Middle East Technical University

Department of Computer Engineering

CENG491 Computer Engineering Design

Fall 2011

Software Design Document for

DIGIMUSE

Written by Group GOBIT

M. Burhan Şentürk

M. Yiğit Yıldırım

Kamila Kuchalieva

Ezgi Berberoğlu

Contents

List of Figures

1 Introduction

Detailed Design Report for DigiMuse contains the detailed design descriptions of “DigiMuse” which is a mobile application to digitalize the printed sheet music using an internal camera and vocalize it by using highly standardized format MIDI. This document provides further definitions and explanations for actualizing the design of the software together with Software Requirement Specifications of DigiMuse [1] which explains the general overview, requirements and functionalities. The approach followed for writing this document is adapted from IEEE recommended practices [2]. This document also abides the standards presented [3].

The main aim of this document is to provide a guide to a design of DigiMuse which could enable any designer to easily implement the actual product.

1.1 Problem Definition

The main aim of this project is to enable the musicians to create their digital sheet music archive and play it easily with illustrations, tabs and vocalizing abilities with different instruments in a mobile device. DigiMuse offers these features both one by one or together upon need. It differs from the other similar products with its mobility, low-cost approach and answerability to multiple needs. DigiMuse is the first Android application as a Music OCR as well as a MIDI workstation. A review of existing solutions can be found in DigiMuse SRS under Section 1.4. User and Literature Survey [1].

Sheet music, which is known as a hand-written or printed form of musical symbols, is used by almost all the musicians. It serves a purpose to be a reference of musical partitions of musicians. Hence, whether professional or not most of the musicians make use of sheet music for studying their partitions. Most of the sheet music is stored as hardcopy since it has been used for decades. Unfortunately, maintenance of these can be very exhaustive and problematic. Mobility and usability of sheet music is another important issue especially for gig musicians. One other problem about sheet music is a method that studying directly from sheet music is a difficult and time consuming job for a new learner or a self-educated musician since it is like reading new language with new alphabet. DigiMuse’s main purpose is to offer a compact, user-friendly and a low-cost solution for these matters.

To start with, maintenance of sheet music can be very difficult for musicians. A musician uses different exercises from different books; also, he/she studies different partitions from different leaflets. Both storing and carrying all the necessary documents is a big problem. Digitalizing sheet music would be very compact and efficient solution for this and editing purposes, however, there are two main problems about it for current systems. Firstly, a musician should scan all the necessary sheet music archive with a scanner in order to make use of an accurate Music OCR program. It is not just an exhaustive work, it also costs so much. For example, one of the popular Music OCR program, SharpEye costs 149$ [4]. Another one, Capella-Scan costs 249$ [5]. Considering scanner will add a significant hardware cost, it would not be a cheap move. Another option is to buy already digitalized sheet music instead of using already owned hardcopies. Good transcriptions can be bought about 5$ for a single sheet music. Considering tons of sheet music, it would not be wise move either. Apart from the musicians, publishers and libraries also have giant printed sheet music archives. Carrying whole the archive and scanning one by one requires a lot of labor as this is not a compact solution.

DigiMuse offers a highly accurate internal Music OCR system which can be used with mobile devices operating on Android OS to come up with a solution for this issue. A user would just use a camera of the mobile device to take a photograph of the desired sheet music and DigiMuse converts it into MIDI which the most common digital music format in music industry then visualizes it as a digital sheet music. DigiMuse also provides editing functionalities both for correcting and editing. A user can save his/her documents as MIDI file and use them later thanks to the MIDI decoder/writer of DigiMuse. In a nutshell, by using DigiMuse musician would have a digital archive of his/her sheet music archive. He/she would also have the opportunity to convert any MIDI file to digital sheet music and add it to his/her archive. Since DigiMuse does not require additional hardware other than a usual Android phone or tablet, it would be a very efficient, compact and economic solution with its future under $10 price for any musician, librarian or publisher.

Secondly, as stated in previous paragraph, DigiMuse's MIDI functionality provides MIDI reading facilities in order to enable MIDI to digital sheet music conversion. Since MIDI is a very common format in music world, musicians can easily reach the MIDI file of a song they want to work on, so thanks to our project's MIDI functionality, the user can visualize a MIDI file in digital sheet music format or tablature format. This feature is very useful especially for gig musicians who need a quick transcription of a song they must learn in a limited time interval. Also, a collection of sheet music/tablature can be used by musicians performing on stage. Combining this with the project's Music OCR functionality the user will have the opportunity to make use of his/her sheet music/tablature library anywhere, anytime with a mobile device.

Last but not least, the nature of the sheet music does not allow self-educated musicians and new learners to study efficiently from sheet music. While schooled musicians take a lot of lessons for years just to read them instantaneously self-educated musicians spend so much time just imagine how their partition should sound-like and be played without caring details so do new learners. Also some of the musicians are not familiar with musical notes but tabs. When studying from an old method book or playing in a professional band, inability to read and use sheet music efficiently cause a huge disadvantage and time related problems for all previously mentioned these musicians.

To solve these issues, DigiMuse enables its users to vocalize and visualize their sheet music. A user can digitalize his/her desired sheet music document with using DigiMuse’s Music OCR facilities, after that Player of DigiMuse will vocalize the sheet music with desired instrument sound and also visualizes the sheet music in desired format i.e. digital sheet music or tablature simultaneously. It will also show the finger positions of the played notes on keyboard of a piano or a guitar. These features together would be a good instructor to new learners to understand how the partition is played in the keyboard and should sound like. Self-educated musicians can benefit from it similarly and save considerable time. DigiMuse eases the users’ life by making sheet music much more user-friendly and also it helps improving sheet music reading skills thanks to its instructive nature. DigiMuse also enables users to define specific tempo, key, octave, tonality and other musical features for studying with user’s setup and allows users to study in different speeds. This is the suggested method of music authorities.

All in all, our project focuses on digitalizing sheet music and uses it or other MIDI sources for self-learning purposes of a musician. It converts printed music to digital, plays MIDI with desired instruments and combine them with visual elements I.e. keyboard of the instrument and sheet music/tablature. The software will also provide a toolbox facility since it has key components for a musician such as tuner and metronome.

1.2 Purpose

This initial design report intends to provide complete description of all requirements of "DigiMuse". The descriptions suggested in this document will serve as a guideline throughout the development process of this project. The end-product will be tested against the requirements to ensure the quality of the software produced.

1.3 Scope

This document contains a complete description of the initial design of "DigiMuse". Its main intention is to provide software design description of the system according to Software Requirements Specifications of DigiMuse [1]. Specifically, this document covers the architectural design, data design, procedural design, design constraints, development schedule. In addition to that, software requirements, dependent libraries and working environment will be explained briefly.

1.4 Overview

This document covers all necessary information about DigiMuse. As it is explained in Section 2, a general view to the system and its modules are presented. These components are explained detailed later in Section 5. In section 3, design issues such as assumptions, dependencies and constraints, design goals and guidelines that are used is explained. Detailed information about Data Design and classes can be found in Section 4. DigiMuse's overall architecture; the architectural design of the software and detailed description of components are stated in Section 5. User interfaces are explained thoroughly in Section 6. Section 7 provides information the libraries and tools which will be used in whole development process of DigiMuse. In section 8, a Gannt chart explains the scheduling of the project. Finally, a conclusion to whole document is presented in Section 9.

1.5 Definitions and Abbreviations

CPU	Central Processing Unit
SRS	Software Requirements Specification
SDD	Software Detailed Design
MIDI	Musical Instrument Digital Interface
OS	Operating Sistem
OCR	Optical Music Recognition
IEEE	Institute of Electrical and Electronics Engineers
RAM	Random Access Memory
SD	Secure Digital
GUI	Graphical User Interface

1.6 References

[1] Software Requirements Specification Report of DigiMuse

[2] IEEE Std 830-1998: IEEE Recommended Practice for Software Requirements Specifications

[3] CSS.06 – Yazılım Gereksinimleri Belirtimi Standardı Rev.7.0

[4] http://www.music-scanning.com/

[5] http://www.capella-software.com/capella-scan.cfm

[6] http://developer.android.com/reference/android/media/JetPlayer.html

[7] http://opencv.willowgarage.com/wiki

[8] http://homepages.inf.ed.ac.uk/rbf/HIPR2/hough.htm

[9] http://www.inescporto.pt/~jsc/publications/journals/2010ARebeloIJDAR.pdf [6]

[10] http://opencv.willowgarage.com/wiki/

[11] http://en.wikipedia.org/wiki/Eclipse_(software)

[12] http://developer.android.com/sdk/ndk/overview.html

[13] http://developer.android.com/guide/developing/tools/index.html

2 System Overview

This product is mainly intended for users who are in some way or another involved in music. Potential users will be able to convert their sheet music into digital sheet music and play it easily on different instruments using their mobile phones. It also allows visualizing the sheet music with desired format or on keyboard of a guitar.

The project has 3 main components to achieve the specified purpose which are namely OMR, Midi Player and System Management Component. Detailed information about the components will be given in the following sections.

If we explain the usual pattern is briefly: First, DigiMuse converts a printed sheet music into a MIDI file using Image Processing and Pattern Recognition techniques. It takes the MIDI file as an input an generates digital sheet music/tablature. After that, the software provides real time simulation of playing the sheet music with sound of desired instrument(s), digital musical notes and keyboard of the instrument. The digitalized music will be stored locally on mobile devices memory. It can also be read from memory directly as a MIDI later on.

The product will be providing the set of functions in order to supply the user with the facilities explained throughout this document. We plan to keep functionality design as simple as possible for the user to use the application most effectively.

3 Design Considerations

3.1 Design Assumptions, Dependencies and Constraints

DigiMuse is going to be a mobile application. Therefore, in terms of the hardware requirements we are going to consider any mobile device which has to be supporting the Android OS. The mobile device is supposed to have the camera.

Since our product is going to be a mobile application, there will be a considerable factor of limitation in terms of the screen size. We are going to have four different screen options for users to choose from and decide on the most appropriate one to view. There will be one screen option where the user will view the screen with all displayable parts displayed, and there will be other three options where the user will view the combination of any two displayable parts. The user will be free to choose which two parts he/she wants to see on the screen in a larger scaled manner. Furthermore, for the sake of the efficient usage of the device screen we will avoid displaying any unnecessary information on the screen. The screen layout will be balanced in terms of the information content and size.

We may have to deal with the time constraint issues due to the fact that one of the main tasks that we are going to implement while developing DigiMuse is going to be the image processing. The CPUs on the mobile devices are not as fast as the ones on the computers for instance, therefore we will have to bring in some solutions or constraints in order to provide the reasonable time performance.

We know that modern mobile devices have cameras of quite high quality which allows the pictures taken buy them to be of large ratios. This is again an obstacle for us in terms of the mobile device's CPU capabilities and memory constraints. We will have to play with the size of the image before we perform the image processing task Thereby, we will put less efford in order to process the image.
Since the image processing task is quite complicated one when is comes to processing different note symbols, we may have a certain amount of performance loss in terms of correctness of the symbols been processed.

3.2 Design Goals and Guidelines

DigiMuse is going to be used by users who are interested in learning or playing musical notes. So our end-users will mostly use this software product as a tool which would help them to enhance their note reading /playing skills. Therefore, from the user's point of view the design must be simple enough to understand and use. Thereby, design simplicity is on of the most important design goals that we as the developers of the product care about. Users will enjoy the user friendly design of the product.

As already mentioned above, we may not be able to correctly process each note symbol all the time, so we are going to provide the user with feature of editing the notes been processed when it is needed to so.

4. Data Design

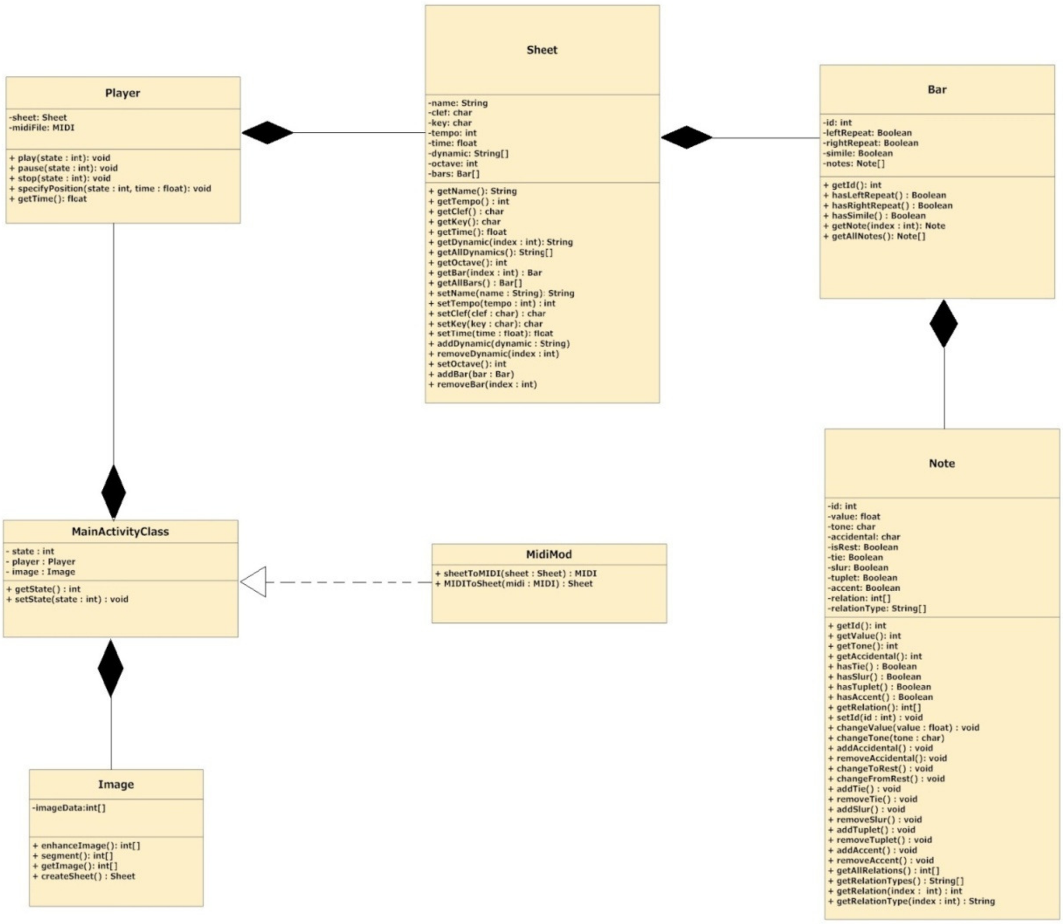


Figure 1. Simplified Class Diagram

4.1. Data Dictionary

4.1.1 MidiMod Class

This class provides full functionality for converting between MIDI format and sheet instance.

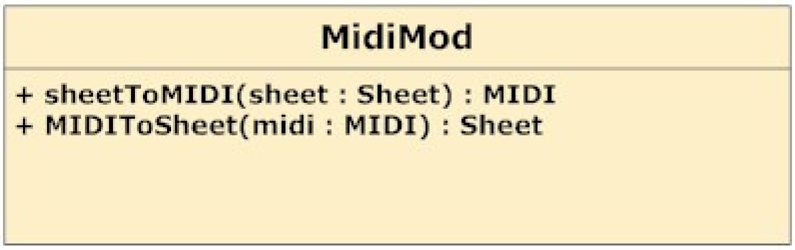


Figure 2. MidiMod Class Diagram

Attributes:

No attributes.

Methods:

sheetToMIDI: This method converts sheet object to a MIDI formatted file. This will provide us to save currently playing MIDI to SD card.

MIDIToSheet: This method converts MIDI formatted file to a Sheet object. Sheet object is used to visualize the MIDI file.

4.1.2 Image Class

This class performs image processing techniques with its methods on the image that is the photograph of the sheet music taken by the camera of the mobile device.

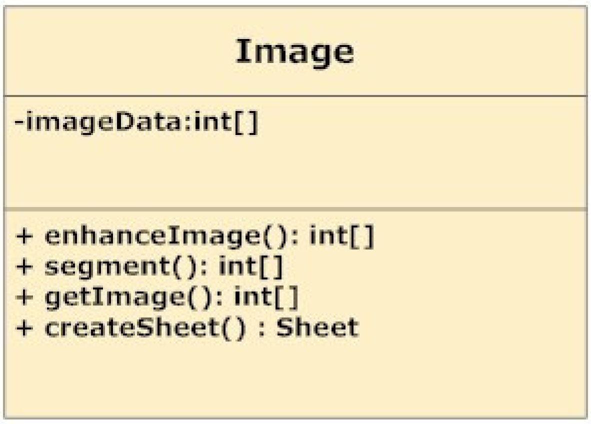


Figure 3. Image Class Diagram

Attributes:

imageData: Contains the data of the provided image constructed from the photograph

Methods:

Getter and setter for the image.

enhanceImage: Image enhancement techniques will be applied with this function. Image enhancement is the process of improving the quality of a digital image for further processes.

segment: Extract the note positions and values from the image.

createSheet: This methods creates the sheet object using the knowledge which is gotten from image processing methods.

4.1.3. Player Class

This class enables to run the MIDI that is given or produced by Image object synchronized with the sheet object.

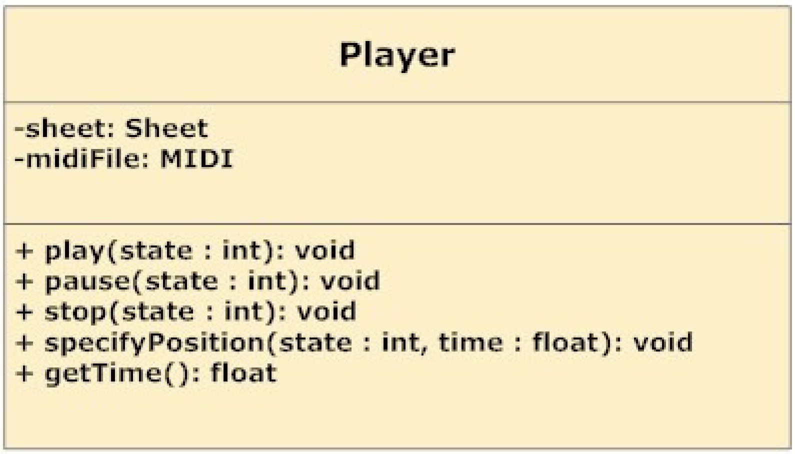


Figure 4. Player Class Diagram

Attributes:

sheet: Contains the sheet object of the selected image object or MIDI file.

midiFile: Constructed built-in MIDI object from the specified MIDI file that is given or produced via Image object.

Methods:

play: This method will start to play current MIDI file.

pause: This method will pause the currently playing MIDI.

stop: This method will stop the currently playing MIDI.

specifyPosition: This method will be used to specify starting point for player.

getTime: This method will return the instantaneous position of player over the file.

4.1.4. Sheet Class

This class is the base for all musical related work. It consists of Bar objects, and specifications of the specific sheet music represented as the attributes.

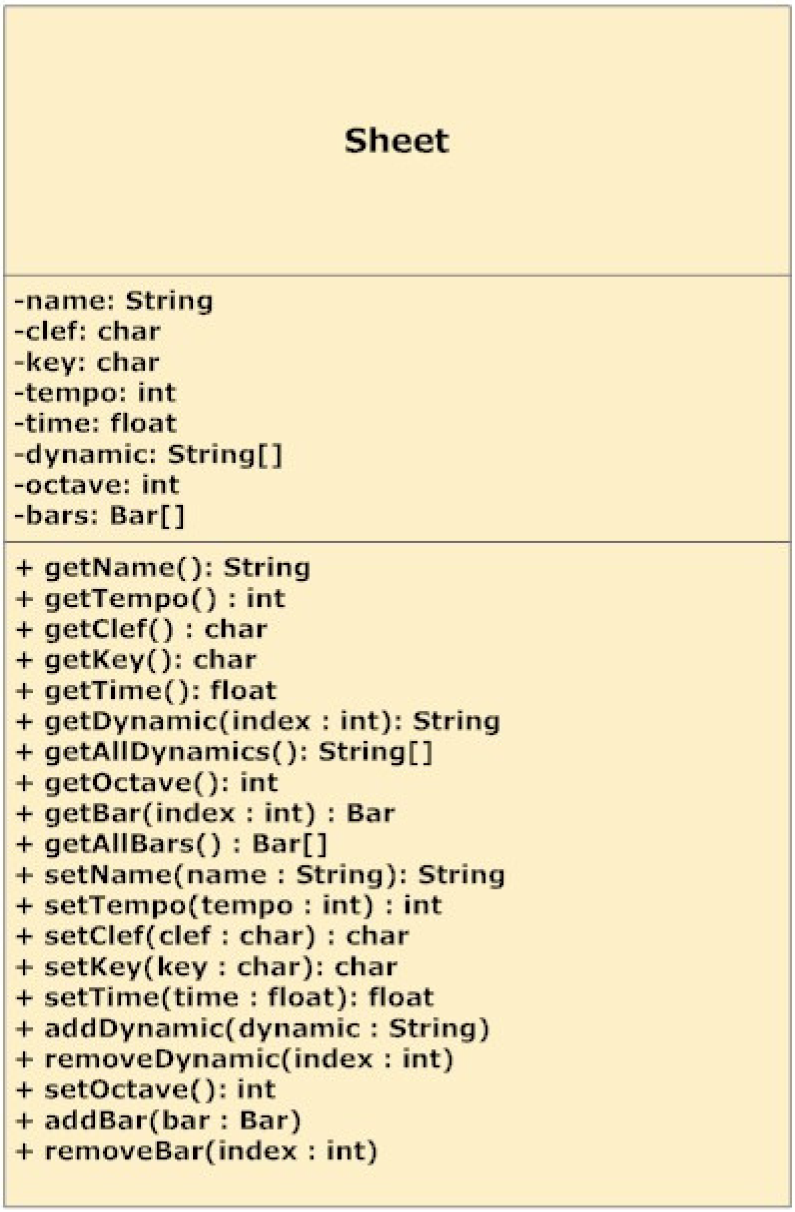


Figure 5. Sheet Class Diagram

Attributes:

name: Unique name of the sheet

clef: Clef attribute defines the pitch range. It can be G, F or C.

Key: Key of the sheet music defined in the start

tempo: Metronome of the music. Default speed is valid unless specified.

time: Represent the specific beat.

dynamic: Indicates the relative intensity or volume of a musical line. i.e. piano, forte, fortissimo etc.

octave: Octave of the sheet music. Default positions are valid unless specified.

bars: Bar object array contains the notes in the sheet.

Methods:

There are only getter and setter methods of all attributes.

4.1.5. Bar Class

This class is the container of basic musical components, notes. It has the attributes on its own to enable repetition facilities between bars.

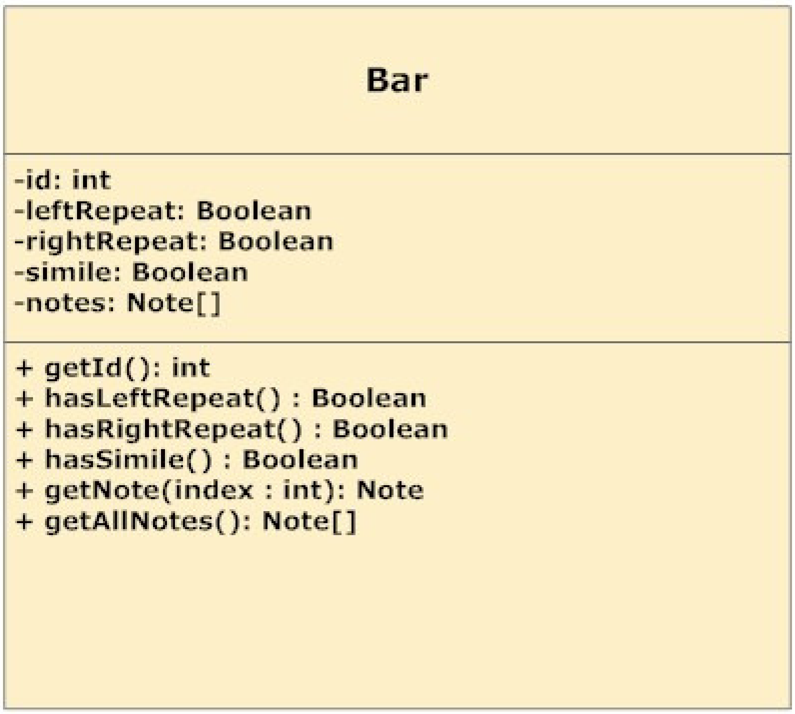


Figure 6. Bar Class Diagram

Attributes:

Id: Unique ID of the bar.

leftRepeat: Indicator of whether the bar object contains a left repeat mark or not.

rightRepeat: Indicator of whether the bar object contains a right repeat mark or not.

simile: Indicator of whether the bar object contains a simile mark or not.

notes: Note object array contains the notes in one bar.

Methods:

There are only getter and setter methods of all attributes.

4.1.6. Note Class

This class basically defines the note concept adapted directly from the real life. A note object has a lot of attributes to have the capability of representing each musical nuance. The attributes are defined below.

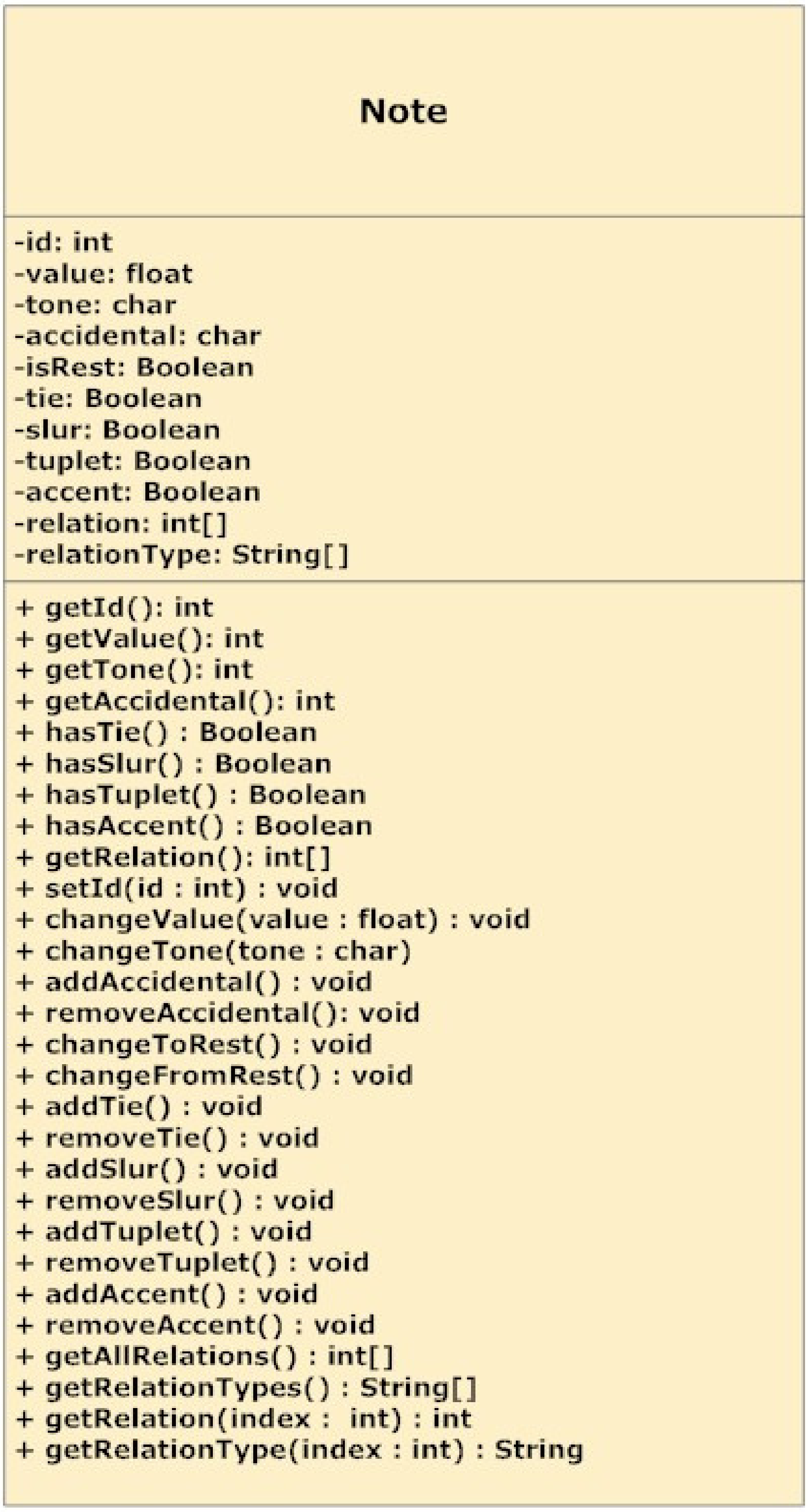


Figure 7. Note Class Diagram

Attributes:

Id: Unique ID of the note.

value: Simply defines the length of the note

tone: Defines the pitch value of the note. i.e. A, B, C, D, E, F, G

accidental: Defines if there is a pitch change by a semitone in the note, i.e. flat, sharp and natural.

isRest: Indicator of whether the note is a rest or not.

tie: Indicator of whether the note has a tie or not. If yes, enumerates all the tied notes with the same value.

slur: Indicator of whether the note has a slur or not. If yes, enumerates all the slurred notes with the same value.

tuplet: Indicator of whether the note is played as a tuplet or not. If yes, enumerates all the notes in the tuplet with the same value.

accent: Indicator of whether the note is played with an accent or not. If yes, the corresponding enumeration value of the appropriate accent is contained as the value.

relation: Indicator of whether the note is connected with other notes or not. If yes, enumerates all the connected notes with the same value.

Methods:

There are only getter and setter methods of all attributes.

4.1.7. MainActivityClass Class

This class is the main class of the application. Application starts with this class and all UI activities are operated by MainActivityClass. Moreover, main features of the application such as image processing and playing MIDI is controlled here.

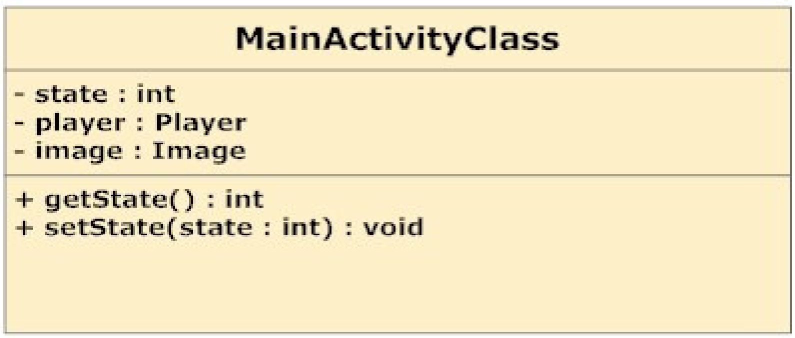


Figure 8. MainActivityClass Class Diagram

Attributes:

state: A state information for GUI.

player: Player class instance. It is used to manipulate all operations over the MIDI file.

image: An image instance which is taken by user. This image is processed by here and used to create sheet object.

Methods:

There are only getter and setter methods of state attribute.

5 SYSTEM ARCHITECTURE

In this section, software system architecture for DigiMuse is presented in a top-down manner.

5.1 Architectural Design

There are three main components interacting with each other in the project, "DigiMuse". These are discussed in the coming sections and their roles are indicated by using sequence diagrams.

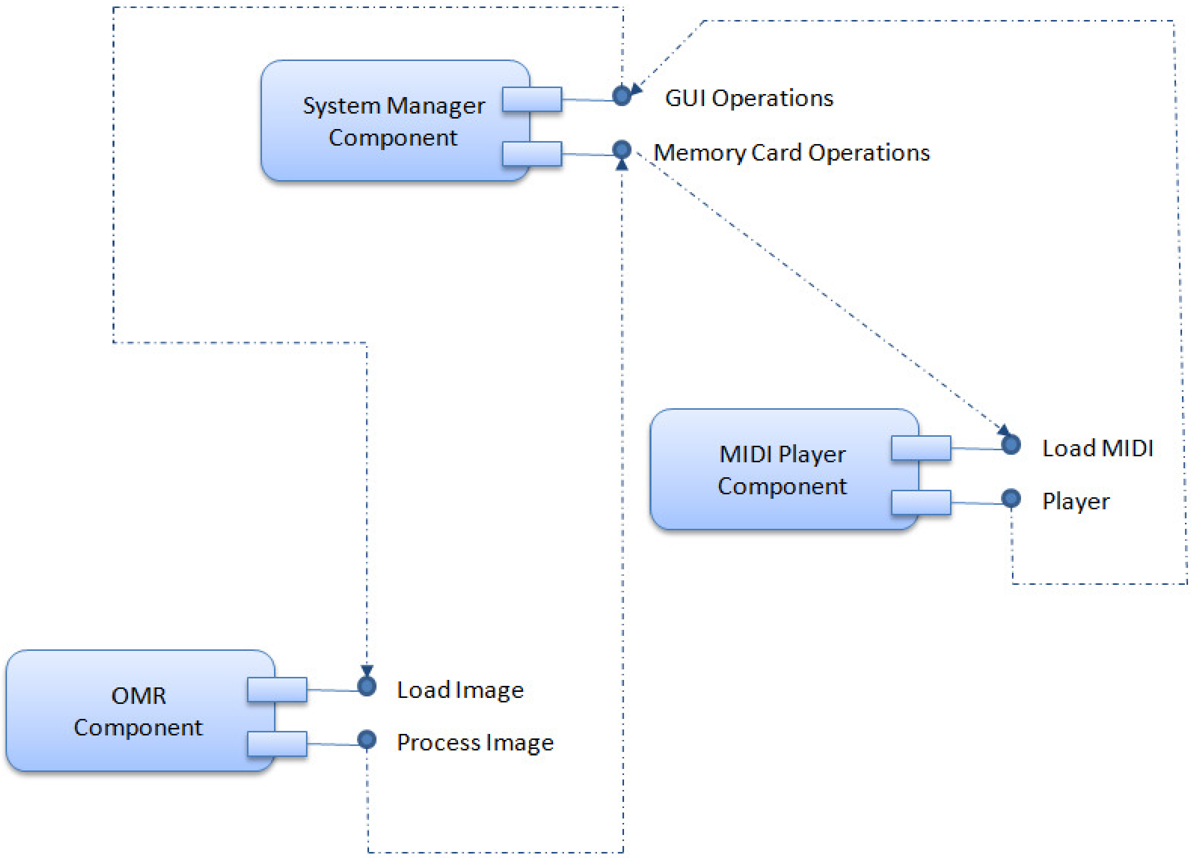


Figure 9: Component Diagram

5.2 Description of Components

This section describes the major components of “DigiMuse”. It has three main components which are Optical Music Recognition Component, MIDI Reader Component and System Management Component.

5.2.1 Optical Music Recognition (OMR) Component

5.2.1.1. Processing narrative

Optical Music Recognition component provides the necessary functionality for converting the photo of the printed sheet music which is taken by a mobile device into a Sheet instance.

5.2.1.2. Interface description

This component gets a photo of a printed image as an input and it returns an instance of a Sheet class.

5.2.1.3. Processing detail

Optical music recognition component is responsible for converting the photos of printed musical sheet into a digital one as accurate as possible. To define all musical characters necessitates the usage of image processing methods. After all the characters are recognized, corresponding attributes are specified as a Sheet instance.

5.2.1.4. Dynamic behavior

Sequence diagram for the dynamic behavior of OMR component is shown in Figure 10.

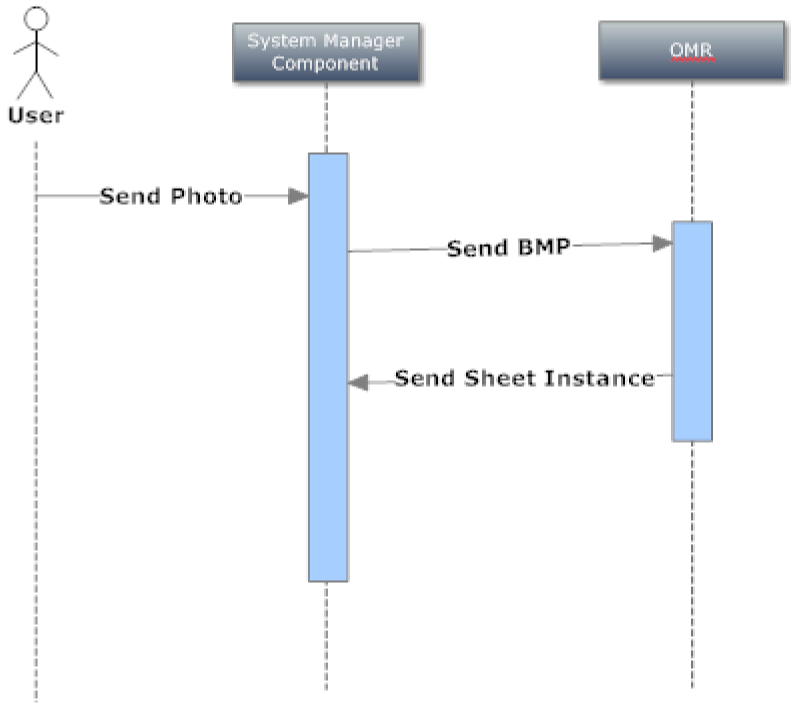


Figure 10: Sequence Diagram for OMR Component

Optical music recognition can interact with the MIDI reader component. After it outputs a Sheet instance, MIDI reader module has the capability to convert into MIDI format.

5.2.2 MIDI Player Component

5.2.2.1. Processing narrative

MIDI Player component provides the necessary functionality to read MIDI files and apply edit functions defined in section 4. These functionalities consist of both the music-related work and general functions such as stopping, playing, saving the MIDI files etc.

5.2.2.2. Interface description

This component gets a Sheet İnstance as an input from System Manager component and it returns a Sheet Instance as an output back to System Manager component.

5.2.2.3. Processing detail

MIDI Player component has interactions with System Manager Component. When it receives a Sheet Class instance from the System Manager component, it can apply editing operations on it. For example, it can change the value of the notes, add or delete accidentals, change the clefs used depending on the user preferences. These operations are just a few of the functionalities which "DigiMuse" has.

5.2.2.4. Dynamic behavior

Sequence diagram for the dynamic behavior of MIDI Player component is shown in Figure 11.

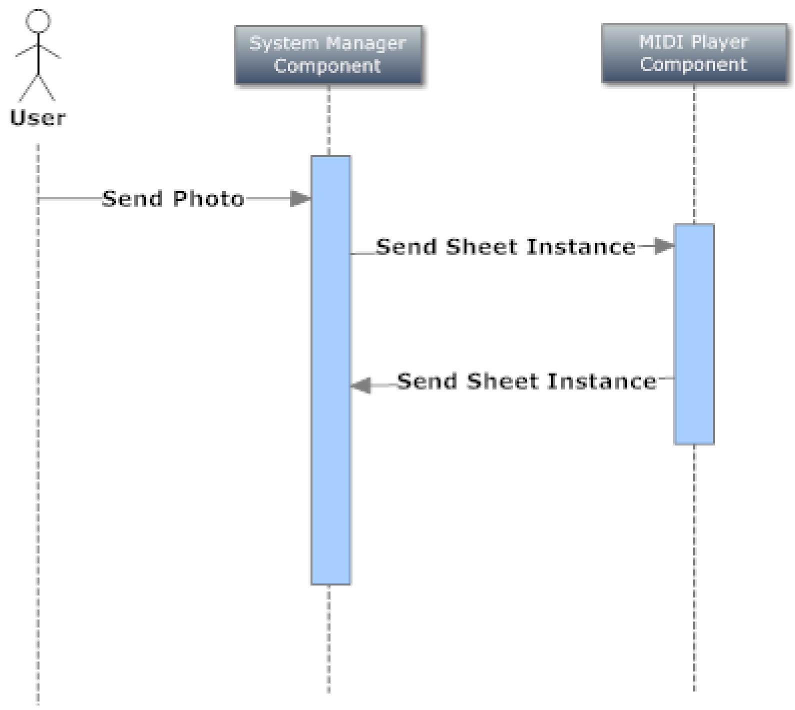


Figure 11: Sequence Diagram for MIDI Player Component

5.2.3 System Manager Component

5.2.3.1. Processing narrative

System Manager Component provides the necessary functionality to create MIDI files and saving them to SD card.

5.2.3.2. Interface description

This component gets a photo from the user as an input. It has interactions with both Optical Music Recognition component and MIDI Player Component. It sends a BMP formatted file to OMR component and as an input, it receives a Sheet instance. Moreover, it sends a Sheet instance to MIDI Player component and receives a Sheet instance as an input.

5.2.3.3. Processing detail

When the user takes a photo of a printed sheet music, System Manager Component takes it and sends it to OMR component in BMP format. Another option is that, System Manager Component has already received its input from OMR, and it sends a Sheet Class instance to be edited by MIDI Player component. Both OMR and MIDI Player component outputs Sheet Class intance and System Manager component receives them .Then, it as the capability to save the files which it received as input to SD Card of the device.

5.2.3.4. Dynamic behavior

Sequence diagram for the dynamic behavior of System Manager Component is shown in Figure 12.

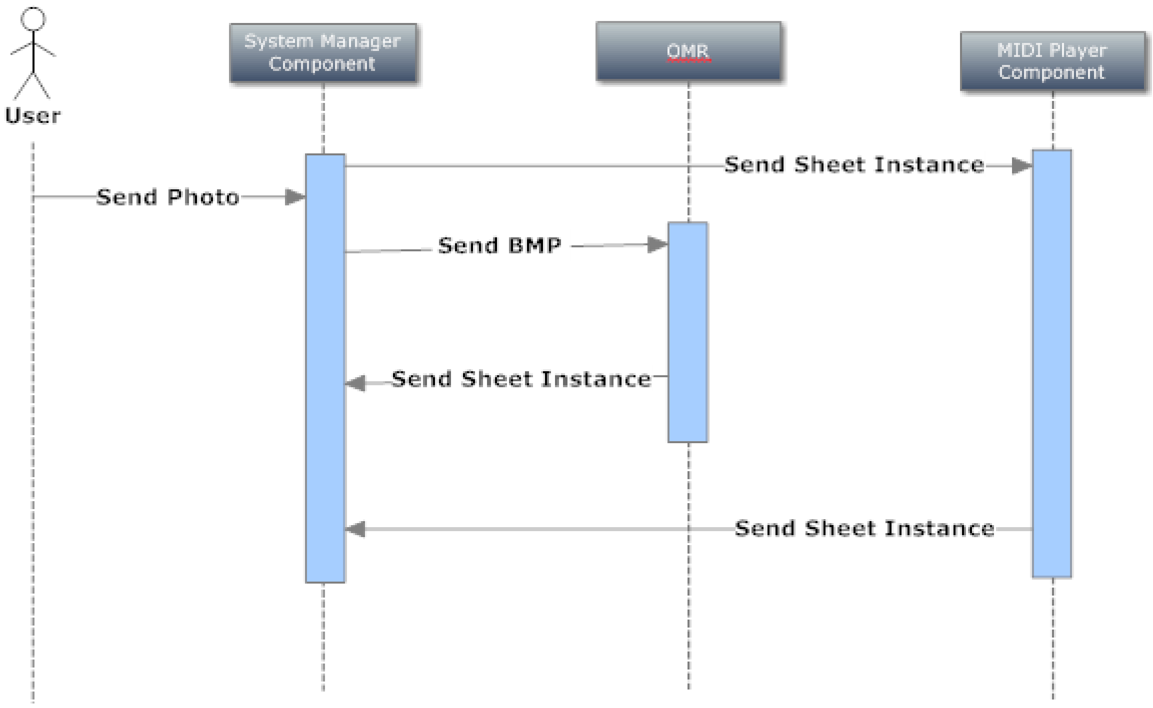


Figure 12: Sequence Diagram for System Manager Component

5.3 Design Rationale

After deciding on the project, the project group, GOBIT, made a detailed search on the shortages of the existing applications about optical music recognition. First of all, the product is decided to be a mobile application and in addition to recognizing printed symbols, our software can also read MIDI files.

System composition is planned in the most efficient way to satisfy what is required. The recognizable characters are chosen to be as many as possible. While designing, all group members discussed on the usage area of the application and it is decided that it would be better at reading classical music characters.

5.4 Traceability of requirements

SRS	SDR
3.2.1 Image to MIDI Conversion	5.2.1 Optical Music Recognition Module
3.2.2 Changing the User Interface	5.2.2 MIDI Reader Component
3.2.3 Instrument Selection	5.2.2 MIDI Reader Component
3.2.4 Note Edit	5.2.2 MIDI Reader Component
3.2.5 Play/Pause	5.2.2 MIDI Reader Component
3.2.6 Transpose	5.2.2 MIDI Reader Component
3.2.7 Clef Change	5.2.2 MIDI Reader Component
3.2.8 Load MIDI	5.2.3 System Manager Component
3.2.9 Save As...	5.2.3 System Manager Component
3.2.10 Export	5.2.3 System Manager Component

6 User Interface Design

6.1 Overview of User Interface

DigiMuse will start with the screen that has two buttons for two different functionalities as shown in the Figure 13.

Specifically:

- Take Photograph
- Open MIDI

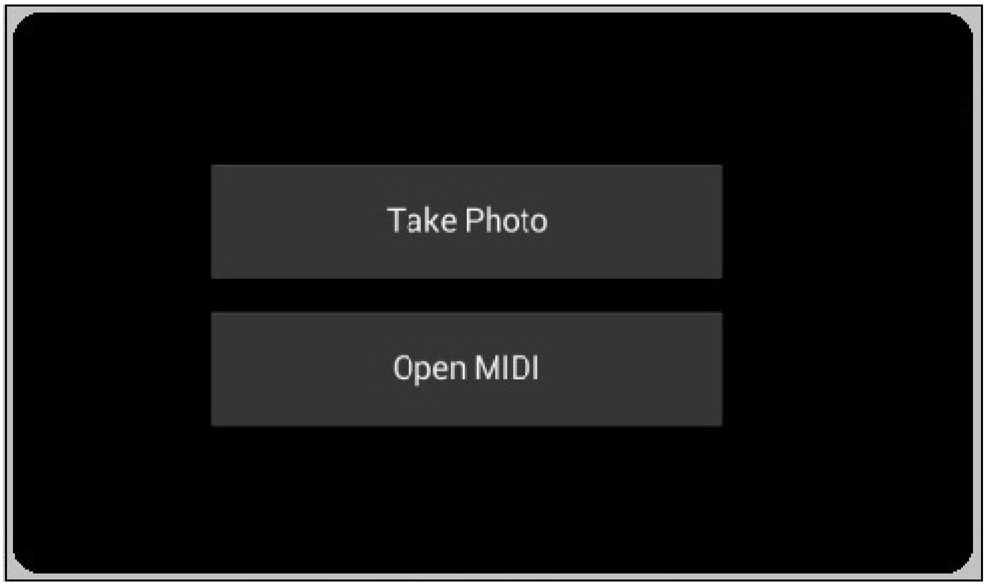


Figure 13. Entrance Screen

At this point the user chooses one of two options, in particular the user may choose to take a picture or load a MIDI file. In either case the user will be directed to the appropriate screen.

In case of “Open MIDI” option, the user will be able to load the file which has to be in MIDI format. If the users want to take a picture of some sheet note, he/she does so using the camera of the mobile device, the screen shown on the Figure 14 will be viewed:

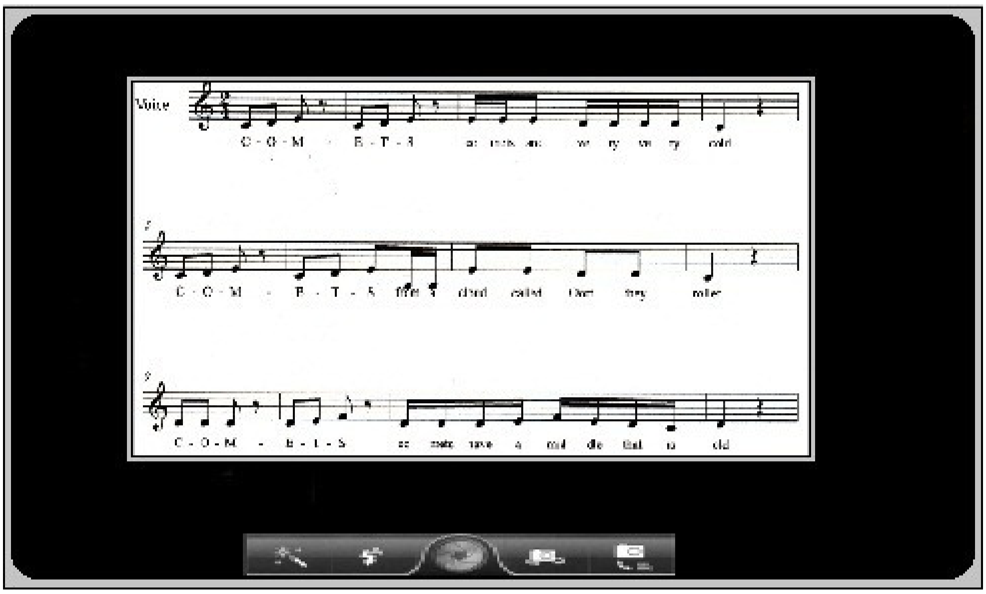



Figure 14. Taking Picture

Once the user has taken the picture, it is now ready to be processed by OMR Module.

The user will see the screen described in Figure 15, while the image processing is being handled:

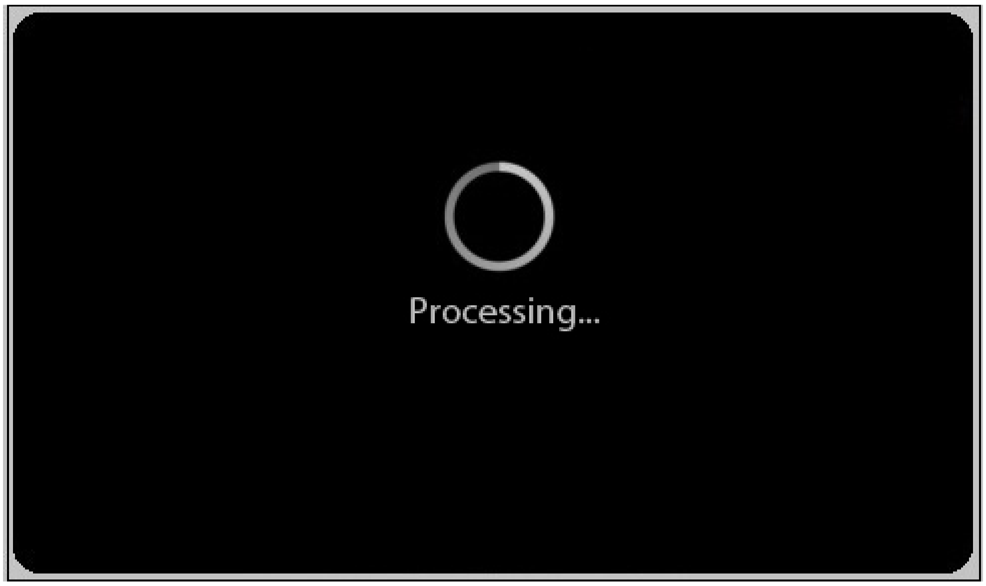


Figure 15. Waiting for Image Processing

Once the image has been processed, DigiMuse is now ready to be used and the user will view the screen on Figure 16:

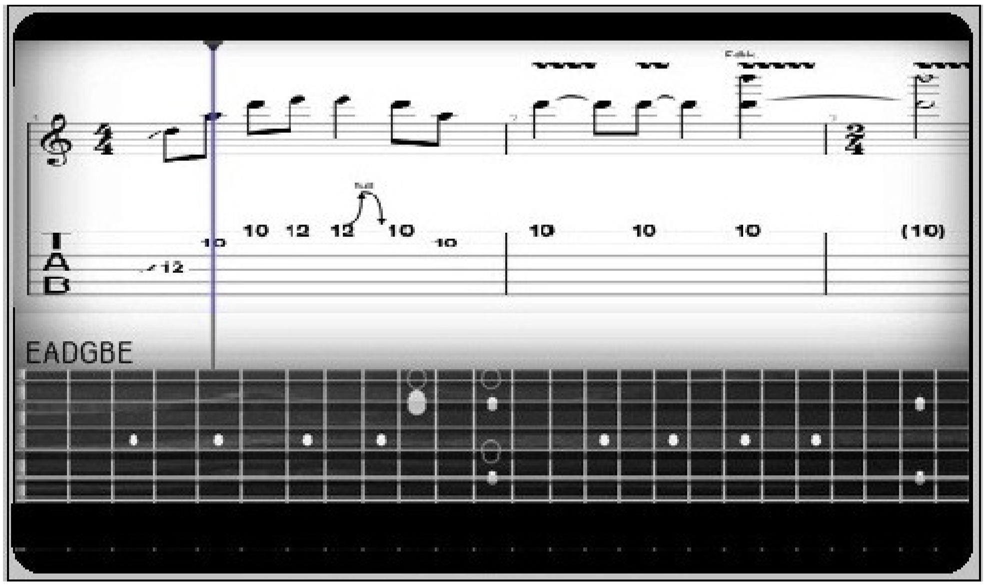


Figure 16. Main Screen View

This screen option views the sheet note, tab and keyboard sections all in one screen. However, if we take into consideration the fact that mobile devices have relatively small screen size, compared to laptops for example, we decided to provide the user with the chance to be able to separate the above screen into several parts.

There three different screen options consisting of different combinations of:

- Tab and Keyboard shown on Figure 17:

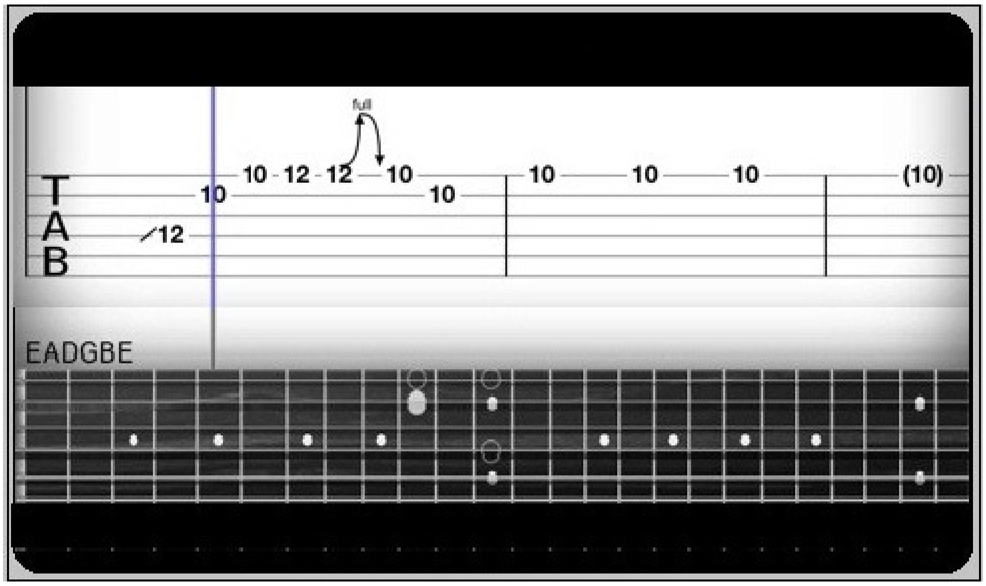


Figure 17. Main Screen View Variation (1)

- Sheet Note and Keyboard shown on Figure 18:

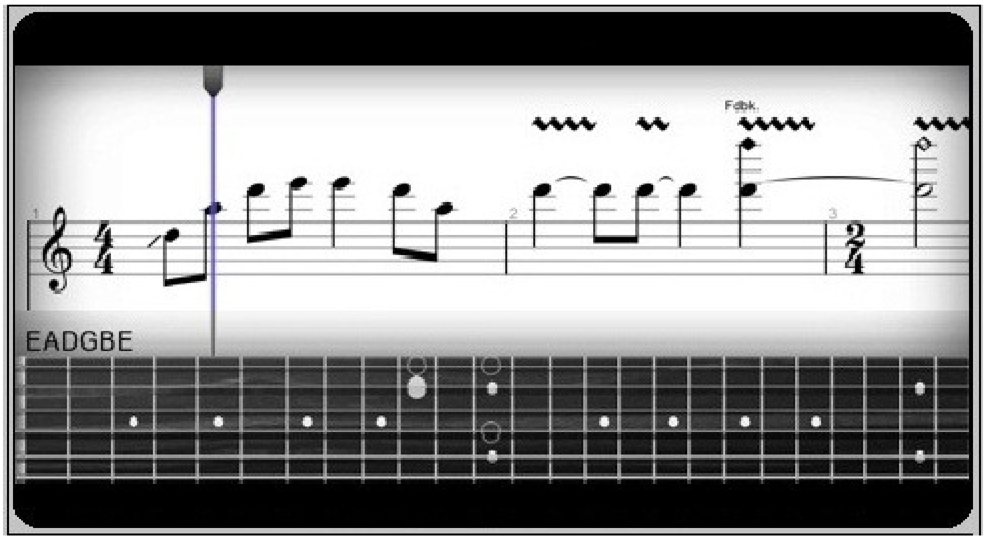


Figure 18. Main Screen View Variation (2)

After the user finishes playing the sheet music, he/she may quit the application using the "Exit" option from the main menu, which is going to be referred to and explained in more details in section 6.3.

- Sheet Note and Tab shown on Figure 19:

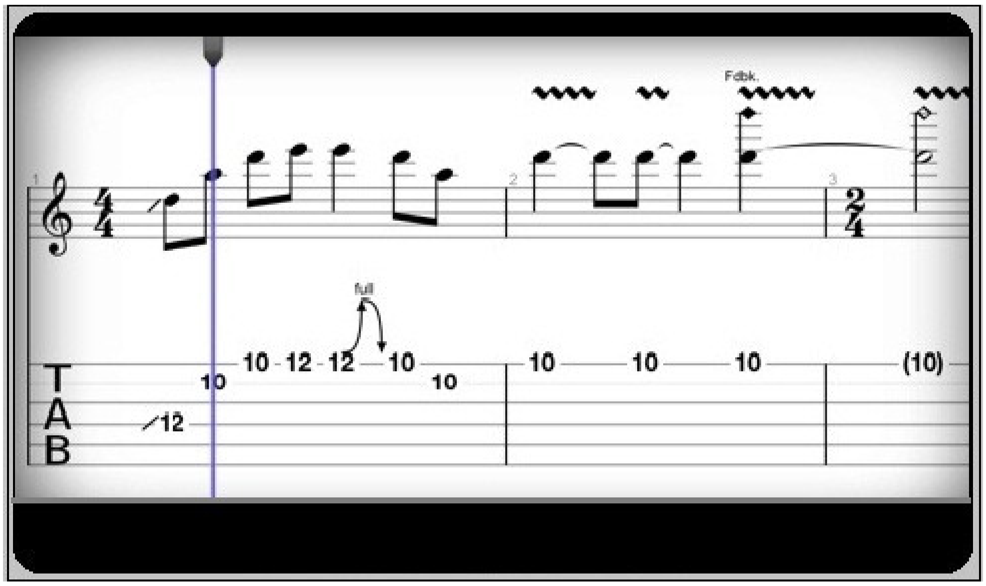


Figure 19. Main Screen View Variation (3)

We decided to give the main screen view in three different options in order to let the user feel comfortable viewing the content in terms of screen size and visibility.

6.2 Screen Images

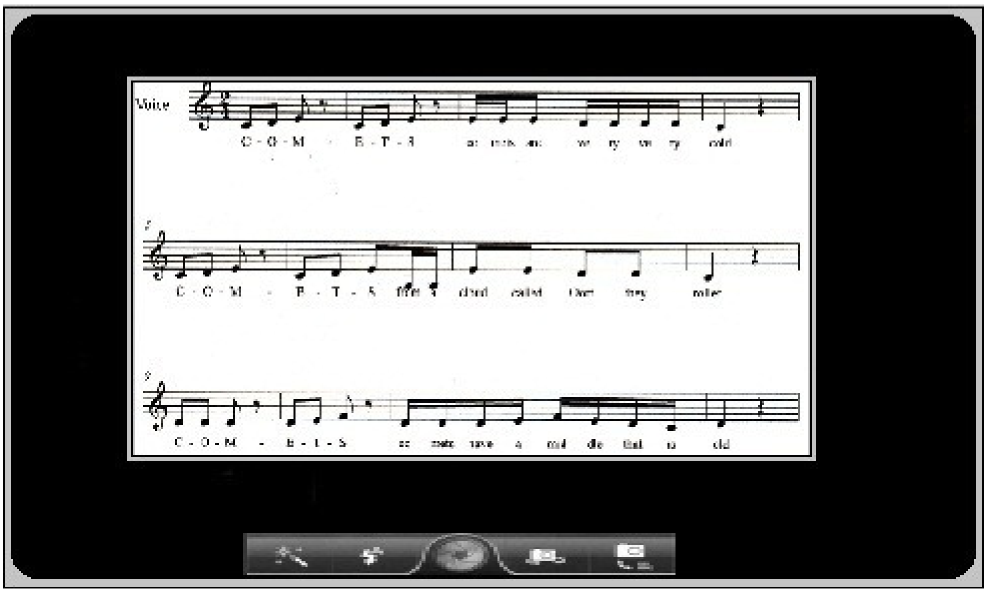


Figure20. Screen Image (1)

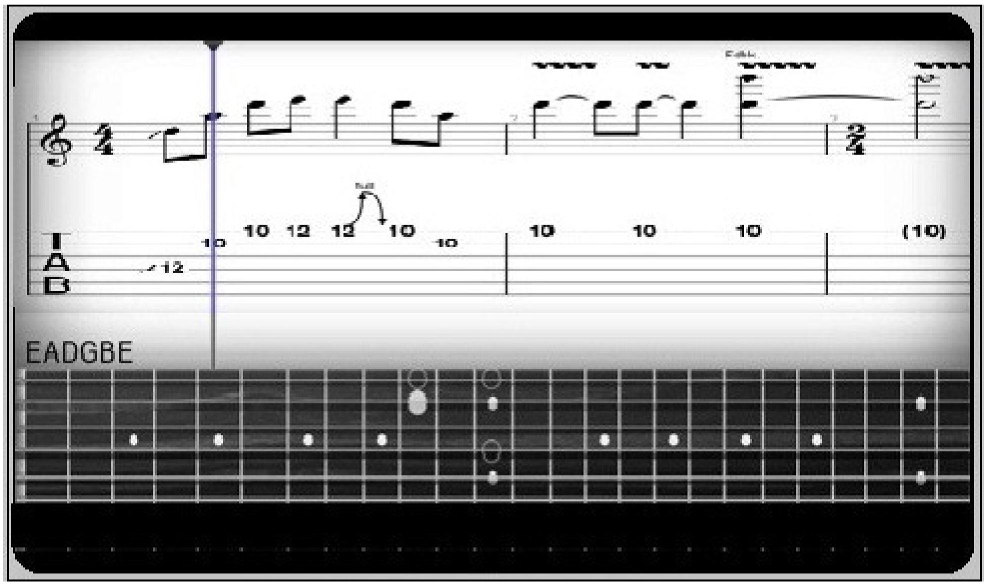
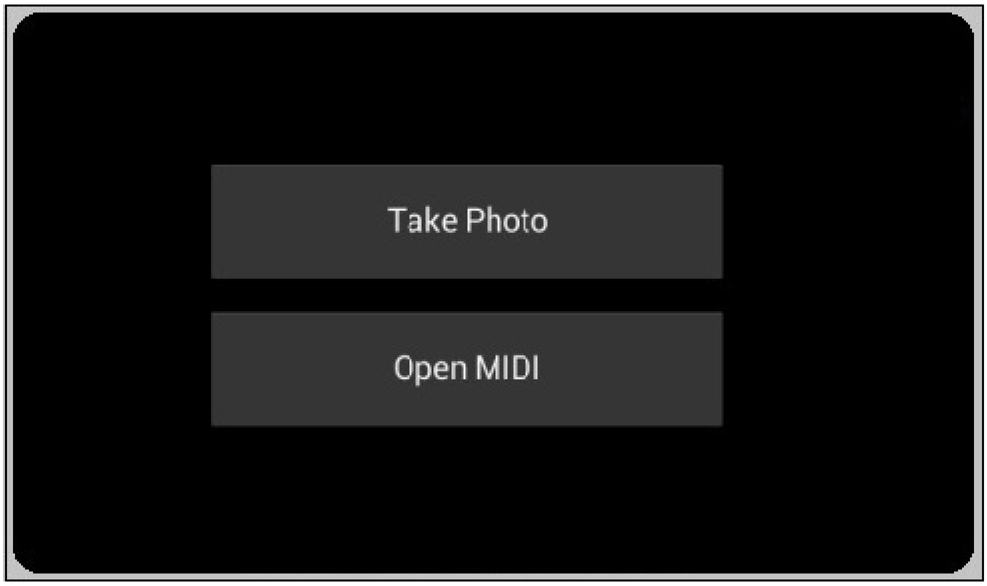


Figure 21. Screen Image (2)

6.3 Screen Objects and Actions

6.3.1. Entrance Menu

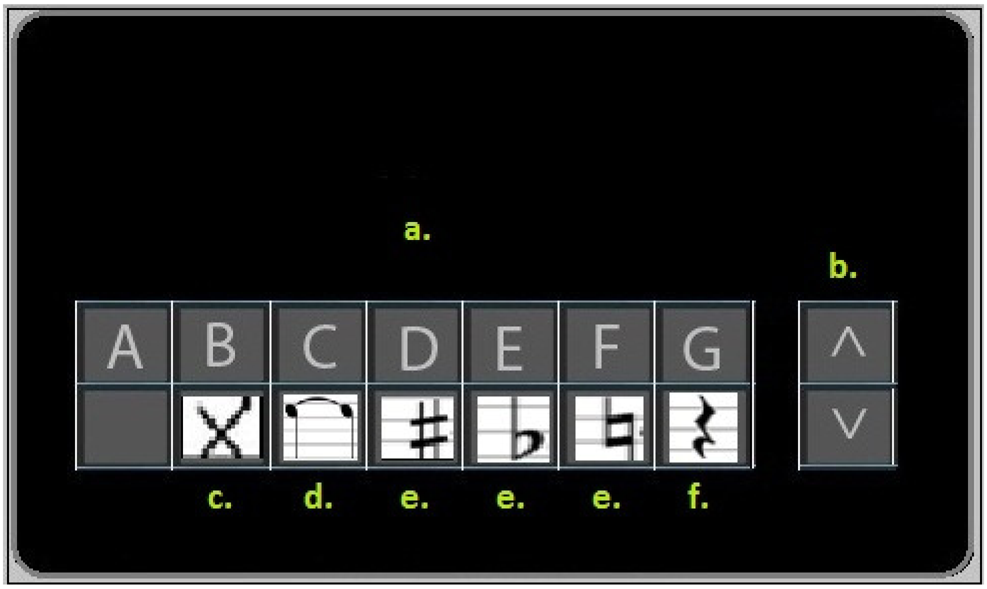


a. Take a photo to be processed. After this is selected camera of the device will be opened.
b. Open a MIDI file to listen and visualize the notes.
c.

6.3.2. Main Menu

a. New: Opens the entrance menu to create new player.
b. Export: Exports the currently playing MIDI to a readable and portable document.
c. Tempo: Change the playing pace of the MIDI.
d. Edit: Used for entering the editing mode.
e. Transpose: Used for transposing the sheet by desired pitch.
f. Octave Change: Used for changing the octave by desired amount.
g. Clef Change: Used for changing the clef with the desired one.
h. Save As: Used for saving the currently playing MIDI to a location on SD card.
i. Exit: Used for exiting the program.

6.3.3. Editing Keyboard



a. Letters: Corresponds to note tone. They are used for changing the tone of selected note.
b. Arrows: Used for changing the value of the selected note.
c. Ghost Note: Changes the note to a toneless click sound.
d. Note tie: Adds a tie between the selected note and the next note.
e. Accidentals: Used for changing the tone of the selected note by half tone.
f. Rest: Converts the note to a rest (silent) note with specified value by the arrows.

6.3.4. Edit Mode Menu

a. Dynamics: Used for changing dynamics of the selected note.
b. Articulation: Used for changing articulation of the selected note.
c. Octave: Used for changing octave of the selected note.

More information about these features can be found in SRS Section 3.2.4.1.

7. Detailed Design

7.1 Optical Music Recognition Component Design

Classification: This component is one of the main modules of the application. It is a distinct component. However it is operations are controlled by system manager component.

Definition & Responsibilities: This component is responsible for all image processing operations performed by the system. It will operate on a digital image. This image comes from the build-in camera of the device. OMR Component will take this image as an input and extract required features from the image using image processing and machine learning techniques. Finally it will create a system specific data structure for further needs of other components.

Constraints: Limited CPU power and limited memory are two main and very important constraints affecting this components' work. Since, mobile devices does not have high powered processor units and most image processing techniques requires lots of CPU cycle, basically CPU power, to run this will be our main limitation. Most Android powered devices have cameras that are capable of taking at least 5MP resolution images. 5MP resolution image means that this digital image consists of 5 million pixels. Many image processing techniques requires doing complex operations on each pixel of the image and to complete such a task we are trying to achieve, we need to apply lots of these operations.

Final implementation of the image processing component will do millions of operations on the image, thus it will take quite a time and limited CPU power makes this task even more time consuming. Limited memory of mobile devices is another important constraint. Android system allows at most 16MB of memory usage for each application. As stated earlier, images that the system is dealing with are at least 5MP. Android system's image format is 'RGBA_8888' which means every pixel is 32-bit, 4 bytes. Therefore, one image requires 20MB of memory which is obviously impossible. The size of the images needs to be lowered using down-sampling techniques. Even so memory is still an important limitation, since other operations and internal data structures requires memory too.

Moreover, OMR component depends on System Manager Component. System Manager Component is responsible for GUI, thus obtaining the image from the camera of the device. Since, OMR component will operate on the photograph that user takes it cannot start its work unless System Manager Component provides the image user took.

Composition: This component has no notable sub-components. It is responsible for only one task which is carried out by a few classes. Effective implementation of required functions is the main consideration for this component.

Uses/Interactions: This component operates on the images that are provided by System Manager Component. User takes a photograph using their devices' camera and System Manager Component gets this image and feeds it to OMR Component in a desired format.

OMR Component operates on the image using its classes. These classes apply required operations on the image which will be explained later in this section. Finally, a system specific data structure will be created and sent back to System Manager Component. This component will send the data to MIDI Reader Component. MIDI reader component will achieve all its functionalities using this data. Moreover, System Manager Component will create a MIDI formatted file out of the output of OMR Component and write it to SD Card of the device.

Resources: OMR Component requires lots of memory and CPU power as stated earlier in this section. To be more specific an image which is down-sampled by 4 requires approximately 1.5 MB memory. Moreover, the output data of this component will require almost as much memory as the image.

This component uses an external library, namely OpenCV. OpenCV stands for open source computer vision [7]. It is commonly used both for academic and commercial needs. The library has >2500 optimized algorithms for image processing and computer vision purposes.

Processing: The main task of the OMR component is to extract desired features from the image of a sheet music. Thus, the whole functionality of this component can be thought as one algorithm. This algorithm is performed in four steps as can be seen in Figure 22.

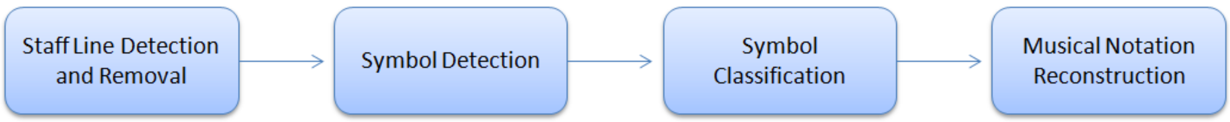


Figure 22: OMR Component Processing Steps

First step is line detection and removal. Lines are detected to determine the relative positions of notes according to stave lines so that determine the value of the note. After detection of stave lines they will be removed. This will be done for better recognition of musical characters. For line detection Hough Line Transform will be used. The Hough Transform is a technique which can be used to isolate features of a particular shape such as lines within an image [8]. Using proper parameters all stave line positions can be extracted from the image. After line detection the image needs to be de-skewed. De-skewing an image means rotating an image to align it in terms of a feature, in our case stave. This rotation will be done using Affine Transform. Affine Transform can be performed using functions provided by OpenCV library.

Second step is finding symbols. Once image is aligned it will be segmented using the local information gained by Hough Line Transform. All line positions are known now and it can be easily found the positions and boundaries of staves. The image is segmented in terms of staves so every segment represents a note value. Now the algorithm will search every segment one by one and detect symbols. Histogram based detection will be done to find symbols. The algorithm will search each segment horizontally and detect the histogram jumps. These jumps mean that there is a symbol on that position.

For the third step the algorithm will classify all detected symbols. Symbols will be checked with pre-defined templates to find a match. However this may not be enough to find all the symbols since the size of a symbol may highly vary from one image to another. Therefore a learning technique can be used here. Our research shows that k-nearest neighbor algorithm is the best choice for this task. In the article named *"Optical recognition of music symbols"* it is shown that k-nearest neighbor gives the second best results for optical music symbol classification [9]. The reason this algorithm is chosen over the best algorithm (support vector machine) is its better time and space complexity performance and ease of implementation.

Fourth and final step is constructing musical notation. Once the positions and types of all symbols are extracted, OMR Component will construct the musical sheet in a digital format.

This format is basically a Java object which contains other class instances. In the system this class is named *"Sheet"* class. It contains all the information of the printed sheet such as, all note information, measure, clef etc. This object will be passed to System Manager Component once it is created. Then it will be used by MIDI Reader Component to play the musical part represented in the sheet.

Interface/Exports: OMR Component simply exports one Java object. This object is an instance of sheet class. The features of this class can be seen on Figure 4.5. More information about this class, so the format of the output of OMR Component can be found in section 4.1.4 of this document. This output is main data of the system. It is used by MIDI Player Component to play the musical part represented in the sheet. This component does not involve any interactions with the user.

7.2 MIDI Player Component Design

Classification: This component is responsible for playing and user interaction purposes together with System Management Component. Although this component provides the functionality for user interactions, it is meaningful in DigiMuse together with GUI of the System Management Component.

Definition & Responsibilities: This component is responsible for all playing and editing purposes performed by the system. MIDI Player can also be defined as a container class for the main data structure Sheet object.

MIDI Player opens a specified MIDI file and sends it to System Management Component. MidiMod object converts this MIDI into Sheet object and sends it back. MIDI Player provides necessary options for GUI and responsible for all playing, editing and user-defined option changing. These functions are controlled via GUI of System Management and necessary operations are done by MIDI Player. Built-in Jet Player is responsible for playing the sound, member functions are responsible for changing the options and visualizing purposes.

Constraints: Memory is the most essential limitation for MIDI Player. Since Android allows the application to run low memory i.e. 16 MB in HTC Wildfire, editing and making use of the sheet object is going to be a major constraint to be satisfied. Apart from that, CPU is another limitation as MIDI Player Component is responsible for all functionalities of playing and editing. Significant amount of CPU should be allocated to MIDI Player considering that it is the most essential processing component for the real time usage of DigiMuse.

Composition: This component has no sub-components. It mainly consists of Player object which contains the main data structure Sheet object. Sheet object is composed of all specific components of musical sheet i.e. bar and note objects. In addition to that, it has the built-in MIDI file component which is used for sound playing purposes. Since it serves as data dictionary and serves all the functionality needed, this component kept simple for the sake of easiness for implementations.

Uses/Interactions: This component operates on the sheet object and MIDI file that are provided by System Manager Component. User desires to load a MIDI or just takes a photo and saves as MIDI, then opens it. System Manager Component converts the MIDI file into Sheet object and feedback both MIDI file reference and sheet object to the MIDI Player Component.

MIDI Player Component operates on the MIDI files to edit them or play sound using its classes. These classes apply necessary operations for editing and a built-in jet player object for playing. MIDI Player always interacts with System Management Component's GUI for providing user interaction both on playing and editing. All necessary GUI back-end functionalities are managed from here and reflected to GUI after that.

Resources: MIDI Player Component requires lots of memory since it contains the sheet object is contained in it. In addition, CPU usage is highly substantial as it is responsible for all functionality of both playing sound and illustrating visualizations. This component uses a built-in library in Android for playing MIDI, namely Jet Player.

Processing: Uses a sheet object which is translated by the MidiMod object of the System Component. Its output is directed to GUI in order to enable user interactions. Detailed explanations of this part can be found in Section 5.

7.3 System Management Component

Classification: This component is the last one of three components explained in this chapter. Furthermore, System Management is the one that establishes the connection between the other two components described above and itself.

Definition & Responsibilities: The task of this component is to provide binding utilities over the other two components along with some other processing tasks to be described. As it has already been stated in 'Definitions & Responsibilities' part for the OMR component, OMR will except the digital image as an input and produce as an output the system specific data structure which is then passed to System Management. At this point System Management component is responsible for saving this structure on SD card of the mobile device as well as passing it to Midi Player component, thereby letting these two components to interact with each other. As it has already been explained our application is going to be able to handle not only digital images as an input but Midi files will also be another input type we are going to process. So, again our System Management component will get the Midi file from the SD card of user's mobile device, parse this file and hand it to the Midi player. System Manager Component also has GUI and Main Activity class operations being handled and provided. This are going to be explained more widely in the sub-section called composition 'Composition 'under this section.

Constraints: Since System Manager Component is mainly going to be using the output of one component and pass it as an input to the another one, therefore, the constraints that are specific to those other components are also subject for System Manager Component too. To state it in more clear way, we have already said that the OMR component is very likely to suffer the time constraint issues due to the low CPU power of the mobile devices, and since System Manager is going to use the OMR Component's services, the related constraints are going to be directly effecting the performance of the System Manager Component as well. Image processing techniques require doing hard and complex operations and the greater the image resolution the more difficulties we are going to have with time and memory constraints.

Composition: System Manager Component consists of GUI, Main Activity and MidiMode subcomponents. In an Android application, the user interface is a built feature that uses View and ViewGroup objects. There are many types of views and view groups, each of which is a descendant of the View class. Since we are implementing an Android application we are going to make use of Android built-in GUI services. The System Manager Component will also control the GUI interactions and functions.

View objects are the basic units of user interface expression on the Android platform. The View class serves as the base for subclasses called "widgets," which offer fully implemented UI objects, like text fields and buttons. The ViewGroup class serves as the base for subclasses called "layouts," which offer different kinds of layout architecture, like linear, tabular and relative.

A View object is a data structure whose properties store the layout parameters and content for a specific rectangular area of the screen. A View object handles its own measurement, layout, drawing, focus change, scrolling, and key/gesture interactions for the rectangular area of the screen in which it resides. As an object in the user interface, a View is also a point of interaction for the user and the receiver of the interaction events. The user interface of an application is displayed on a device through an Activity, typically with one Activity created for each unique screen. Internally there is a stack of Activities, when moving from one screen to another, the next Activity to be visible is pushed onto the top of the stack – put another way, the Activity on the top of the stack is what is visible on the display. Activities are popped from the stack by pressing the back button, which resumes the previous Activity. In order to create a new Activity we extend the Activity class. The Activity class is and important part of an application’s overall lifecycle, and the way activities are launched and put together is a fundamental part of the platform’s application model. Finally, the MidiMod sub-component is the last one of the System Manger sub-components. System Manager uses MidiMod to get the sheet object from OMR Component, save it to some file on SD Card and then pass it to the Midi Player. In the same way MidiMod is used by the System Manager to get the some Midi formatted file from the SD Card then parse it and then hand it to Midi Player again.

Uses/Interactions: System Management is the component that is responsible for all interactions between all three components of the application. It uses the OMR component's sheet object as data needed to be passed to the Midi Player. System Manger also saves this data on SD Card of the device for later needs and uses. If it is the Midi file that has to be processed instead of some digital picture data is it again the System Manager who takes this data from the SD Card and parses it. Afterwards, this data is sent to the Midi Player in order to be played. System Manager also uses the built-in GUI subcomponent to handle all the user interactions that we are implementing for our application.

Resources: All resources that our application uses directly are hold on the SD Card. We do not have any database to accommodate the data we use. So the System Manager uses the SD Card and the other components to get all resources that are needed for its operations.

Processing: Processing details of this component are referred to in details in the System Architecture part of this document.

Interfaces / Exports: The component interfaces with the user via a graphical user interface. The input from the user will be handled directly within management component with the help of the Main Activity class as well. The results of user actions will be transmitted to other components if necessary.

8. Libraries and Tools

8.1 OpenCV

8.1.1. Description

OpenCV (*Open Source Computer Vision Library*) is a library of programming functions mainly aimed at real time computer vision, developed by Intel. It is free for use under the open source BSD license. The library is cross-platform [6]. OpenCV has C, C++ and Python interfaces so far. It focuses mainly on real-time image processing and the implementations are highly optimized [10]. OpenCV provides lots of functionality for segmentation and recognition.

8.1.2. Usage in Digimuse

Digimuse involves lots of image processing and pattern recognition due to its Optical Music Recognition engine and mobile device's camera usage. First of all, digitalizing a sheet music requires highly an enhanced image in order to achieve high accuracy rates. Also, the image will be saved using mobile device's camera, so it needs aligning and pre-processing. For these, image processing facility of OpenCV will be used. After that, using the enhanced sheet music image, segmentation and pattern recognition functionalities of OpenCV will be used for note detection.

8.2 Android SDK

8.2.1. Description

The Android SDK includes a variety of tools that help you develop mobile applications for the Android platform. The tools are classified into two groups: SDK tools and platform tools. SDK tools are platform independent and are required no matter which Android platform you are developing on. Platform tools are customized to support the features of the latest Android platform.[11]

8.2.2. Usage in Digimuse

Since, DigiMuse operates on Android OS; Android SDK is a must to use.

8.3 Android NDK

8.3.1. Description

The Android NDK is a toolset which enables the users to embed components that make use of native code in your Android applications. Android applications run in the Dalvik virtual machine. The NDK allows the user to implement parts of his/her applications using native-code languages such as C and C++. This can provide benefits to certain classes of applications, in the form of reuse of existing code and in some cases increased speed.[12]

8.3.2. Usage in Digimuse

DigiMuse uses OpenCV as it is explained in Section 7.1. OpenCV is not yet compatible with Java but the finest quality and easiest to use library in Computer Vision. Thanks to Android NDK, the code using OpenCV can be written in C++ and embedded into main java code. The project greatly makes use of this functionality.

8.4 Eclipse

8.4.1. Description

Eclipse is a multi-language software development environment comprising an integrated development environment (IDE) and an extensible plug-in system [13]. Eclipse is a free and open source software. The IDE has different versions for different languages such as Eclipse CDT for C/C++, Eclipse JDT for Java and etc.

8.4.2. Usage in Digimuse

Digimuse, runs on devices operating on Android. Using Android Development Tools (ADT) plug-in of Eclipse JDT enables us to implement clean and efficient Java code. With C++ plug-in (CDT), OpenCV usage becomes easier.

9 Time Planning

9.1 Term 1 Gantt Chart

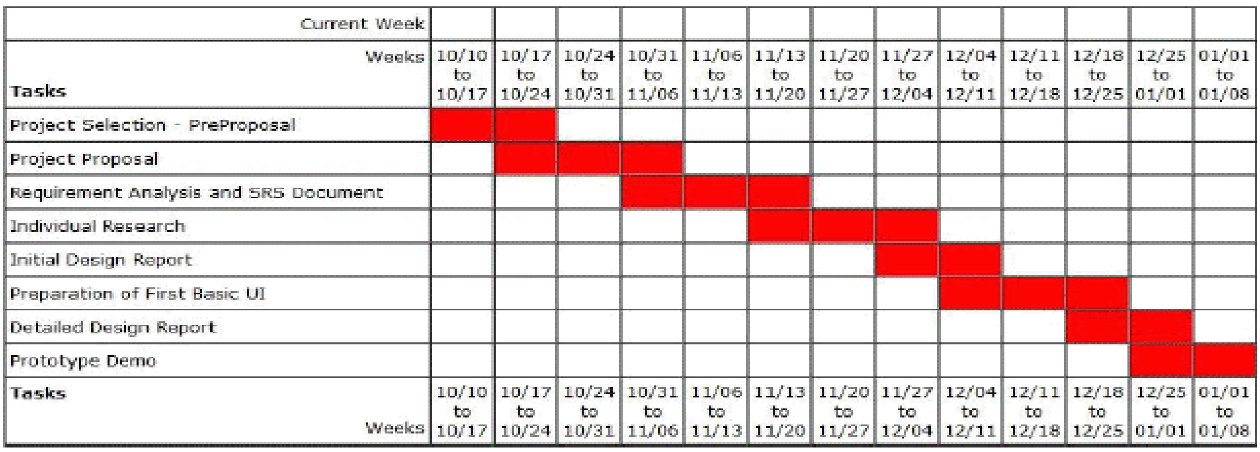


Figure 23: Term2 Gantt Chart

9.2 Term 2 Gantt Chart

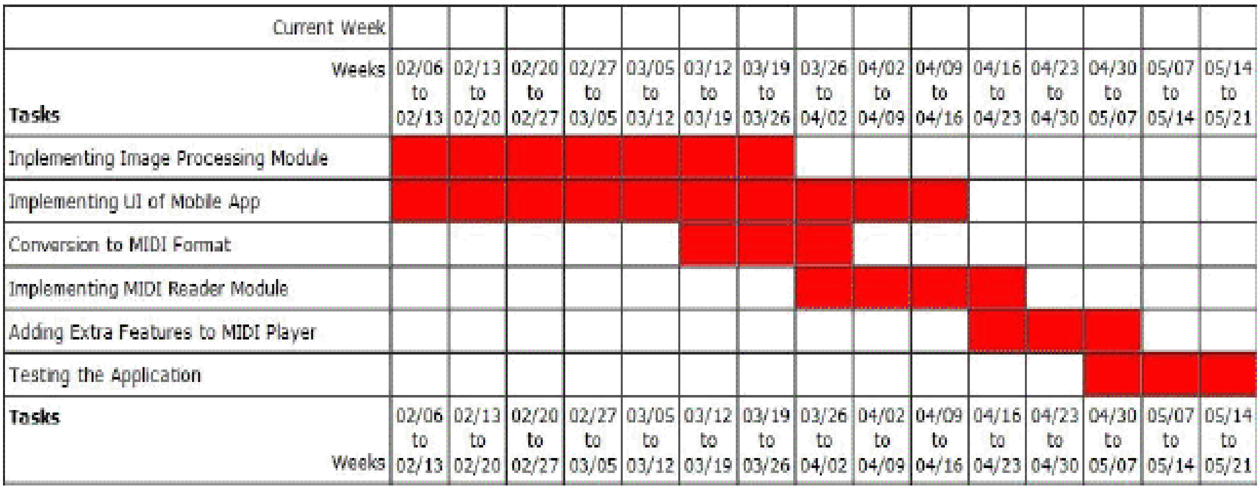


Figure 24: Term2 Gantt Chart

9 Conclusion

Detailed design report for "Digimuse" have been presented in this document. It includes the design for data model, system architecture, and user interfaces. A lot of detailing has been done on the project scenario that shows the most of the essential details. Moreover, it defines all the components with details by class diagrams and sequence diagrams, user interface design and how the user interacts with DigiMuse. Further information on the technical design is given with detailed explanations of the components. Different from IDR, this report includes a detailed design part. Lastly, the progress of the project so far and future plans are stated.