System Design Documentation

Project: TalkBox
Date(s): April 3rd, 2019
Prepared by: Kai Xu, Eli Frungorts and Yash Desai

Document status: Submitted for Review

1. Introduction

The talkbox is a simple input-output device that registers a users input, processes the input, and outputs a sound corresponding the the input. The purpose of this document is to provide insight into the design of the systems architecture.

1.1 Purpose of the system

This system is designed to provide individuals who have speech and communication disabilities to communicate with the people around them.

1.2 Design Goals

The primary goal of this design is to have a modular, robust system that can be modified by all members, without risk of damaging the main system architecture.

1.3 Definitions, acronyms, and abbreviations

TB: TalkBox
CFG: configurator
Device: part of the system that aims to emulate the real life implementation of this design
MVC: Model-view-controller, a design philosophy for software design that handles processing behind the scenes, only providing users with a limited set of controls that they can use to modify the state of the system.

1.4 Overview

2. Current software architecture

2.1 Overview

The current system architecture is based on the MVC system, where the user interacts with a physical interface, and all operations are handled in the background, where they will be processed and sent back the the user.

2.2 Subsystem decomposition

Refer to [1] to see the UML breakdown of the talkbox
This code is designed to organize various components to be independent of each other. To clarify, the talkbox is coded in a way where the audio playing portion is explicitly separated from the visual interface. This is done so that when the code is ready to be placed onto a raspberry PI in the future, the visual interface code could be completely deleted, without the code require full reprogramming to function correctly.
If you refer to the diagram, you will notice that the only medium of communication between the visual interface and the device is a simple button list filled with 0's. When a button is pressed on the visual interface, the interface turns a 0 into a 1 in the button interface, and the device can register this 1, to play audio. If you think about it, this would be exactly how a raspberry PI would operate. It would constantly monitor the state of the input pins that run along its side, when a touch capacitive sensor is pressed, the voltage potential generated by the touch would register as a high, and the device would react accordingly.

The one caveat to this design is that, since the visual interface is dependent on system resources like the image files and config files, all panels contain a pointer to the devices configurator, which will tell the button what image it has and what its list position is.

Refer to [2] to see the UML breakdown of the configurator
The Configurator is structured in a way that everything is as locally contained as possible. This philosophy can be realized, as an example, when the user wants to edit one of the buttons, when they press on a button to edit it, the following happens. The panel that contains the button will create a localized JFrame titled Editbtn that will launch overtop of the current frame in focus. Inside the editBtn Frame, there are 3 main features. The first is the DragNdropLabel, which can take in any audio or image file, a file browser system, which allows you to navigate the local system file structure to find the image and/or audio file you are looking for, and finally, the RecorderFrame, which is a special frame that opens, and allows the user to record their own voice files that can be played in place of the default/computerized version.

2.3 Access control and system interaction

Below are the diagrams for the talkbox and Configurator, explaining the flow diagram from startup to system close.

Talkbox

The talkbox flow diagram describes the procedure that will occur when the talkbox is first opened. If you refer to [3] in the glossary, you will see that there are 3 color coded pathways. The first is the light pink pathway. This pathway is used to describe the process that happens when the user opens the talkbox. When this happens, the talkbox will allow the user to select their desired profile. Upon selection, the program gets to work, initializing the Talkbox, the device, and the Button interface. These 3 components all interact with one-another, so it is critical that they are all initialized before the user can even press the button. This Diagram futher illustrates the MVC implementation of the talkbox, demonstrating the lack of control the user has over the program beyond the set of controllers provided to them with the program.

Next if you follow the green line, you can see how the device operates when a button is pressed. Instead of directly interfacing with the audio device, the talkbox emulates a medium of communications; a button interface. The purpose of the button interface was described above, but in brief, it acts as a digital 'breadboard' that will register a high or low input, without consideration of intermediate levels, Ie: a 1 or a 0.

Finally, the red path is used to indicate how a user could switch between sets. When a user presses the switch set button, the buttonsetSelector can notify the visual interface to change the current button set it is working with. Additionally, when the button is pressed, a command is sent to the device, which simply changes a number in the device, which indicates the current audio set the visual interface is displaying.

Configurator

If you refer to [4], you can see the configurator flow diagram. This diagram is used to show the differences between the talkbox's design philosophy and the configurators. The configurator is different than the talkbox because, unlike the talkbox, there is no reason to seperate the audio and the images. Thus in this program, there are less containers, only truly having 3 layers of interface.
On the first level (the left most one), there is the user. The user will open the configurator, where they will select the configuration that they would like to modify. In this interface, the user must first select the config before any action can happen. This is important, as the other 2 interfaces fail to function until either a default or an existing configuration is selected.

Next, there is the main Interface, this interface level behaves as a preview level, where the user can see every button, and cycle through the sets of buttons to ensure all the buttons that they want are there.

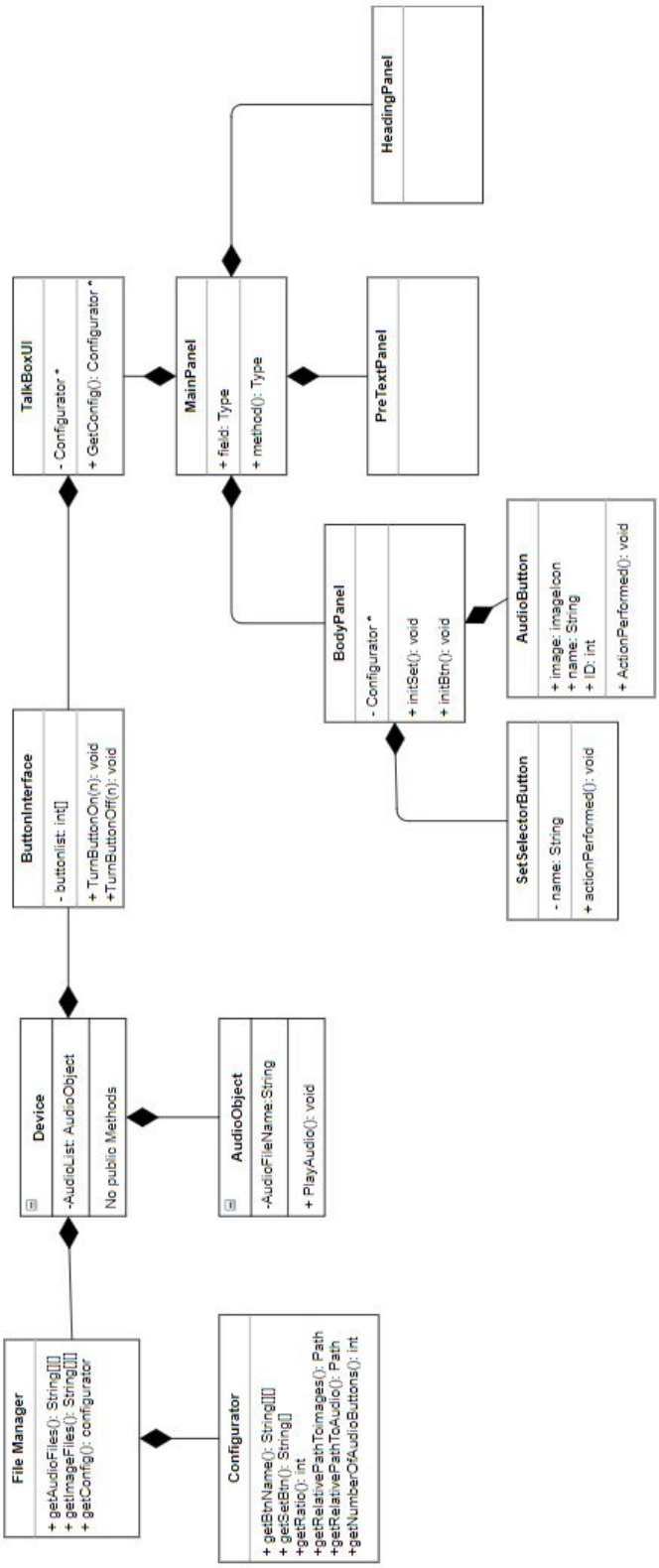
Finally, and most importantly, there is the Button Modification level. In this level, the user must indicate their desired audio files, Image files, and button names. This interface also enables the user to record their own voice, so managing where the files will be deposited afterward is of great importance.

3. Subsystem architectures

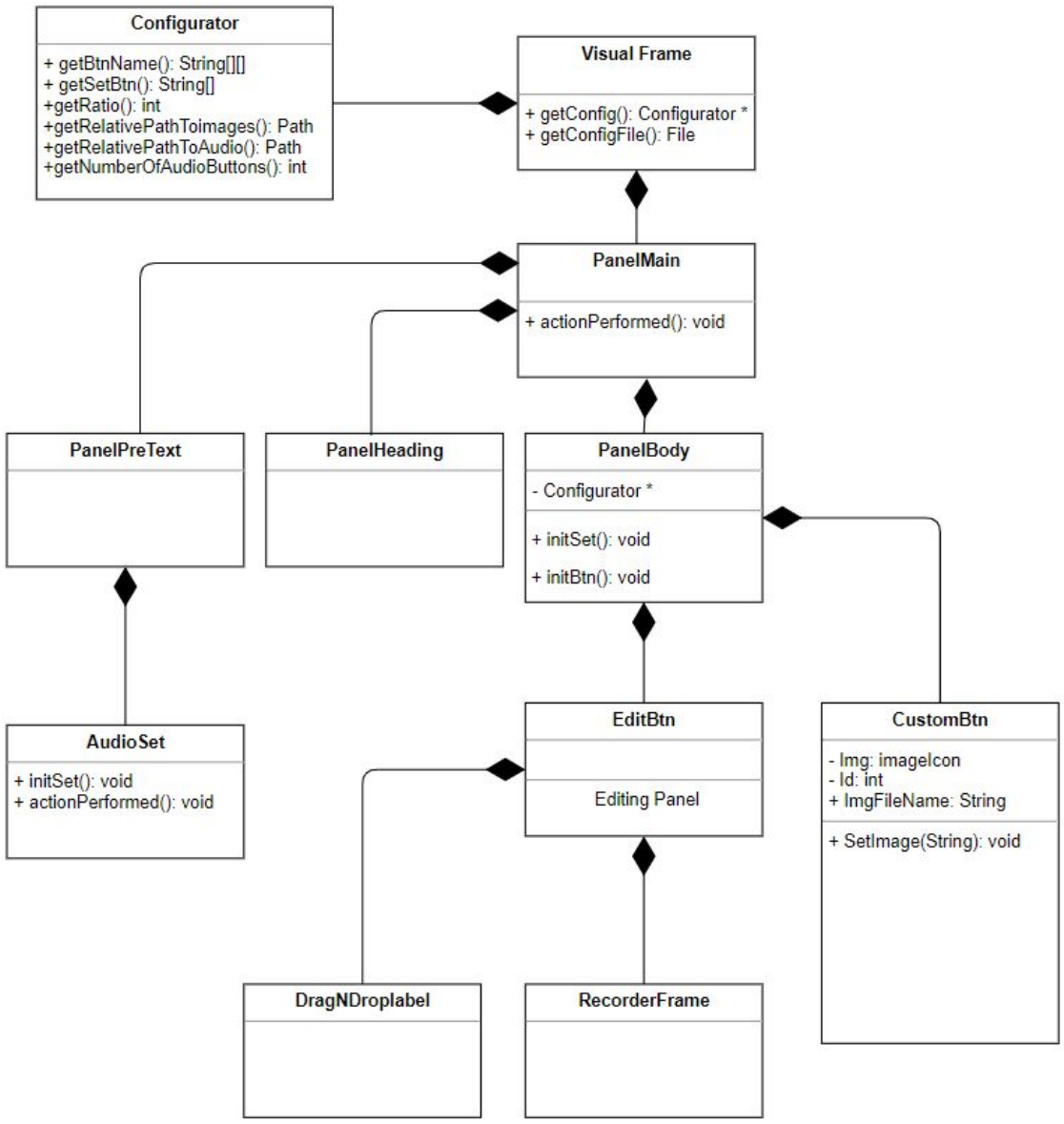
If you refer to [5], you will find the sequence diagram of a user opening the talkbox and pressing a button. This sequence diagram is disambiguation of the flow diagram[3]. The sequence diagram helps understand how the device monitors the button interface. In a perfect world, the button interface would be able to notify the device that it is in a high state. Although it is possible to implement such circuitry, it is far more reliable and modular to have the device instead poll the interface at a high frequency rate (roughly 10-60 times a second), to check if any change has occurred in the button interface.

4. Glossary

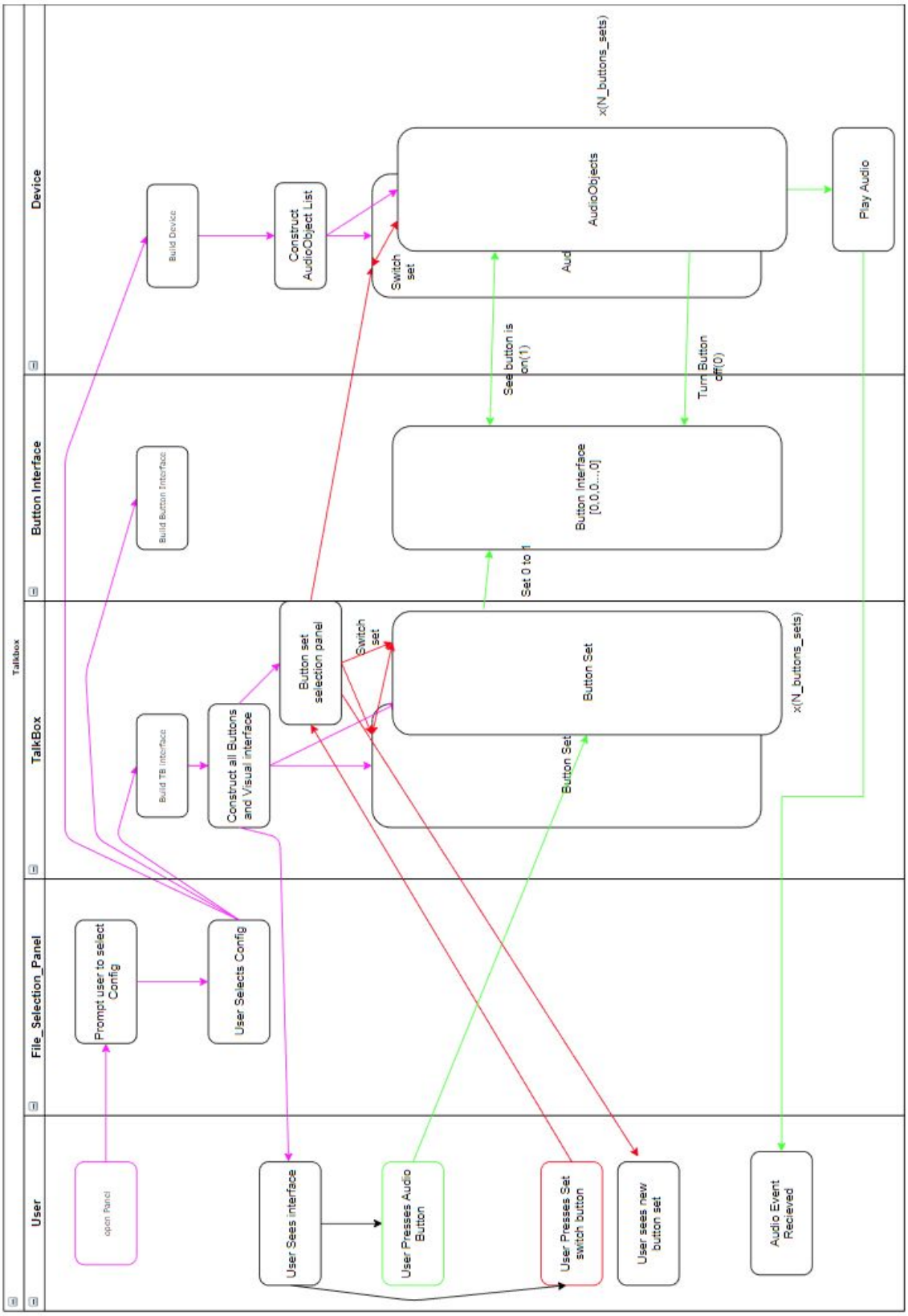
1.


HeadingPanel
TalkBoxUI
- Configurator *
+ GetConfig(): Configurator *
MainPanel
+ field: Type
+ method(): Type
PreTextPanel
BodyPanel
- Configurator *
+ initSet(): void
+ initBtn(): void
AudioButton
+ image: imageIcon
+ name: String
+ ID: int
+ ActionPerformed(): void
SetSelectorButton
- name: String
+ actionPerformed(): void
ButtonInterface
- buttonlist: int[]
+ TurnButtonOn(n): void
+TurnButtonOff(n): void
Device
-AudioList: AudioObject
No public Methods
AudioObject
-AudioFileName:String
+ PlayAudio(): void
File Manager
+ getAudioFiles(): String[][]
+ getImageFiles(): String[][]
+ getConfig(): configurator
Configurator
+ getBtnName(): String[][]
+ getSetBtn(): String[]
+getRatio(): int
+getRelativePathToImages(): Path
+getRelativePathToAudio(): Path
+getNumberOfAudioButtons(): int

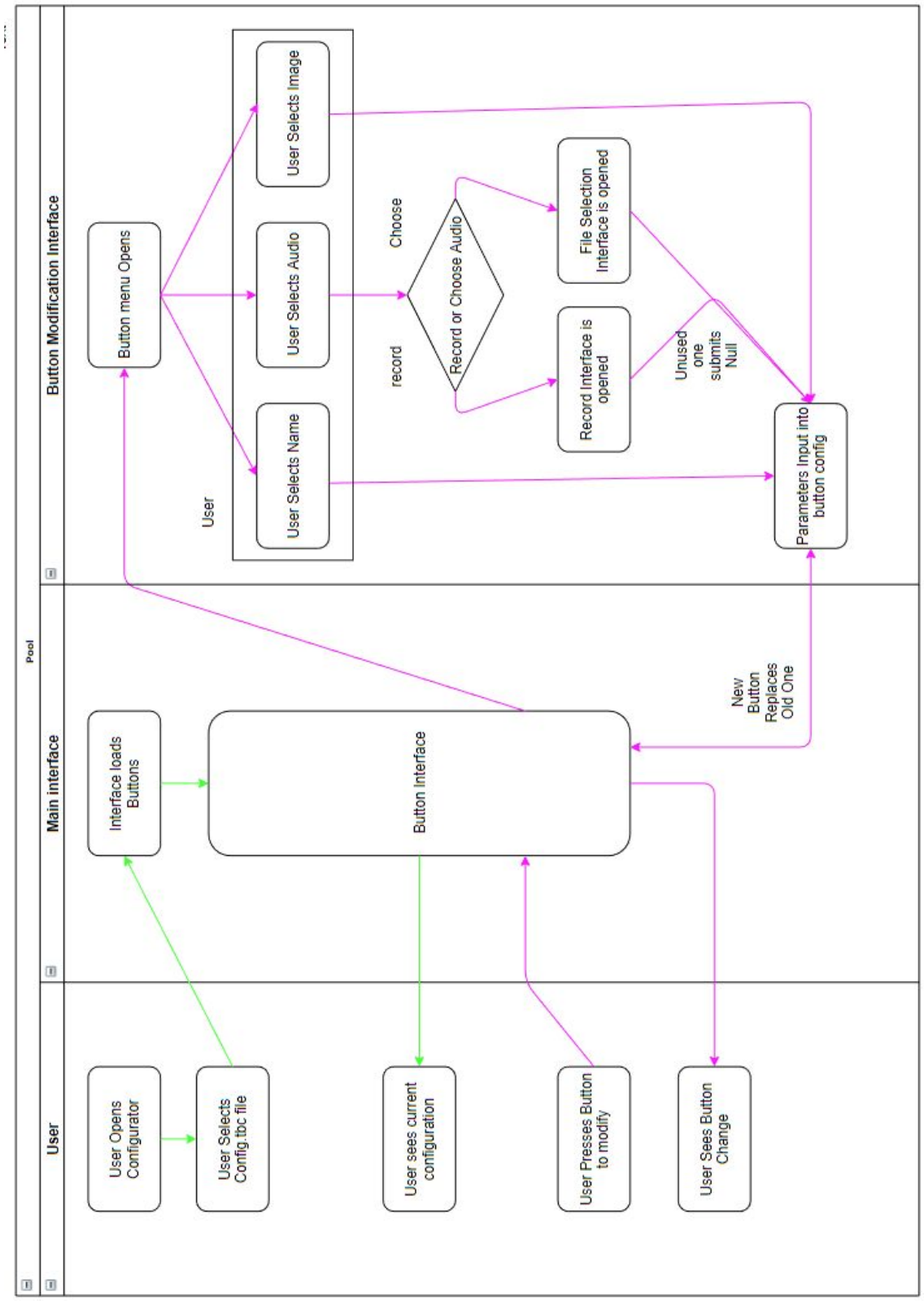
2.


Configurator
+ getBtnName(): String[][]
+ getSetBtn(): String[]
+getRatio(): int
+getRelativePathToimages(): Path
+getRelativePathToAudio(): Path
+getNumberOfAudioButtons(): int
Visual Frame
+ getConfig(): Configurator *
+ getConfigFile(): File
PanelMain
+ actionPerformed(): void
PanelPreText
PanelHeading
PanelBody
- Configurator *
+ initSet(): void
+ initBtn(): void
AudioSet
+ initSet(): void
+ actionPerformed(): void
EditBtn
Editing Panel
CustomBtn
- Img: imageIcon
- Id: int
+ ImgFileName: String
+ SetImage(String): void
DragNDroplabel
RecorderFrame

3. Flow diagram of the talkbox - higher resolution version attached in folder



4. Flow Diagram for configurator showing main path for changing a button



5. Sequence diagram of user pressing button