Functional Specification

for

Music Transcriber

by

JERZY BARAN

13307086

23/11/2016

Table of Contents

1. Introduction

1.1 Overview

Music Transcriber is a desktop application aiming to help a musician transcribe music from a binary music file to some kind of notation, be it sheet music or MIDI.

There are times when I would like to learn to play a particular piece of music, but I can’t find any sheet music for it on the internet. With a little patience, I could transcribe the MP3 I found on the internet using this application.

The program helps by slowing down the music and analysing which notes are being played at any given time, offloading the work for the human. Music Transcriber will **not** transcribe the music all by itself. The user still needs to be somewhat competent in the field of music.

Its main functions are:

- Slow down the music without altering its pitch.
- Show the audio in a form of a waveform.
- Analyse the current portion of audio for currently played notes.
- Attempt to find the key of the music.
- Attempt to find the tempo of the music.
- Allow the user to filter out certain frequencies.

1.2 Business Context

Music Transcriber is a personal project and business context is not applicable.

1.3 Glossary

Sample	A single unit corresponding to a “change in sound”. Usually a byte or a short for monophonic sound and 2 bytes/shorts for stereophonic sound.
Window	A collection of samples. Usually between 512 and 4096 samples. Fast Fourier Transform requires it to be a power of 2 size. It transforms an otherwise continuous signal into finite collections of data which can be analysed.
Time domain	When the audio is represented in terms of time and amplitude, ie. A common waveform visualisation.
Frequency domain	When the audio is represented in terms of frequency and amplitude, ie. A spectrogram.

2. General Description

2.1 Product / System Functions

By using the Fourier Transform, the application will be able to decompose a window of audio into its fundamental frequencies by converting the audio into the frequency domain. Those frequencies will then be mapped onto a piano keyboard and shown to the user. Using this feature, the user receives help with identifying which notes are being played. They can then take that information, and alter it to their needs, for example simplify a chord into a single note or maybe change it into arpergio.

Sometimes the melody is just too fast. To help the user find the right rythm and notes, Music Transcriber will stretch the audio, making it seem slower, without altering the pitch. The program will achieve that by converting the audio into frequency domain, interpolating the data and converting it back into time domain. This way the pitch of the music will not change and the user will hear the exact same notes only slower.

Another useful feature is the equalizer. EQ is an audio filter which can silence selected frequencies while leaving others intact. This is especially useful when the user wants to focus on a particular part of the song (eg. vocals), but other instruments are polluting the frequency analysis.

When the user wants to identify the tempo of the music, they have two options. They can place special markers on the waveform visualisation, showing the beats of the music (these are usually easily identifiable by a large spike where a drum kick or a hat occurs). After a sufficient amount of those has been placed, the application will use the information it posseses about the audio (the sample rate of the file and number of samples per pixel on the waveform) to calculate the tempo of the music. Alternatively, the user will be presented with a widget containing a button. They will be asked to click it to the beat, and after a while, the average tempo will be approximated.

A MIDI note editor will allow the user to quickly note down the music they hear, without leaving the application’s workspace. They will also be able to connect a MIDI keyboard to the computer in order to record what they play.

2.2 User Characteristics and Objectives

The application will not require a large amount of technical expertise in order to be usable. Every option will have a tooltip explaining what it does. However, the user will be expected to at least be familiar with music theory and its jargon.

The application’s aim is not to transcribe a music piece for somebody, but to aid them in their creative process. That means the application is not much use if the users have no clue what they are doing music-wise.

2.3 Operational Scenarios

The user wants to record a short piece on his MIDI keyboard.

1. The user plugs in the keyboard via USB cable.
2. The user launches the application. If the user has used the keyboard with Music Transcriber before, all the settings will be remembered. Otherwise, the user open options, selects the keyboard from a dropdown list. He also selects the soundfont he wants to use and then saves the settings.
3. He presses Record on the main window.
4. When he presses the first key, the application starts recording all key strokes.
5. When the user is done, he presses Record again to stop recording. The notes show up in the editor. He does not want to edit anything, so he simply saves them to the MIDI file.

The user wants to transcribe music without using the built-in editor (he prefers traditional sheet music notation).

1. The user launches the application.
2. He clicks *File -> Import -> Audio* and selects the wav/mp3/ogg file he wishes to work on.
3. The file gets loaded and the waveform is filled with data.
4. He clicks on the waveform where the music begins and selects speed multiplier of x0.25.
5. He sets a loop marker for the first 2-4 beats and listens to them a couple of times.
6. When he’s ready, he pauses the playback on each note and writes down on the music notepad the notes the virtual keyboard shows him.
7. If the user gets tired or has worked enough for the day, he saves the project. When he comes back to it tomorrow, the audio file and all the markers will be loaded and he will be able to resume from where he left off.

The user wants to compose some music in the editor.

1. The user opens the application.
2. He decides on which key the music is going to be in and selects it from the dropdown. The editor automatically highlights all the keys which are contained in the scale the user chose.
3. The user selects the tempo of the track and the size of the grid. He starts putting down some notes.
4. After he’s done, he saves it to a file.

2.4 Constraints

Computations

The application will make use of a very computationally heavy algorithm called Fast Fourier Transform (which is a derivative of a Discrete Fourier Transform).

It will have to perform various operations (analysis, stretching, filtering) on an array of thousands of elements in less than 32ms. This will require careful architecture and good optimisations. This is the reason I chose to go with C++.

Audio Analysis Limitations

An audio wave is a collection of different waves generated by all the instruments in the music. All of those waves overlap each other and often are difficult to separate. Because of that, the application will not work as well with pop songs as it will with piano concert, for example.

Another problem is that no frequency is born equal. A fixed-size window will contain considerably less lower frequency cycles than it does higher. This results in lower frequency analysis to be much less accurate (typically anything below middle C is difficult to analyse). One way to fix that is to increase the size of the window, but then another problem arises. The bigger the window, the less accurate the analysis is in terms of time.

If the window is a second's worth of samples big (that is a very big and inpractical size), the analyser could miss a note that lasts for a split second, especially if its a higher note, because its frequency cycle is much shorter.

A right balance needs to be found for the size of the window to allow both for low frequency analysis and to mitigate the loss of accuracy in time.

3. Functional Requirements

Description	Loading WAV, MP3 and OGG audio files.
Criticality	High
Technical Issues	MP3 standard is patented.
Dependencies	

Description	Displaying the audio data in the form of a waveform.
Criticality	High
Technical Issues	
Dependencies	Depends on the ability to load audio files.

Description	Ability to play audio.
Criticality	High
Dependencies	Depends on the ability to load audio files.

Description	Showing which notes are being played (frequency analysis)
Criticality	High
Technical Issues	FFT is expensive but is going to be used for many of the features multiple times per frame.
Dependencies	Depends on the virtual keyboard to display the played keys. Depends on audio playback.

Description	Slowing down the music.
Criticality	High
Technical Issues	Again, dependant on FFT which is expensive.
Dependencies	Depends on audio playback.

Description	Saving current project.
Criticality	High
Technical Issues	
Dependencies	Depends on markers placed on the waveform. Depends on loading audio file.

Description	Changing settings for processing algorithms.
Criticality	High
Technical Issues	
Dependencies	Depends on frequency analysis. Depends on slowing down of music.

Description	An equaliser
Criticality	Normal
Technical Issues	
Dependencies	Depends on audio playback

Description	Placing markers on the waveform.
Criticality	Normal
Technical Issues	
Dependencies	Ability to load audio files.

Description	Detecting tempo.
Criticality	Normal
Technical Issues	
Dependencies	Depends on ability to place markers.

Description	Detecting music key.
Criticality	Normal
Technical Issues	Frequency analysis is not very accurate and in some cases the results may contain too much noise to be useable for analysis.
Dependencies	Depends on audio playback

Description	Setting audio looping of a small fragment.
Criticality	Normal
Technical Issues	
Dependencies	Depends on the ability to place markers.

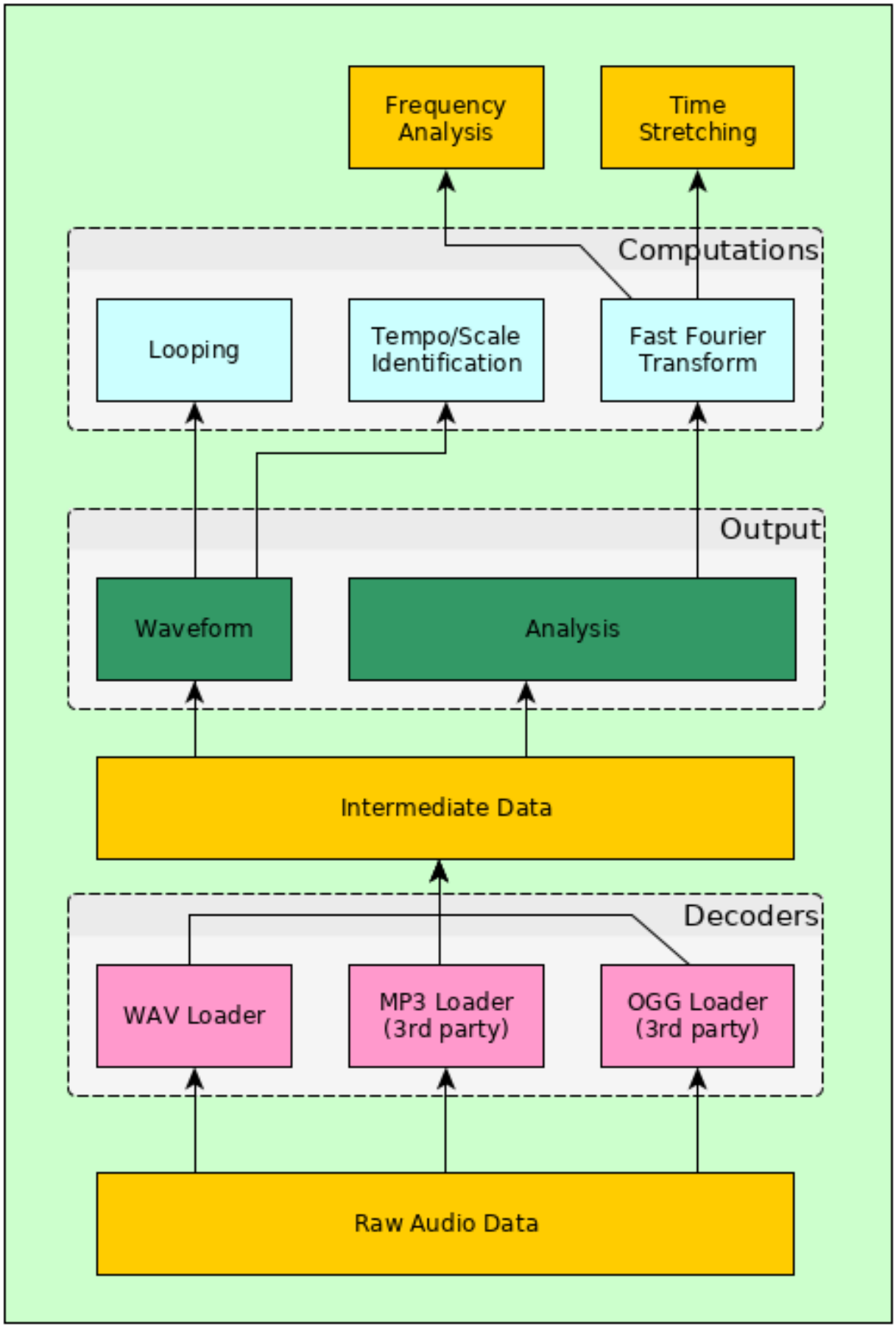
Description	Ability to put down notes in the editor.
Criticality	Low
Technical Issues	
Dependencies	

Description	Saving/loading MIDI files.
Criticality	Low
Technical Issues	
Dependencies	Depends on the ability to create notes in the editor.

Description	Connecting the software with MIDI keyboard.
Criticality	Low
Technical Issues	
Dependencies	Depends on the virtual keyboard.

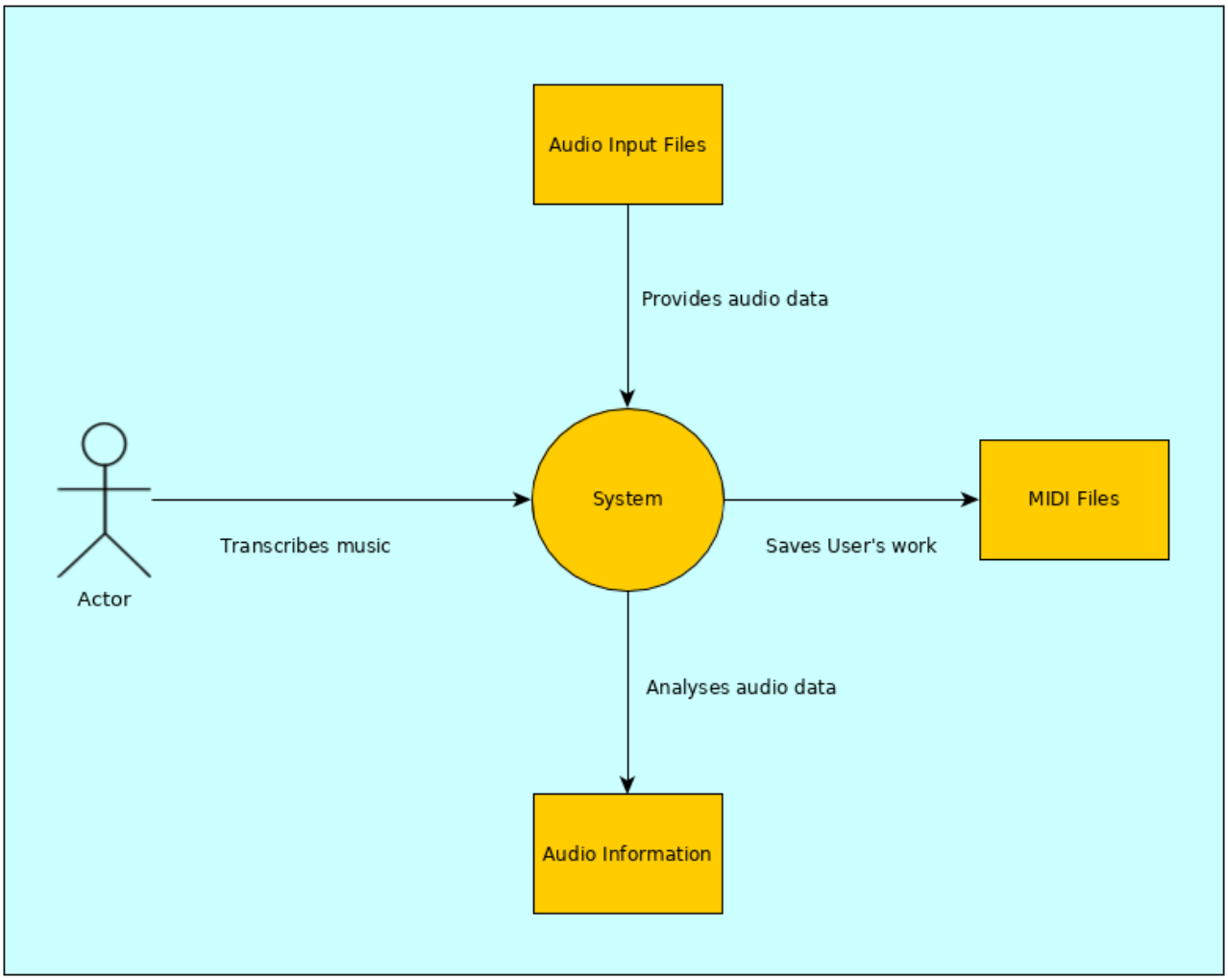
Description	Recording the MIDI keyboard output.
Criticality	Low
Technical Issues	MIDI latency is big if you don’t use special non-free audio drivers.
Dependencies	Depends on the ability to plug in the MIIDI keyboard. Depends on the ability to save MIDI files. Depends on the ability to create notes in the editor.

4. System Architecture

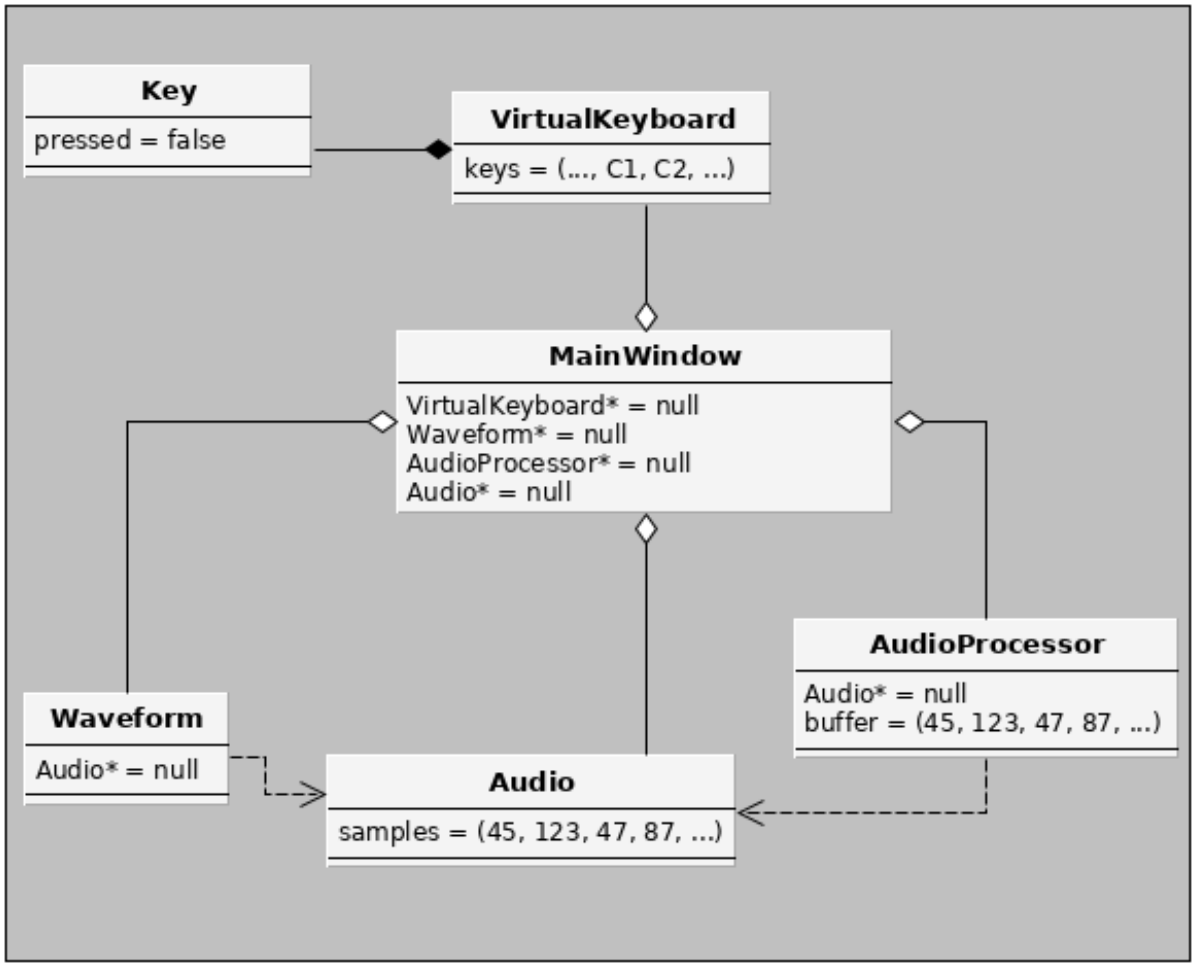


5. High-Level Design

System Context Diagram



Object Diagram



6. Preliminary Schedule

	Task Name	Start date	End date	December 2016	January 2017	February 2017	March 2017	April 2017	May 2017
1	Research	1/12/16	1/1/17						
2	Get audio loaded from file	1/1/17	7/1/17						
3	Display audio on the waveform	8/1/17	14/1/17						
4	Get a virtual keyboard set up	15/1/17	21/1/17						
5	Frequency analysis	22/1/17	7/2/17						
6	Equiliser	8/2/17	21/2/17						
7	Waveform markers	22/2/17	1/3/17						
8	Tempo / Key detection	1/3/17	14/3/17						
9	Music slow-down	15/3/17	14/4/17						
10	MIDI keyboard support	15/4/17	21/4/17						
11	MIDI note editor	22/4/17	14/5/17						
12	Debugging / polish	15/5/17	1/6/17						

Hardware

- MIDI Keyboard (only if I get to task 10 and 11)
    - USB type A to USB type B cable
- Laptop

Software

- IDE (QtCreator)
- Software synthesizer (if I get to support MIDI keyboards)

7. Appendices

- What is an equaliser?