VIDIVOX
Development Report

SoftEng206 2015 Project

Jay Gradon, jgra718, 6320267

1. WHAT IS VIDIVOX?

Vidivox is a video editor designed for the purpose of editing the audio of user selected videos through the addition of other audio to the video. Vidivox currently provides support for adding audio to a specified portion of video, limiting the duration of said audio, looping said audio for a specified duration, and altering the volume of said audio relative to the video.

Vidivox is particularly integrated with text-to-speech synthesis systems, allowing users to quickly create and add computer synthesized commentary to a video. Vidivox currently allows users to select from a range of diphone voices, as well as altering the pitch or speed of speech. Vidivox also comes with two emotional settings, which use the pitch and speed to create speech that conveys emotion.

Vidivox utilizes a project functionality which uses a folder on the user's system to store dependent files. This allows the user to edit audio which has previously been added to the video or remove added audio altogether. The user should note that tampering with project files could cause unexpected behaviour on the part of the application.

This application is intended for a wide range of users, from professional to casual with primary functionality being immediately visible and having a wide range of features in the use of festival, the GUI for which is a primary feature in the application. Given that the application was being designed with the idea of simplicity with depth, the target user had a minimal impact as it is intended to be suitable for any user. The lack of a graphical representation of where audio is added does make the application less suitable for the technically unskilled (namely young children).

1.1 VIDIVOX DEPENDENCIES

Vidivox has been designed with the intent to be used on Linux systems, and has been particularly optimised in terms of the user interface to be used on the Ubuntu operating system.

Vidivox is dependent on three packages being installed on the users system, namely been the:

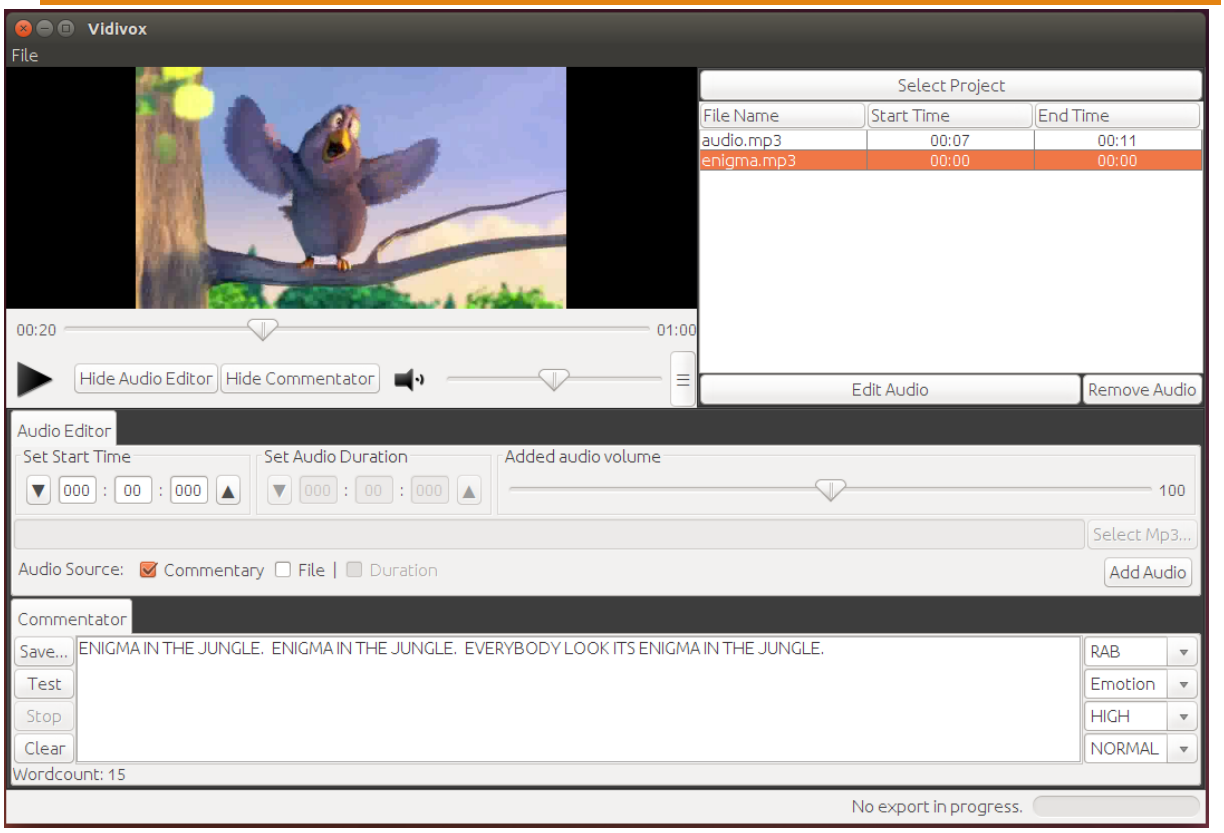
1. VLC video player package
2. Festival text-to-speech synthesis package
    a. For voice selection, the four basic diphone voices must be installed
3. Ffmpeg video editing package

In the absence of any of these packages the application will run with reduced or no functionality.

Vidivox has been designed to utilise .avi video and .mp3 audio files. While the application can handle other file types, other file types are not guaranteed to function correctly or well.

TABLE OF CONTENTS

2. GUI DESIGN



2.1 CHOICE OF PROGRAMMING LANGUAGE AND PACKAGES

The programming language used was Java, using the swing package for GUI development given the familiarity and frequent use of said package throughout the 2015 SoftEng206 course. While the package can at first be difficult to use, it swiftly becomes very powerful in enabling almost any GUI to be created once the developer has a good understanding of the different components and their purposes, or in particular, the different layout managers and their strengths.

2.2 COLOUR CONSIDERATION AND LAYOUT

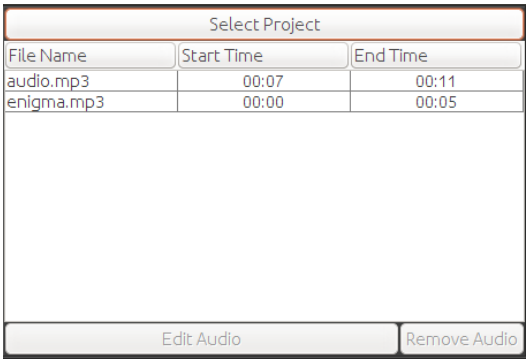
The application is designed for use with the Ubuntu OS system look and feel, and as such the colour scheme has been developed with consideration for the Ubuntu colour palette. Some hardcoded colour aspects, such as the dark background behind the tabbed panes of the Audio Editor and Commentator, are sourced from the Ubuntu Colour palette, as Ubuntu is the primary system the application has been designed to be used on, alongside other Linux OS. The purple colour of the mute and play buttons hover icons are also sourced from the Ubuntu colour palette and were designed by Akram Darwazeh, my assignment 3 partner.

The layout of the application was developed with concept of functional modularity in mind, where there are six primary modules in the GUI design (identified graphically in the user manual). The six modules include a menu bar, the video player, the audio list, the audio editor, the commentator, and a footer with the export progress. JTabbedPanes over a

dark background were used to enforce this modularity, where the tabbed panes are clearly distinct from each other in the form of the Audio Editor and Commentator. This ensures that users will not become confused as to what buttons or functionality refers to what part of the application; for example, the *"Save…"* button on the commentator is now unlikely to be interpreted with saving edited audio or video, and does not force an overdependence on borders to separate functionality. The JTabbedPanes also allow for future development in the form of such functions as a graphical representation of added audio, or an advanced commentator that allows the stitching of multiple voices talking as if having a conversation, through the use of additional tabs.

The use of distinct modular panels and proper layouts (namely nesting BorderLayouts for major panels) also allows panels to be easily hidden and the GUI to be resized while retaining a high quality look. Allowing the user to hide functionality also makes it easy for the user to only see what is relevant for their task at hand, and not have a cluttered GUI.

2.3 PRESENTATION OF INFORMATION



The presentation of the information on added audio is also represented in a somewhat modular format, with all audio having a brief information presentation in the form of their start and end time in the audio list on the GUI's right side, and that of a detailed presentation when editing a piece of audio. The detailed presentation is in the form of all fields – the start time, duration, volume, and file path or commentary settings all being displayed when editing a piece of audio. This means that while the user can quickly get detailed information about a piece of audio by selecting the audio and clicking the *"Edit Audio"* button, they are not inundated with information on all pieces of audio at once, as they would be with a larger and more informative list. This also allows a specific audio file to be easily found due to the compact nature of the list.

All settings in the Audio Editor panel are labelled to be easily understood, with names that are as compact as possible. There is also a tooltip on the time setters to inform the user that the expected format is minutes, seconds, and milliseconds.

The combo boxes in the Commentator panel use a custom renderer so that they initially display descriptive titles informing the user of each setting. This ensures the user is quickly aware of how much they are able to modify their synthesized speech, and does not introduce possible error as the initial title is not a selectable setting. The voices use their festival names, as while a new user will not understand the difference between Rab and Don initially, they will quickly recognise the difference through experimentation. It was also difficult to give all four voices short descriptive names as while being different, they are similar, with Rab's accent being the only notable difference.

The idea that the user can quickly determine functions through experimentation or the manual also applied to the use of the triangle icons for setting the start or duration times, where icons reduced the clutter of words which would only be useful to a novice for a short time (see the audio editing panel).

3. VIDIVOX FUNCTIONALITY

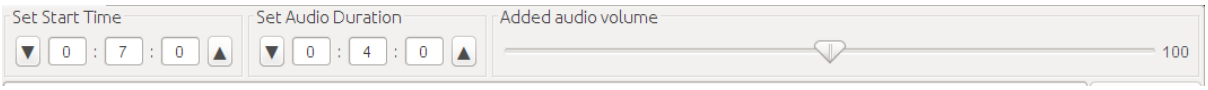
3.1 MOTIVATION FOR SELECTED FUNCTIONALITY

3.1.1 ADDED AUDIO SETTINGS

The basic functionality requirement for the Vidivox application was the ability for users to add audio to a specified point in the video. The included audio editing capability in this Vidivox application was:

1. Setting when the audio would start during the video playback
2. Setting how long the added audio is
3. Setting the volume of the added audio

The user also has the option of adding a selected audio file or the synthesized text directly from the commentator. Audio settings were given borders and titles for swift distinction between setting functionality.



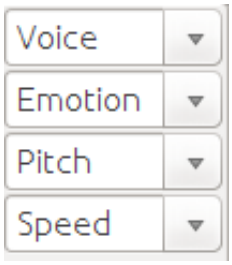
Setting the start time of the audio, beyond being a requirement, allows the user to accurately add commentary to specific points of the video, without needing speech throughout the entire video. Setting the length of the audio is disabled for commentary to prevent users believing the commentary will complete during the audio duration, but is enabled for audio files. This serves the purpose of shortening added audio and looping short audio.

By allowing users to set the duration of the added audio, it enables them to use long audio files, such as instrumental music, and only overlay it where they want on the video, as well as enabling the prevention of the music from extending past the length of the video should they not want it to. Conversely, it also allows them to loop shorter audio, like a music snippet, over a longer portion of video without adding the file in multiple times. It felt sensible that should a user be able to choose when the audio starts, they should also be able to choose when it ends.

Users are also able to set the volume of the added audio. This is so that if a user is overlaying music, they can make it quieter so they can still hear commentary or people in the original video speaking. It can also be used to make commentary louder, as otherwise the commentary is generated at a lower than optimal volume.

A piece of primary functionality that would be the first new function to be added would be giving the user the option to have a new piece of audio replace the existing audio. This was not, however, implemented due to the consideration of time constraints.

3.1.2 COMMENTARY SETTINGS



The Vidivox application allows the user to select a voice, emotion (which overrides and disables pitch and speed settings), pitch and speed. The voice functionality had already been implemented in assignment three, with other settings being demonstrated in lecture. These settings were included as they provide simple customisation for the commentary and allow depth in terms of the possibility of creating commentary that is distinct from other created commentary.

3.1.3 PROJECT SETTINGS

The use of a project was developed so that users that desired to extensively modify a video could edit a video over time without leaving the application open, or edit previously added video without redoing all previous edits or managing a series of files with different audio. The use of a project was sensible as with the addition of more functionality requiring objects to manage added audio, such as displaying audio in a list the start and end of the audio in the video, project functionality was just one more step.

It was decided to make a project folder centric so that the application could create dependent files (such as audio commentary) and copy added audio files to one location, rather than a single text file that linked to file locations so that the use of ordinary files was not impeded by the application.

The primary motivation behind the establishment of project functionality was the ability for a user to maintain progress over several sessions and edit previously added audio, two functions which significantly improve the user experience due to convenience and ease of use (i.e., a mistake in adding audio is not a serious detriment to time spent editing a video).

3.1.4 VIDEO PLAYBACK SETTINGS

It was decided to keep the video playback settings simple to avoid cluttering the GUI with unnecessary functions that added little value to the application. As such, fast forward, rewind, and frame skipping functions were removed from the application (after being present in the prototype) in lieu of the progress bar for skipping or searching through large segments of video, and the time setters in the audio editing panel for smaller skips, down to milliseconds, enabling users to go to any exact time or frame.

3.2 USABILITY

3.2.1 SINGLE FRAME FUNCTIONALITY

A primary usability decision was to keep all of the functionality on one frame and mostly outside of dropdowns. This was so that the entire functionality was quickly obvious and the user would not have to manage keeping multiple frames visible at once or would have to switch between different frames; all information is readily available in a sensible layout at all times.

3.2.2 MODULAR LAYOUT

The second primary usability decision was the modular layout of the application. The layout allowed portions of the application to be easily hidden, and made it obvious as to what functions referred to what part of the application, whether that be editing audio or creating commentary.

3.2.3 PROJECT CAPABILITY

The project was also a large usability decision as it allows swift editing of the video without the user managing their own changes should a mistake be made.

4. CODE DESIGN

4.1 SOFTWARE ARCHITECTURE

4.1.2 PACKAGE HEIRACHY & CLASS SEPERATION

The Vidivox application contains seven packages, including:

1. FESTIVAL ENUMS

The festival enums are FestivalVoice, FestivalPitch, and FestivalSpeed, containing the required festival settings for each type, so they can be easily inserted into a festival bash command for synthesis.

2. VIDIVOX AUDIO

The audio package handles the object classes (the Audio interface and the Audio Commentary and Audio File classes) that are used to store information on the audio added to the vidivox project.

3. VIDIVOX FILECHOOSERS

The file chooser package handles the file choosers and their custom file filters for the purpose of being used in the application, namely a ProjectChooser, VideoChooser, and AudioChooser.

4. VIDIVOX GUI

The vidivox GUI package handles the GUI components, including the menu bar, main frame, and significant panels such as the commentator panel, audio editing panel, and list panel, with significant panels and the menu bar being separated into their own classes. As the panels in the GUI interact with each other often throughout the application, particularly the list and editing panel, the components in the package were given protected visibility to allow the GUI to easily interact within its individual classes. The images package with the various icons for play, pause and volume is a sub-package of this package.

5. VIDIVOX MANAGERS

The managers package contains adapters with custom functionality for the VLCJ media components and managers to handle the exporting of video or the creation of commentary using the swing worker classes.

6. VIDIVOX WORKERS

The workers class contains the three swing worker classes responsible for spawning and managing bash processes, namely ffmpeg, festival, and text2wave.

Packages attempt to group similar or dependent functionality.

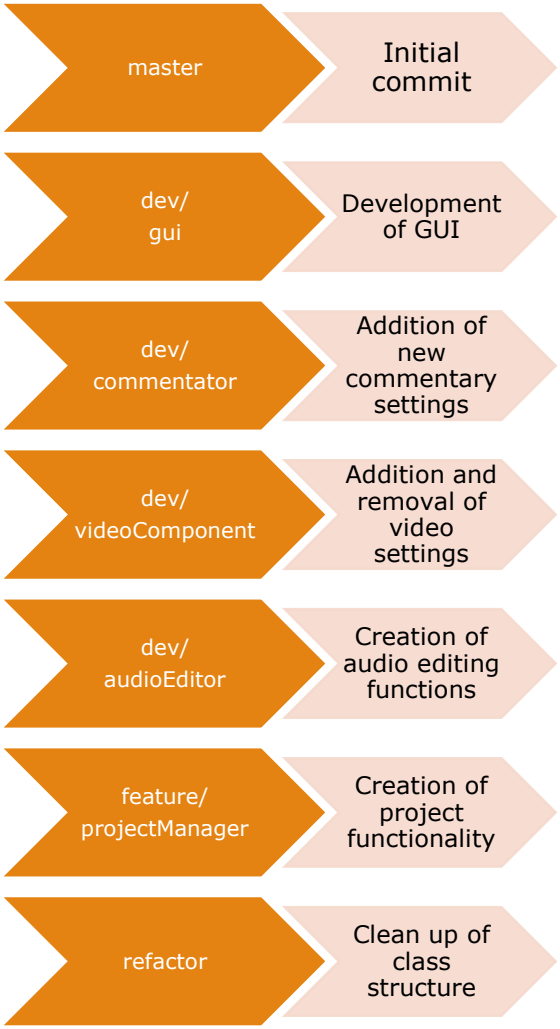
4.1.2 POTENTIAL DESIGN CHANGES

A portion of the design that would be changed is the handling of creating of new audio objects would be further separated from the GUI classes, which handle much of the getting of information from the GUI components themselves. This would be refactored out into the Project Manager class.

A second design change would be an effort to reduce coupling in the GUI package through the use of more methods that can handle inputs, rather than direct access of fields. Fields are directly accessed as the GUI was originally less separated and was later refactored into a larger number of separate classes, as the original GUI classes became too large over time.

4.2 DEVELOPMENT PROCESS

The development followed a process of creating the basic interface, followed by basic functionality, followed by more complex functionality and refinement of the GUI to match any new unanticipated functions or removal of functions that were not able to be completed during the time constraints. This process was reflected in the use of GIT branches:



5. EVALUTATION AND TESTING

5.1 TESTING THE APPLICATION

Rudimentary testing was manually performed on the application, as the testing was, for the most part, seeing if GUI elements responded correctly, or the video sounded correct in the addition of the audio.

Multiple bugs were found, primarily after the refactor, where some small portions of code were lost (such as the commentator combo box listeners).

Other errors were found through the use of the application where some oversight was required, for instance not realizing that a given button should be disabled at a given time.

Manual testing was primarily used (alongside debugging functions) due to time constraints pressuring into not developing a proper test suite, which may contain errors of its own, and did not have prior knowledge as to how everything would be implemented due to lack of experience with such applications.

5.2 EVALUATING THE APPLICATION

5.2.1 BETA FEEDBACK

The primary change that was to the application due to feedback was setting the visibility of the editing and commentary panes to false upon opening the application, as while it was initially decided to make all of the functionality obvious there were comments that it was overwhelming as to where to start at first.

The *"select project"* button was also added on the main frame to make the starting point more obvious, as some users commented that the file drop down was not obvious.

Other comments were primarily on partially/unimplemented functionality that only addressed intended development as had been specified in the provided readme; such as error reporting or the project viewer.

5.2.2 DEVELOPMENT SINCE BETA

1. IMPROVED CLASS STRUCTURE

The class structure has been significantly improved since the beta with the GUI being split up into several more classes and additional packages being used.

2. PROJECT FUNCTIONALITY

Prior to the beta the *"add audio"* function was being used to test ffmpeg commands. Since the beta the use of ffmpeg has been shifted to the *"Export Video"* function and *"add audio"* deals with creating audio objects and the project viewer is not a stand in GUI component with dummy text.

3. ERROR CHECKING

The application now has more error prompts when a project is not loaded. Most buttons are disabled when they are not to be used.

6. FUTURE DEVELOPMENT

6.1 SIMPLE DEVELOPMENT

6.1.1 ADDITIONAL GUI REFACTOR

Some of the GUI classes (namely the audio editor and list) need to access many fields in the other GUI classes and also handle the creation or modification of audio objects or changes to the list of audio objects.

The handling of audio should be moved into the project manager class, while methods need to be created for the accessing of fields in other classes in an appropriate fashion.

6.1.2 REPLACING EXISTING AUDIO

A simple function that could be implemented is the option for added audio to replace sections of the existing audio, or to remove portions of the existing audio from the video (without removing all of the audio entirely).

6.1.3 SORTING THE LIST OF AUDIO

Allowing the user to sort the list of audio by start time, end time, or filename would be helpful for users with a lot of audio to find or compare audio.

6.2 ADVANCED DEVELOPMENT

6.2.1 GRAPHICAL REPRESENTATION OF ADDED AUDIO

Graphically representing the audio would require some redesign of the GUI to prevent crowding, unless the tabbed panes became more advances, such as being able to drag tabs between panes or drag tabs out to create new panels, so that the user could see what functionality they needed at a given time. This would help users in seeing how audio is being added and compare when audio is playing easily.

6.2.2 ADVANCED COMMENTARY

A commentator that allows users to create commentary with multiple voices talking as with a natural conversation, alike to two commentators on a sports game would be interesting and could be incorporated into the GUI as an advanced commentator tab. It would involve have lines of text where the lines are synthesized in a defined order with defined gaps between speakers. This would help users that are trying to emulate conversation significantly, not having to create and add commentary separately.

7. CONCLUSIONS

Developing vidivox was a valuable experience. The pair development of the prototype was the most valuable experience I have ever had in learning how to work in a team, as both partners had to understand how functions were being implemented. This meant code had to be easy to understand and work had to be assigned well to help prevent merge conflicts.

The development also showed the importance of not just planning in the initial stages of development through screen and class diagrams, but also the importance of stopping to evaluate and plan the continued development in stages, which would have helped immensely as the code developed and the structure had to be changed to cope with the larger than expected amount of code and coupling. More planning after the initial stages would have meant less refactoring and less messy field accessing.

Vidivox was also most interesting as an exercise in GUI development, and increased my appreciation for the ability to create GUI's that are well structured and appealing, and that fit their intended user environment well.

Using external packages such as swing, festival and ffmpeg also increased my awareness of the importance of adequate documentation, as many functions (particularly for festival, and somewhat for ffmpeg) were difficult to discover and learn the syntax for due to unhelpful documentation.