SaLED (**S**peech **a**nd **L**anguage **ED**itor) Software Development

GRP Report, Ver. 1.6

Evan Karim

9/6/2024

GRP, Summer 2024

Embry-Riddle Aeronautical University

Daytona Beach Campus

1 Aerospace Boulevard

Daytona Beach, FL 32114

Table of Contents

Introduction

Speech and Language AI (SaLAI) models are constantly being tuned and trained to better understand human speech. In order to effectively train these models, they must first be trained on existing transcripts, before being prompted to transcribe themselves. However, even after the initial round of training, the model will not be completely accurate. It is important to provide fixed versions of these transcripts back to the model, so that it's errors can be fixed. In order to accomplish this, the model users must manually modify the transcript so that it can be fixed for errors. While transcript editors exist, such as Praat, the stakeholders involved want an improved version, that combines a number of functionalities from other transcript editors, and provides a better, cleaner user interface. The goal for this project is to develop a transcript editor, that would allow the addition and exporting of transcript files and their associated audio. This would be the foundation for developing the software in the future, which aims to include other functionalities, such as providing an audio file to a model in order to upload a transcript directly to the program, or providing the user with more tools to modify the provided audio.

Abstract

There are many tools and processes needed to optimize the learning of SaLAI models, including transcript editing tools. The need for a new transcript editing tool has led to the development of the Sound and Language Editor (SaLEd). To develop this new software, an Agile Scrum Software Development Lifecycle had begun. By utilizing Requirements, Design and Architecture the development of the software smoothly progressed. This paper aims to inform people about the project's development, and its progress, as well as establishing artifacts for future developers to have as resources.

Background

The project itself is the development of a new software tool, so the methodology itself begins with an explanation of the Software Development Lifecycle (SDL) and how it is applied to this project. This project involves the use of transcript files, as well as AI Models, which are usually implemented in Python. From this alone, we understand that the paradigm we are using is the Object Oriented Paradigm(OOP). Additionally, there is a relatively small development group of only 4 people, and we will constantly be in touch with our stakeholders at all times. All of these contextual aspects of the system's development indicate a strong favor towards an Agile Scrum process.

The selection of an Agile Scrum process significantly changes the way the team operates. In a typical SDL such as Waterfall, the program is created in a series of linear steps. First the Requirements are created, then the Design, then the design is Implemented into code, and finally Verification & Validation (V&V) is performed to ensure the code is built correctly. Each one of these steps can take weeks or months, depending on the size of the project. In an Agile Process, instead there are periods of time called Sprints, in which one entire cycle of the software is to be completed. Each sprint is to be self-contained, and with each Sprint the developers build upon what was created in the prior Sprints. This process is completed until the software reaches a satisfactory state for the stakeholders. Our project's conditions suit an Agile process better

because we have frequent access to our stakeholders. With frequent access to our stakeholders, we can incorporate them into meetings more frequently, giving them a better understand of our progress, and being able to ask them about aspects of the system, such as User Interface (UI), functionalities, preferences and priorities. OOP is also important for the iterative process, as the abstraction of code as objects helps contain sprint objectives. Each sprint objective or task can be abstracted to an object that needs to be created, allowing for the system to be easily modularized and modifiable. Even though we are in an Agile process, we still make use of the major points of the SDL, which are Requirements, Design and Architecture, Implementation, and V&V.

Defined Terms

Software Development Lifecycle (SDL) – The process taken to develop a piece of software.

Object Oriented Paradigm (OOP) – A paradigm of programming that seeks to model data structures as "objects" that can be created and manipulated within the system.

Agile Scrum – A SDL that aims to have high stakeholder interaction and a continuously iterated product.

Waterfall – A SDL that focuses on a step-by-step development, in which the product is completed in phases.

Verification and Validation (V&V) – The process of ensuring the product being built is what the customer wants, and that it functions as designed.

Requirements – The needs/wants of the stakeholders involved with the system/product.

Requirements Elicitation – The process of communicating with the stakeholders to extract information pertinent to the product's needs.

Functional Requirements – Requirements of a system that focus on the system's functionalities. (ie. The system shall allow the user to make 3 login attempts before locking them out of the system).

Non-Functional Requirements – Requirements of a system that focus on qualities or attributes of the system that cannot be tested. (ie. The software shall be simple and easy to use).

Unified Modelling Language Model (UML Model) – Also called a Class Model. Illustrates the classes of an Object Oriented system, and their relationships between one another.

Data Flow Diagram Model – Illustrates the flow of information within a system, and how it relates to the actors.

State Chart – A model that illustrates a view pertaining to the functionalities of a system, and what actions can be taken in certain states.

Requirements Elicitation

The first place the Software Development Lifecycle starts is generally Requirements Elicitation. After reviewing the information that was provided in the project description, many questions arose regarding the operations and structure of the system. Two interviews were conducted with stakeholders. The first interview was with the product owner, Dr. Jianhua Liu, who gave detail into their needs and wants. It was established that the primary concern for the stakeholders was that they just wanted a running version of the transcript editor, and that convenience was a non-issue. They prioritized the performance and responsiveness of the system less than wanting the system to have the core functionalities working. However, this does not mean there was no desire for convenient features. They expressed that they wanted features like hotkeys and responsiveness, but that the development of these things did not have to occur in the first iterations of it. This was similar for other features, such as a login system and the connection of the editor with the AI engines.

After the first interview, another interview was done with Dr. Liu's partner, Dr. Andrew Schneider. Dr. Schneider acted as both customer an experienced user of transcript editors. More key information was elicited in this interview as well. During the interview, the topic of Video Editing Software, such as Adobe Premiere Pro, was brought up in reference to ways to manipulate audio. Dr. Schneider discussed how many of the functionalities of current transcript editors such as Praat or Elan were built focused less on the side of audio manipulation and more on the side of supporting AI models. He described the need for the system to follow more like Editing Software and less like current transcript editors, as it was more valuable to have methods to manage the audio and its transcript like he would in editing software. Additionally, the topic of cross-platforming was brought up, in which it was discovered that instead of developing the system for one or more specific platforms, instead that the system was to be hosted on a server, and users would connect as clients to the server and be given a webpage interface to interact with. This way, we could circumvent the need to develop on multiple platforms while also opening up a central area to store modified transcripts in the form of the server, which could host a database. These points would be crucial to the development of the system, providing one each of a guideline to the system's functionalities, and also a major constraint that influenced the structure of the system.

User Stories

The first artifact we had created was a series of User Stories. Utilizing both the provided project description as well as the information elicited from Dr. Liu and Dr. Schneider, we were able to establish some User Stories to create a prototype version of the product. These User Stories are included in the ATS, and also exist within another branch of the GitHub repository. Generally, these User Stories were more focused on actionable things the user could do while operating the program. This included actions like editing the transcript, being able to see the waveform and change what parts to look at, and many of the operations involving interval manipulation. Additionally, instruction for how new projects would be handled or which hotkeys

would be established existed as User Stories, but did not see creation in the prototypes for the project. However, the most important User Stories provided the most guidance on how the project would be constructed going forwards, and gave the team a strong abstraction of the project's structure.

Requirements

Please see Appendix A for a list of Requirements, derived from the SRS

After the User Stories were established, Requirements were created using the project's prototype as a guideline. These requirements are all detailed in the Software Requirements Specification (SRS). The first thing that needed to be established was the Functional Requirements (FRs) and Non-Functional Requirements (NFRs). Each set of requirements was split into multiple sections based on category the major parts of the program fell into. For FRs this included:

General Requirements – Functionalities of the system that are key to its operation, but do not fit into any distinct category.

UI Requirements – Functional Requirements that involve modifying or interacting with the User Interface in some form.

Database Requirements – Functional Requirements that involve the manipulation or verification of data within the database.

File I/O Requirements – Functional Requirements that encapsulate all operations involving File input and output.

Currently, each of these categories highlights the major aspects of the system, which will be further detailed in the "ASRs and Architecture" section of this report. Notably, the UI Requirements section is currently the largest, as it deals with the majority of the system's functionalities. Most of the system operates in a call and response relationship with the user, in which the user will perform some form of action, such as a clicking on part of the screen, or providing an input such as text. Afterwards, the system responds to these actions with an appropriate reaction. Ideally, as more functionalities are desired for the system, both new categories will be created and the UI Requirements will grow in proportion to the number of new actions that are needed to support these categories.

NFRs also had their own categories too:

Performance Requirements – These requirements were briefly mentioned during the elicitation process. A category was created for them, but as of now it only contains one requirement. It is expected as the project expands, more will be added to this list.

Security Requirements – These requirements were also briefly mentioned, however there were a few important things to note with them. The first is that the system's server was anticipated to be

hosted at the university's campus, meaning that they would need clearance from the university's IT department first. Additionally, there was a need for a login system. As the system took the form of a server with a webpage, it was expected that we would have a number of different users all wanting to use the server to perform their own transcript editing. Thus, the need to maintain separate directories for each user while also ensuring they can only access their own directory added more security NFRs.

Maintenance Requirements – As of now, there is not a set plan for system maintenance. It is expected that Embry Riddle students will continue to maintain the system, so this was put down in this section as its own requirement. However, there are a number of desired features that need to be implemented. While these features already have requirements laid out in the Functional Requirement section, they still need to be implemented after the course of this GRP. Thus, anything related to this is also documented here.

Usability Requirements – The model of the system being hosted as a server and webpage was a desire of the customer to be able to develop and use the system without the need to develop it for multiple platforms. Thus, this requirement was logged as a usability requirement.

Accessibility Requirements – These requirements were briefly mentioned during the elicitation process. These currently are not high priority, but a category was created, as it is likely more accessibility requirements will be made to support different users, should the need arise.

User Documentation – These requirements give instruction to what kind of documentation must be established to be given to the users so that they may understand how to operate the system. The contents of this manual were described briefly during elicitation, but the requirements highlight more of the documentations' contents too.

Additionally, it is expected that there will be more categories for NFRs to be created, and for these categories to expand.

UML Class Model

See Appendix B for the model

The UML Class model follows the current implementation of the system. Each class and its interactions exist within the system as follows:

The app is the central part of the system. It acts as a connector and host for all of the objects of the system. It contains the entirety of the layout of the UI, and handles the distribution of information throughout the system. It separates the two main parts of the system: SALED and SALAC. SALED is focused being the primary view of the system, its interface. Meanwhile, SALAC contains a lot of the information of how data is structured, including the Annotation and Cues system we use to modularly store transcript information so that it can be properly displayed to the user.

The side bar, top bar, and transcript region are the various parts of the layout that exist. Some have graphs that need to be displayed, such as the waveforms and spectrogram. Each of these parts is imported to the app, so that it can be organized and displayed.

The callbacks class contains callback functions, which are the primary forms of interaction the user will have with the system. Each callback function performs some sort of action, and some of these actions require the use of SALAC functionalities, so this information must be passed through the app to access it. It has two other classes it uses, audio and File I/O, which are used to handle non-callback functionalities that are exclusively consequences of callback events occurring in the interface.

Through SALAC, there are a variety of functions that encapsulate the manipulation of files and data within the system. The large majority of these functions are methods that convert transcript files to objects within the system, and vice-versa.

In the future, more classes will be created to support the AI model. The current identified place for them would have a connection to the app, and it would have its own series of functions. They would not interact with SALAC or the callbacks directly. Rather, the instruction to utilize the AI engine will be triggered from a callback function, sent to the app class, which then executes the appropriate functions from the new class to interact with the AI engine. This way, the program is kept modular, and the connection to the AI engine is only through one point, which will only accept a transcript file as a return.

State Chart

See Appendix C for the model

The State Chart defines the ideal model for the system's actions as a final product. The primary functionalities that were highlighted are the creation of a project, the ability to load an existing project, the ability to create a transcript from an mp3 if the user does not already have a transcript, and the various functionalities of the running program, such as modifying the transcript, editing the intervals, and saving changes.

The current product has many of the functionalities prior to the "Running Program" state either implemented as functionalities connected to Running Program, or has the functionality occur separate from the system. This would be actions such as uploading an mp3 or generating a transcript. Other Actions such as creating and loading a project do not exist yet in any capacity. These functionalities are still in development, but in the system's requirements these things need to be handled by the system, which is why they are established in the model.

As the system continues to develop, it will approach this model's structure further, but in order to ensure quick and simple access to all of the functionalities for testing and observation purposes. Ideally, we want these functionalities to be as shown in the model, as it better connects to the stakeholder's needs as seen in the requirements. The model better enables the division of user personal directories, as you can implement a login feature that restricts what directory you have access to at the Start Node. Additionally, it allows for far better organization of the user's projects and data, and helps guide newer users.

Data Flow Diagram

See Appendix D for the model

The Data Flow Diagram captures the flow of information in the system. Primarily it was used to describe how information in the system would be generated and through what methods it was modified or displayed to the user. This view helped capture important things, such as the need for a database to store transcripts in on the server, as well as the things the user was allowed to edit in the system, such as the intervals or the transcript itself. While it could be considered that the intervals and transcript are the same, they were abstracted separately, as the intervals were largely visual and represented with colors, while the transcripts were just the text within each interval and encapsulated all intervals, regardless of their size.

This model also illustrates the idea of having the transcripts be generated in system. While this may change in the future, it was still proposed here to be safe. If it does not come to fruition, then the transcript data and audio data will simply be uploaded directly by the user.

Additionally, it was important that the Server Owner was placed as a sink in the diagram. This is because of the tool's primary use being as a tool for students. The Server Owner ideally is the customer in this case, and they had requested to be able to have access to all of the student's transcripts that were being stored within the database, and so the model reflects this.

Architecture

See Appendix E for the model

The architecture was structured around two major patterns, the Model-View-Controller (MVC) pattern and the N-tiered architecture pattern. The MVC pattern was chosen first, as it was most applicable to the system. Observing the previous models, as well as the elicited information, it was very clear that between the interface and the methods needed to setup our data structure that the structure would largely follow this pattern. The MVC pattern is established through the parts "Interface", "SALED" and "SALAC". In the architecture, the "Interface" piece acts as the view, giving the user a way to interact with the system and view the information the system is displaying. It displays all information to the user, such as the transcript they've uploaded and the waveforms of the audio. The model is the "SALAC" piece, as it contains the data structure of our Cues and Annotations, as well as the various methods to manipulate these structures. This includes manipulating file types, and converting from a file to a usable object in the system and vice versa. Finally, the controller is the "SALED" piece, as it acts as the central controller of the system, organizing commands received from the user into changes to the interface's view or the data itself in the model. This includes receiving transcripts to be turned into Annotations, passing edits to the transcript to the model, and updating the view with new information.

The N-tiered architecture was put into practice when considering the other pieces of the system. Currently, only the pieces described in the MVC pattern are actually implemented. In the future, the other pieces (and maybe more) will be established, so this model is attempting to capture how they should be organized in context to the current system. This model was chosen second because of the considerations taken on how the system should be expanded. In an N-tiered architecture, each tier can only communicate with its next immediate tiers, and the user will only be able to communicate with the topmost tier. The reason for this is to maintain the abstraction of data, to limit the amount of data each tier can/should be able to access. For the system, this means making the Interface the only tier the user will be able to interact with. We do not want the user to interact directly with the other pieces of the system, all of their commands should be routed through the Interface and sent down tiers as needed. Afterwards, the tiers are separated into the middle tier of "SALED", "SALAC", and "AI Model". And then finally the Database is at the end, in its own tier. This separation of tiers also prevents the user and Interface from directly interacting with the Database. Ideally, we do not want either of these to interact directly with the databases, as it can compromise the Security Requirements of our system, which want to maintain the partition between user profiles. Additionally, if more modules need to be implemented, all the engineer needs to do is determine which tier it belongs in, and what modules it needs to communicate with, making this type of structure easy to modify for the future.

Current State of Project

Currently, the primary parts of the project that exist deal with being able to load, edit, and save transcripts, uploading audio files to display their waveforms, and overlaying the intervals of the transcript on the waveform. It even allows for the playing of these audio files so that you can listen to the audio as you are editing. This allows the system to be fully functional as a transcript editor. Below are a series of figures that describe the functionalities of the system, using images of the running program as a guide.

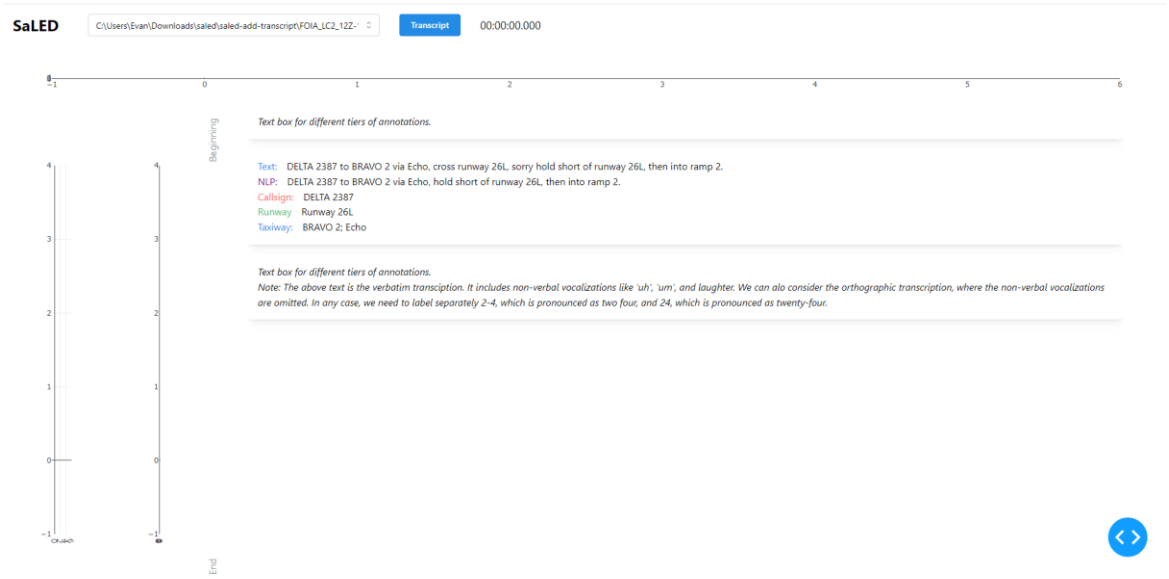


Figure 1 – SALED Launch

When the program is launched, it will have nothing on it beyond empty graphs and placeholder pieces. However, once the audio loads and a transcript is uploaded, the program will update to the following:

Figure 2 – SALED Running

There are many aspects of the system, so a breakdown is required:

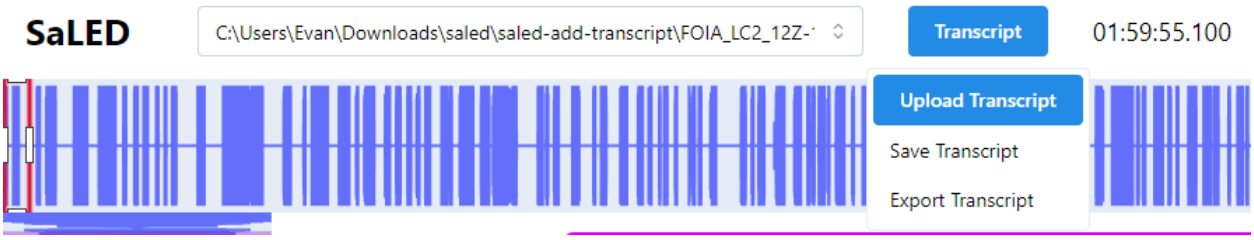


Figure 3 – The Top Bar

Beginning with the Top Bar, this primarily is here to display the global waveform, the large blue graph going horizontally across the screen. It also displays the audio file I am reading from in a selector, which I can change to switch between audio files within the directory I launched from. Next to the selector is the transcript button, which gives the options to either Upload, Save, or Export the transcript. Next to this there is the total time. This is the total time of the entire audio file, the one that is currently being used is about 2 hours long. Finally, there are the red brackets, which can be selected by left-clicking on a point on the graph, dragging, and then releasing left-mouse. It is also possible to drag each of the borders using the small white parts located on each side of the red brackets. These brackets dictate what is displayed in the Local Waveform.

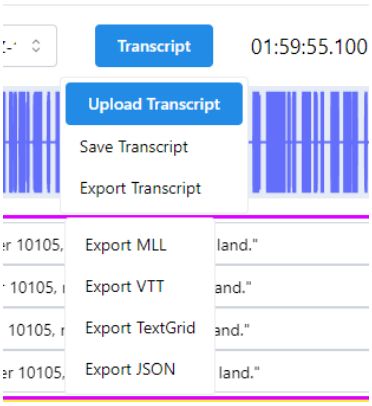


Figure 3.5 – Transcript Buttons. It is not visible, but the mouse is hovering over the Export Transcript option.

The options located in the transcript button all have different functions. The Upload Transcript button allows the user to pick an applicable file from their system and upload it to the system. It will then take that file and display its contents in the transcript section of the system. The Save Transcript button saves the transcript's current modifications to its original file. Finally, the Export Transcript button takes the current transcript, and exports it to an applicable file type.

Figure 4 – The Local Waveform and the Transcript. The red lines in the figure were added for clarity, they do not show in the actual program.

Next is the Local Waveform. The Local Waveform displays a closer view of the waveform as determined by the placement of the red brackets. It is vertical, and the left bracket corresponds to the top of the waveform, and the right bracket to the bottom. Each part of the Local Waveform is color-coded to align with the transcript. The transcript itself is shown to its immediate right. Each Cue is color coded one-to-one with the Local Waveform's audio data. The above figure illustrates this with red lines demonstrating which segment of the graph corresponds to which cue. In each Cue there are multiple tiers of annotation. There can be any number of tiers, it is entirely dependent on what kind of transcript is uploaded. For the example, 4 tiers are made, each of which can be edited in any way by left-clicking on the appropriate tier and typing.

Figure 5- The Spectrogram

The spectrogram is located in between the Local Waveform and the transcript. It is disabled by default, as it takes a long time to load depending on its size. However, there is a button that can be pressed to enable it and load the image of the Local Waveform's spectrogram.

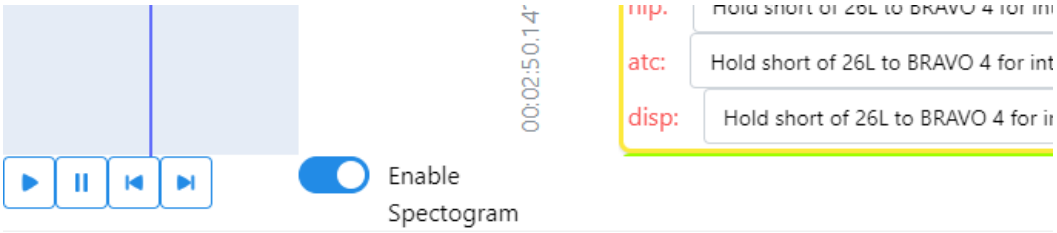


Figure 6 – Bottom Buttons

Directly next to this button are the buttons to play and pause the audio of the system. These are located at the bottom of the screen. This feature still needs to be expanded further, so it currently is a bit primitive, only playing and pausing the audio.

Conclusion and Future Planning

Overall, this project was able to accomplish many of the things it set out to do in its first few months. It was able to establish a working transcript editor that could manage both audio files and modify and save transcripts. Although there are many pieces that still need to be implemented in the future, the project met its goals of having a working product. More than this, the objective of leaving behind software artifacts, such as Requirements, Design, and other documentation fulfilled my goals as a Software Engineer. Before we began this project, it was understood that we may not have the time given our resources to be able to finish every single aspect of the system, as the system is intended to be continued beyond just a single GRP. In this scenario, it is crucial as a Software Engineer that I provide all future developers the tools and knowledge needed to be able to pick up the work that our team had started on and be able to continue with minimum difficultly. With the artifacts that I have provided, I believe they are able to give a strong understanding of the system to allow future developers to continue the lifecycle.

The parts of the system that still need to be developed are as follows. Firstly, there needs to be a way to synchronize where the transcript is audibly being played with the waveform. As it currently is, the user cannot determine where in the local waveform the audio is playing from. The customer has described that they want a play bar in the form of a red line traversing the local waveform. While we have been able to make this work in some capacity, the libraries we are currently using don't immediately support it, so many changes have to be made to accommodate the new library. Second, the connection to the AI engine also needs to be established. Currently, users have to process their transcript through another software before uploading it to the system. Ideally we want this to happen as a function within our system, not analogous to it. Third, we need to establish the login system and standup the server. These things should be done parallel to each other, as the primary way of saving your transcripts to the server will be done by saving to that person's personal directory. So when a person logs into the server, they will only be able to access their personal files. Essentially, in order to implement the login system, we need the server to be stood up first. Finally, when all of this is complete, the next step is to add more

convenience to the users, thorough hotkeys and performance improvements. There is a major performance bottleneck currently, due to the way the graphs are loading. There needs to be a singular large update to the Global Waveform, and then many updates to the Local Waveform depending on how many times the user moves it. Additionally, the spectrogram also updates constantly upon being moved. Because of the large amounts of calculations and visual updates, it can make the system lag or even unresponsive. We have taken steps to accommodate this by disabling the spectrogram at launch, but more steps need to be taken to improve real-time factors.

References

Boersma & Weenink (1991) Praat [Computer Software].

Caroll, John (2012) *Agile Project Management in easy steps*, In Easy Steps.

Karim, Evan (2024) *SaLEd Software Requirements Specification*, ERAU. Unpublished.

Karim, Evan (2024) SaLEd UML Class Model, ERAU. [Image] Unpublished.

Karim, Evan (2024) SaLEd DFD Model, ERAU. [Image] Unpublished.

Karim, Evan (2024) SaLEd State Chart, ERAU. [Image] Unpublished.

Karim, Evan (2024) SaLEd Architectural Model, ERAU. [Image] Unpublished.

Liu, J. (2024) 'Cheatsheet of Plotly Dash', ERAU. Unpublished.

Liu, J. (2024) 'Lctr 7a. Ideas of Class Projects', ERAU. Unpublished.

Liu, J. (2024) Github Repository of the SALED Software and its associated systems https://github.com/orgs/eraus-projs/teams/saled/repositories

Liu, J. (2024) SALAC, ERAU. [Computer Software].

Liu, J. (2024) SALED, ERAU. [Computer Software].

Ochoa, O. (2023) 'Achitectural Documentation', ERAU. Unpublished.

Ochoa, O. (2023) 'Verification and Validation of Requirements Specifications', ERAU. Unpublished.

Roques, A. (2009) PlantUML [Computer Software].

Appendix A

Requirements (Derived from []):

1. Overall Description

The project itself is the development of a new software tool, a tool to edit transcripts generated from SaLAI models.

2. Constraints

2.1 The system shall be created using Python.

2.1.1 More specifically, the system shall utilize the Plotly Dash series of libraries.

2.2 The system shall not be ran locally, solely on a server.

2.3 The system's user functionalities can only be accessed by connecting to the server's webpage.

3. System Environment

3.1 The system shall be hosted on a server located in Embry-Riddle Aeronautical University.

3.1.1 The system shall have Embry-Riddle Aeronautical University's IT department approve of the server's hosting.

3.2 The system shall allow users to connect to the server remotely.

4. Functional Requirements

4.1 General Requirements

4.1.1 The system shall allow the user to upload an audio file, pass it to an AI engine, and have the model produce a transcript file.

4.1.1.1 This transcript file shall be stored in the system's database, under the user's personal directory.

4.2 UI Requirements

4.2.1 The system shall display the Global Waveform at the top of the webpage, horizontally from left to right.

4.2.2 The system shall display the Local Waveform at the leftmost side of the webpage, vertically from top to bottom. The display starts under the Global Waveform, shall be perpendicular, and does not intersect it.

4.2.2.1 The Local Waveform shall be a display of a segment of the Global Waveform. This segment is denoted by a set of brackets the user can adjust on the webpage.

4.2.3 The spectrogram shall be displayed at the leftmost side of the webpage, parallel and to the right of the Local Waveform, vertically going from top to bottom.

4.2.3.1 The spectrogram shall reflect only the data that is currently being viewed in the Local Waveform at all times.

4.2.4 While the user is playing audio, the system shall prevent all interactions with the UI except for the modification of the transcript.

4.3 Database Requirements

4.3.1 The system shall save all uploaded transcripts to the user's personal folder on the server.

4.4 File I/O Requirements

4.4.1 The user shall be able to upload transcripts to the system through a button on the webpage's UI. Uploaded transcripts must be either vtt, srt, or mll files.

4.4.2 The user shall be able to upload audio files to the system through a dropdown menu on the webpage's UI. The audio file must be an mp3 or mp4 file.

5. Non-Functional Requirements

5.1 Performance Requirements

5.1.1 The webpage shall be able to run with a maximum RAM of 16GB.

5.2 Security Requirements

5.2.1 The system shall require users to login to the system with a unique username and a password.

5.2.2 New users shall be able to register with a unique username, and whichever password they desire.

5.2.3 The system shall prevent users from being able to access files outside of the ones inside of their personal directory/folder.

5.2.4 The system shall be isolated from any other systems at Embry Riddle, beyond the users that connect to its webpage.

5.3 Maintenance Requirements

5.3.1 The system shall be maintained by Embry Riddle Software Engineering students.

5.3.2 The system shall be easy to modify and add features.

5.3.3 The system shall be able to easily incorporate more and newer AI engines as necessary.

5.3.4 The design of new features for the system shall be kept as modular as possible, to ensure high cohesion between functionalities and reduce coupling of features.

5.4 Usability Requirements

5.4.1 The system shall be able to be ran on all Operating Systems, via webpage connection.

5.5 Accessibility Requirements

5.5.1 The system will have hotkeys to the following functionalities: splitting an interval, combining an interval, playing the audio, stopping the playing of the audio, undoing transcript modifications, redoing transcript modifications, saving the transcript and opening a new transcript.

5.6 User Documentation

5.6.1 The system shall come with a User Manual, that describes the following:

5.6.1.1 Table of Contents

5.6.1.2 Short Introduction – small introduction to the system and what it is meant to do.

5.6.1.3 Capabilities of the AI engines provided (pros/cons) – Describes the pros and cons of the base models that are provided with the system.

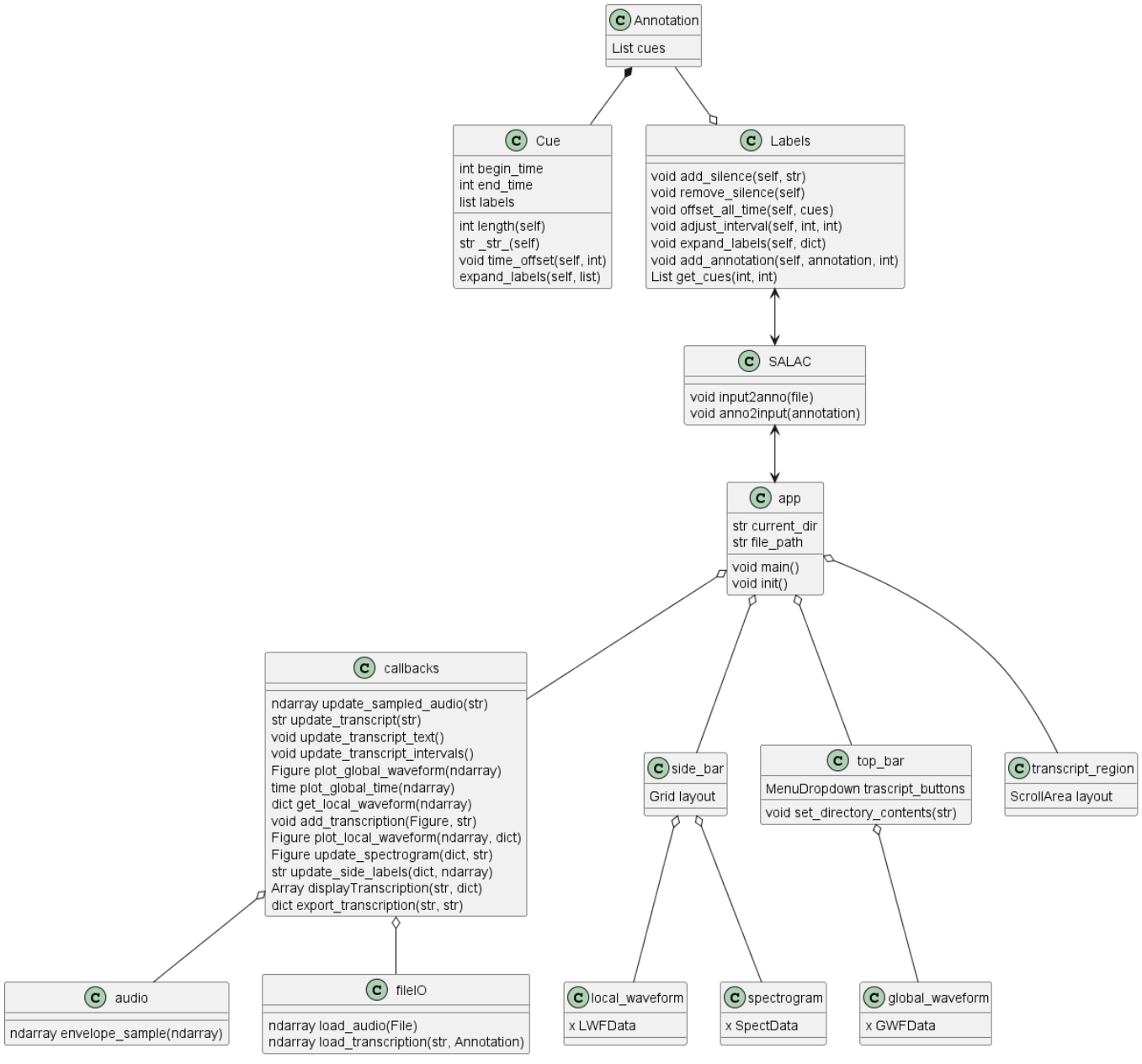
5.6.1.4 How to upload files – How to upload your files to the server so that you can use them in the software.

5.6.1.5 How to convert files – How to convert the file type into something that can be used, in the event the transcript was generated into a strange file type.

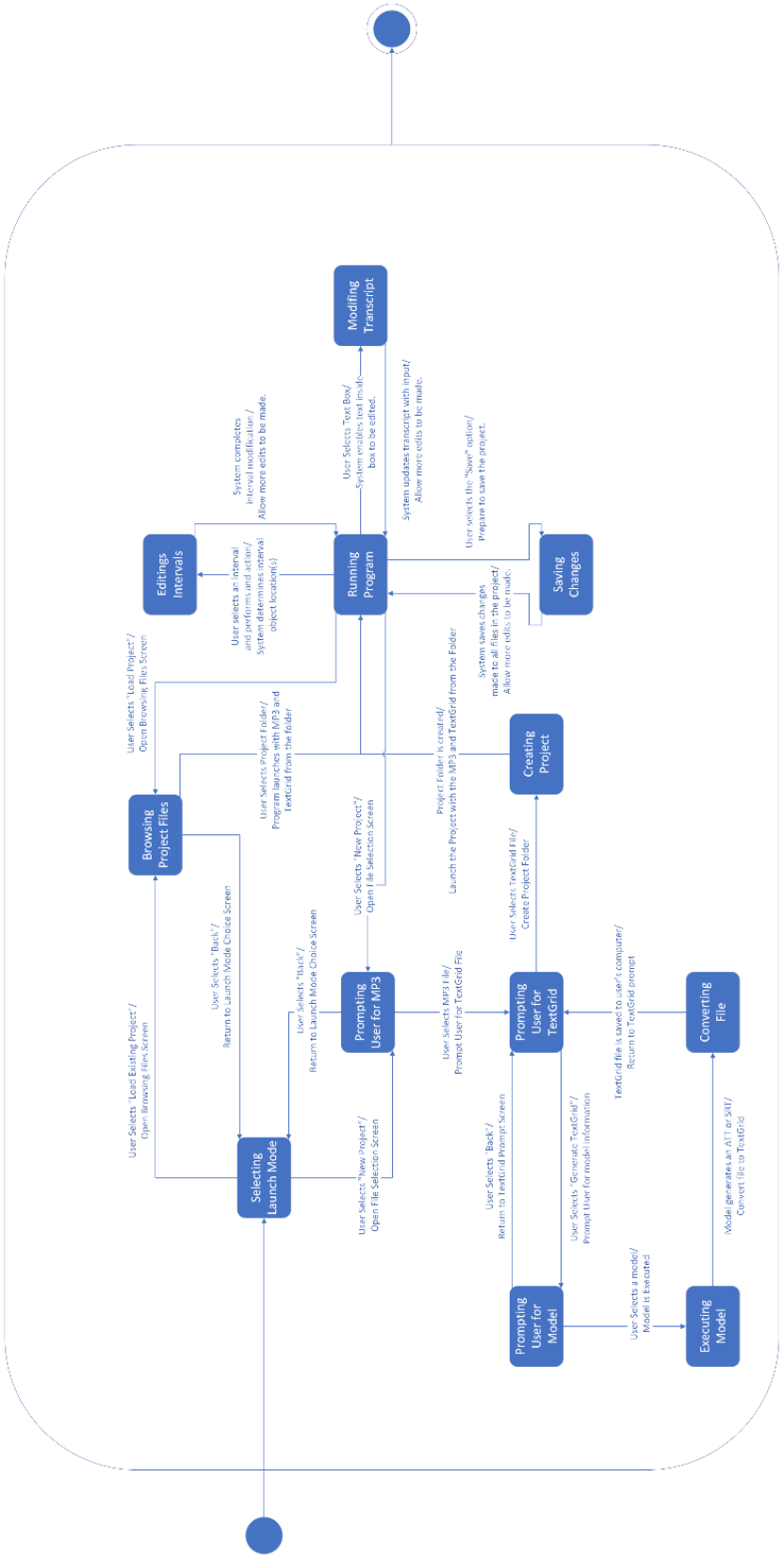
5.6.1.6 How to edit the transcripts – the steps and functionalities associated with modifying a transcript.

5.6.1.7 FAQ Section – A small set of frequently asked questions.

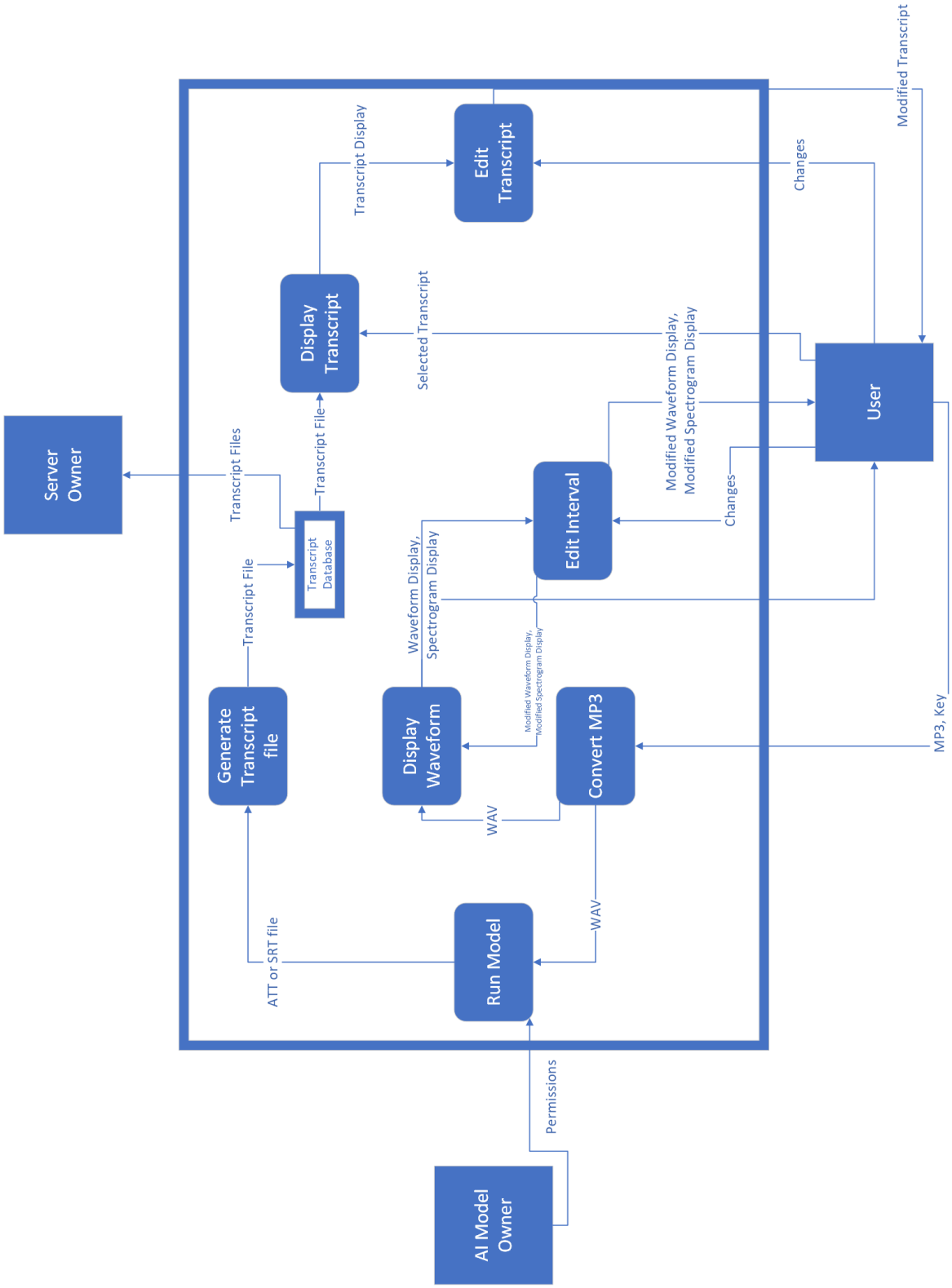
Appendix B – UML Model



Appendix C – State Chart



Appendix D – Data Flow Diagram



Appendix E - Architecture