moreover, if the user interacts with the sample audio button on the MainFrame, the MainFrame signals to the controller to set the filename to be played. This information is retrieved from the AudioSelectionPanel. The controller then relays the information to the MainFrame where the user can interact with it.

3.) Sequence Involving Logging



This sequence ensures we cover all the functionality of the application. Mainly, we introduce the logger and the simulator.

First, the user opens the TBCLog. The TBCLog instance signals the MainFrame to render the logging interface. The user now interacts with the application. In this case, they select an audio set from the AudioSelectionPanel instance. Two things occur simultaneously: The AudioSelectionPanel renders an updated JList on the MainFrame and the logger sends a string to the user highlighting their input.

With the Audio set, the user clicks on the start button on the MainFrame. Two events occur again: The logger returns a string of the user event and the MainFrame signals to the MainFrameSim (simulator) to render itself. The simulator is now issued to the user.

When the user clicks on the audio buttons on the simulator, the logger depicts this information. Additionally, the MainFrameSim sends the filename registered to the button the the Stereo instance . The stereo instance calls it's play method, returning the audio to the user to be heard.

1.5 Closing Remarks

By using the MVC design pattern, and decoupling each component of the GUI into it's own class, it became much easier to debug the app. Having the app structured in such a way to reduce interdependency made it much easier for each member of the team to work separately on different parts of the app without creating too much conflicting code. Going forward, we would've liked to base the foundation of our app on JavaFX instead of Swing, as we ran into many issues as a result of its cumbersome nature.