Software Design Document

March 31, 2021

Tabs2XML

Alp Sirek
Andrew Ngov
Arjit Johar
Daniel Santorelli
Muhammad Azizi

Submitted in Fulfillment of The Final of
EECS 2311 Software Development Project

Table Of Contents

Introduction

Tabs2XML was created with the intention of aiding digital music enthusiasts by enabling conversion of Tablature (of guitars/drums in regular/bass) music to MusicXML format. By doing this, users will be able to take advantage of this new format type's versatility and wide array of compatibility with third party software.

The program accepts an inputted tablature formatted song. It does this when the user either inputs text into the provided text box, or by clicking the "Browse" button (when the latter method is used, the user can then select any text file in the chosen folder via the left-side list). The user inputs the piece's name and time signature in their specified areas of the window, and then the user chooses a location for the output file to be saved. Finally, the program converts the tablature to MusicXML and saves it in the specified location. The program automatically identifies the tablature type of the inputted text (Guitar, Bass or Drums) when the conversion begins.

Program Overview

- Software used to build this program (languages, libraries and frameworks) include:
    - Pip ---> package managers to help install and import needed packages
    - SimpleGUI → allowed us to build a front GUI in Python
    - Shutil ---> File copier
    - Lxml → Interface to simplify handling of the XMl
    - Numpy ---> array to matrix and matrix transformations

Multiple frameworks and libraries are utilised by the program so that it can function. The program is written in Python because of its simplicity and its support for libraries and frameworks. The framework on which Tabs2XML is based is SimpleGUI. This framework was chosen because it provides the program with a visually appealing interface which the user can interact with. In terms of libraries, Pip, Shutil, Lxml, and NumPy are utilised. Pip was used as the package manager for the project because it gave the team a simple way to add libraries and frameworks to the program. The use of Shutil allowed for our program to copy file content from one file to another in an effortless manner. Lxml was used because it adds extra features that allows us to generate XML files more easily. Lastly, NumPy was used to convert the input file to a matrix to apply matrix functions on it, such as transposing it to “flip” the input file to allow us to better handle the input.

Front End

- Information needed from the user includes the directory, file to convert, name of the piece, time signature of the piece and the directory to save the output file to.
    - After choosing an input directory, the user can then choose the file they want to convert, which is displayed to the user.

    - The name of the piece is then appended to the XML file (default is Classical Guitar if the user does not edit it)
    - After choosing the save directory, the converted output file is saved there.
- The save directory, file name, time signature and text are saved in main.py as variables.
    - The text from the chosen text file (should be tablature, or an error occurs), time signature, and piece name that is stored within main.py is then passed onto the xmlConverter as parameters (passed on to the back end - explained below), where it is then converted and written as a XML file.
- When the file has finished being converted, it is saved to the save directory that was stored in main.py.

Back End

The conversion engine is built into a single class with the main focus of creating a MusicXML file to output. The conversion engine does not require other objects in order to function properly.

The main function/method in the conversion engine is known as “noteArrayMaker“. The input is a list of lists containing each character in every column of the original tablature. The function creates an empty list to be finally returned at the very end of the program. The main conversion/parsing is done via a nested loop which first goes through a column and the characters inside the list.

The nested loops’ first loop also creates a temporary Measure object.

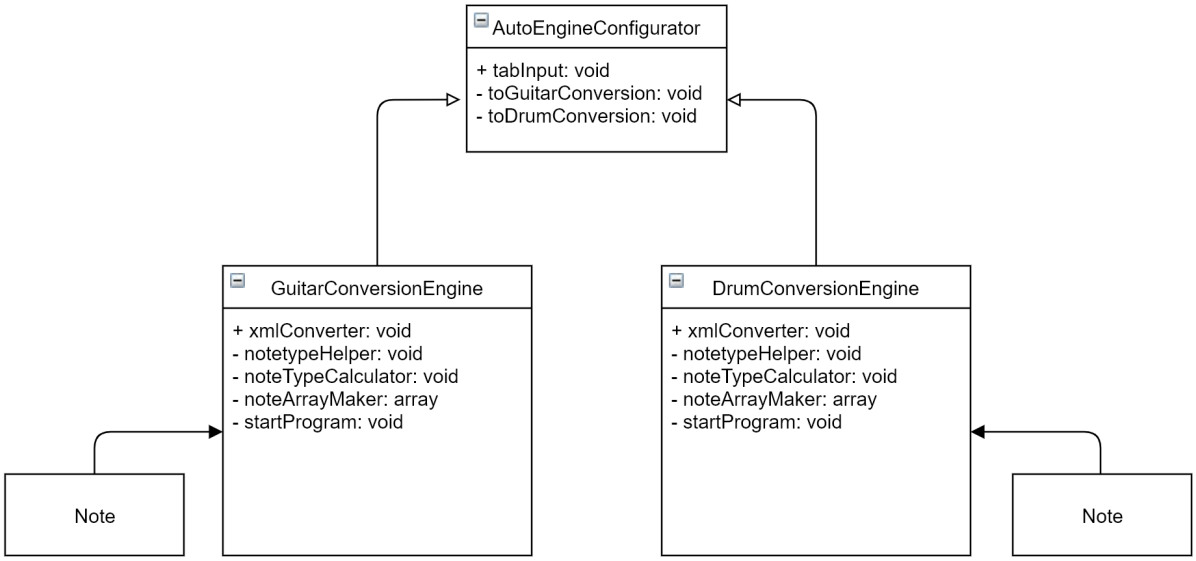
When the parsing is done for one part of the list, when a note is detected, a note object is created appropriately based on the information. The position of the note is also recorded to later calculate its type and duration. After a simple note object is created with the parsed information, attributes such as the octave, what note it is, and alter (if needed) are saved into the object using helper functions.

After all the parsing is done for one measure and a new one is found by the parser, a new Measure object is created. This is after adding the temporary list of notes inside the Measure object, and then adding that to the main list that was created in the beginning of the whole function/method. This is done until the end of the inputted text is reached.

After all the parsing is done, the list containing Measure objects(the position of the notes and the notes itself) is inputted into the noteTypeCalculator function. This calculates the types and durations of all the notes contained in the main list based on the saved positions and length of each measure. It assigns the note objects the correct types and durations it individually calculates. By changing the attributes of the notes in the list, the list can be reused for the xml conversion since the list exists somewhere else in the memory. The function returns the first created list in the function. The startProgram function is then called - the input being what was returned from noteArrayMaker function. This function makes a new MusicXML file and writes in it based on the measure objects (which contain the notes and their attributes).

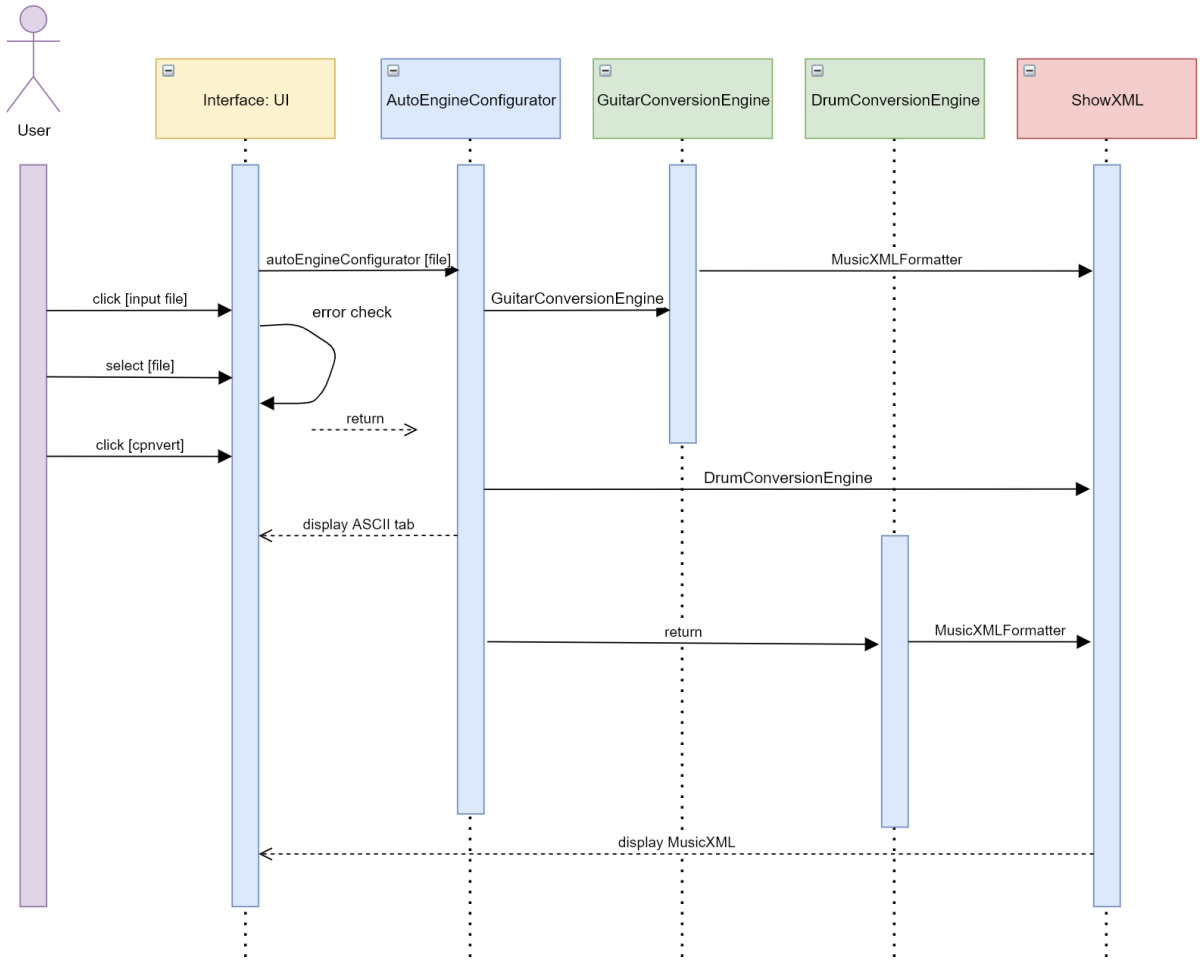
Class Diagram

NOTE: Our program is written in Python and guitar/drum conversion engines are built into one class. Explanation of how these functions work is provided in the program overview for the back end above. Also note that the interactions are identical for GuitarConversionEngine and BassConversionEngine (both are represented as GuitarConversionEngine).



Sequence Diagram

Sequence diagram for the whole program (shows the process of the program overtime). Explanation of the parts of the sequence diagram is shown in the program overview above. Note that the interactions are identical for GuitarConversionEngine and BassConversionEngine (both are represented as GuitarConversionEngine).



Maintenance Scenarios (Adding Extra Features)

Changes to Features That Are Not in the Note List

Vibrato, etc.

To add features like vibrato for the guitar tablatures, you would need to define the vibrato, downstroke or upstroke inside class “Note” which is located in “xmlConverter” at the top of conversionEngine.py (from line 10). Define vibrato as a boolean. Before the function noteArrayMaker (line 286), create a function that reads the line before the note array to determine whether it will be played in vibrato or not.

Upstroke, Downstroke, Palm Mute etc.

To add downstroke, upstroke and palm mute for guitar tablatures, you would need to define the upstroke, downstroke and palm mute inside class “Note” which is located in “xmlConverter” at the top of conversionEngine.py (from line 10). Before the function noteArrayMaker (line 286), create a function that reads the line before the note array and stores the results in an array. In noteArrayMaker, add a line to append whether upstroke, downstroke, or palm mutes occur to the final MusicXML file.

Changes To Features That Are in the Note List

Hammer-on, Pull-offs, Slide, etc.
To add hammer-ons, pull-offs, and slides, define the hammer-on and pull-offs inside class Note located in xmlConverter at the top of conversionEngine.py (from line 10). Inside noteArrayMaker, add an algorithm to determine slides, hammer-on and pull-offs (ex. Search for p, h, or s which signify pull-off, hammer-on, or slide) and then append to the final MusicXML file.