Design Document

EECS2311 TAB2XML_G14 PROJECT

GROUP 1:

Abdelrahman Altamimi

Hieu Le

Mahdiar Shoraka

Prabjot Dhaliwal

Yongjie Ba

Table of Content

Introduction

This design document is going to describe two different kinds of diagrams for the TAB2XML_G14 project: the class diagram and the sequence diagram. Also, we are going to talk about some scenarios for the maintenance of the project.

Overview

This application functions with the combination of 4 different units: (1) XML parsing unit, (2) Sheet Music Graphics Generation Unit, (3) Music Generation Unit, and (4) View Controller Unit.

1. *XML Parsing Unit*: Works virtually independently of the other 2 units. This Unit parses a music XML file in a string and stores all relevant parsed information within a single Score object.

    Score Class: Contains music XML information using objects of the classes found in the **custom_component_data** package. These classes usually correspond directly with an element/tag name in MusicXML files.

2. *Sheet Music Generation Unit:* Processes the parsed XML data from the XML Parsing Unit, and generates all the appropriate sheet music graphics. The whole sheet music graphics are represented by a single SheetScore object.

    SheetScore Class: This is a JavaFX graphics element that is composed of other graphical elements of classes found in the **custom_model** package and its sub-packages. Each class is a graphical element (as they extend a JavaFX class)

3. *Music Generation Unit:* Uses the Score object from the XML parsing unit to develop a JFugue pattern string. This audio is played via this string using the JFugue library. Handles all audio playback requests. This unit is composed of the single **music player** class within the **customer_player** package.

4. *View Controller Unit:* Links all the units, to create a GUI where input is received as tablature and the following are outputted accordingly with user interactions: sheet music graphics, audio playback, PDF exporting, and customizability settings. This unit is composed of classes found in the **GUI** package (classes of interest are shown on the next page)

Class Diagram

Notice: Because adding all the classes into one diagram may lead to a crowded and hard-to-read diagram, thus This document has separated them into several class diagrams

Music-XML Parsing Unit:

Score
~ title: String
~ author: String
~ partList: List<Part>
+ getTitle(): String
+ getAuthor(): String
+ getParts(): List<Part>

Part
~ id: String
~ name: String
~ measures: List<Measure>
~ instruments: HashMap<String, Instrument>
+ getId(): String
+ getName(): String
+ getMeasures(): List<Measure>
+ getInstruments(): HashMap<String, Instrument>

Instrument
~ id: String
~ name: String
+ getId(): String
+ getName(): String

Measure
~ tab: boolean
~ timeSignature: int[]
~ iaRepeatStart: boolean
~ isRepeatStop: boolean
~ percussion: boolean~ staffLines: int
~ notes: List<Note>
~ barLineRight: BarLine
+ getTimeSignature(): int[]
+ getStaffLines(): int
+ getNotes(): List<Note>
+ getPercussion(): boolean
+ getTab(): boolean
+ getIsRepeatStop(): boolean
+ getIsRepeatStart(): boolean

BarLine
~ repeatNum: int
+ getRepeatNum(): int

Note
~ instrumentID: String
~ type: int
~ dot: int
~ stem: String
~ notehead: String
~ grace: boolean
~ chord: boolean
~ rest: boolean
~ timeModification: HashMap<String, Integer>
~ notation: Notation
~ position: int
+ getNotation(): Notation
+ getType(): int
+ getDot(): int
+ getStem(): String
+ getNotehead(): String
+ getGrace(): boolean
+ getChord(): boolean
+ getRest(): boolean
+ getTimeModification(): HashMap<String, Integer>
+ getInstrumentID(): String
+ getPosition(): int

Technical
~ bend: Bend
+ getBend(): Bend

Notation
~ slur: List<Slur>
~ tied: list<Tied>
~ slide: list<Slide>
~ ornaments: Ornament
~ string: int
~ fret: int
~ technical: Technical
+ getSlurs(): List<Slur>
+ getSlides(): List<Slide>
+ getTieds(): List<Tied>
+ getString(): int
+ getFret(): int
+ getOrnamnets(): Ornament

Use (Technical ← Notation)
Use (Note ← Notation)
Use (Ornament ← Notation)

Bend
~ bendAlter: double
+ getBendAlter(): double

Slur
~ type: String
+ getType(): Strin

Slide
~ type: String
+ getType(): String

Tied
~ type: String
+ getType(): String

Ornament
~ tremolo: Tremolo
+ getTremolo(): Tremolo

Tremolo
~ number: int
+ getNumber(): int

Use (Technical → Bend)
Use (Ornament → Tremolo)

Music Generation/View Controller Unit Class Diagram

```
                    Application
                    <abstract>
```

Extend   Extend

The Controller for MainApplicaton   Used to customize display the sheet music

MainViewController

- prefs : Preferences
+ executor : ExecutorService
+ saveFile : File
- isEditingSavedFile : boolean
+ convertWindow : Window
+ settingsWindow : Window
+ highlighter : Highlighter
+ converter : Converter
+ tempoInput : TextField
+ mainViewState : Label
+ instrumentMode : TextField
+ mainText : CodeArea
+ gotoMeasureField : TextField
+ borderPane : BorderPane
+ <<handled>> saveTabButton : Button
+ <<handled>> saveMXLButton : Button
+ <<handled>> showMXLButton : Button
+ <<handled>> previewButton : Button
+ <<handled>> goToline : Button
+ cmbScoreType : ComboBox<String>

+ MainViewController()
+ initialize(): void
- handleCurrentSongSettings(): void
- handleSystemDefaultSettings(): void
- handleNew(): void
- handleOpen(): void
- handleSaveAs(): boolean
- handleSave(): boolean
- promptSave(): boolean
- openNewWindow(Parent, String): Window
- saveTabButtonHandle(): void
+ saveMXLButtonHandle(): void
- showMXLButtonHandle(): void
- previewButtonHandle(): void
+ unImplementedFunctionOnClick(String, String): void
+ refresh():void
- handleGotoMeasure(): void
- goToMeasure(int): boolean
+ listenforTextAreaChanges(): void
+ update(): Task<StyleSpans<Collection<String>>>
+ start(Stage): void

Use →

PreviewController

- mvc : MainViewController
- player : musicPlayer
- score : Score
- sheet : SheetScore
+ displayWindow : Window
+ mxlText : CodeArea
+ sp : ScrollPane
+ tempoField : TextField
+ gotoMeasureField : TextField
+ <<handled>> playButton : Button
+ <<handled>> pauseButton : Button
+<<handled>> goButton : Button
+<<handled>> exitButton : Button
+ <<handled>> displayButton : Button
+ <<handled>> exportButton : Button

+ PreviewController()
+ setMainViewController(MainViewController): void
+ update(): void
- initialButton(): void
- getBufferimage(): void
+ playHandler(): void
+ pauseHandler(): void
+ stopHandler(): void
+ exit(): void
+ goHandler(): void
+ displayHandler(): void
+ expotHandler(): void
+ namesHandler(): void
- initialValue():void
- openNewWindow(Parent, String): Window
+ start(Stage): void

DisplaySettingController

- pc : PreviewController
+ fontValues : ObservableList<String>
+ noteSpaceValues : ObservableList<Integer>
+ lineSpaceValues : ObservableList<Integer>
+ noteSizeValues : ObservableList<Integer>
+ <<handled>> applyButton : Button
+ <<handled>> resetButton : Button
+ <<handled>> applyAndExitButton : Button
- fontValue : ChoiceBox<String>
- noteSpaceValue : ChoiceBox<Integer>
- lineSpaceValue : ChoiceBox<Integer>
- noteSizeValue : ChoiceBox<Integer>

+ setPreviewController(PreviewController): void
+ update(): void
+ ApplyHandler(): void
+ resetHandler(): void
+ applyandexitHandler(): void
- loadFonts(): void
- loadNoteSize(): void
- loadNoteSpace(): void
- loadLineSpace(): void

Use

<<handled>> :
means this is a button, and a conect with
buttonNameHandler() method

Use

musicPlayer

+ noteList : List<Note>
+ score : Score
+ listner : StaccatoParserListener
+ parser : MusicXmlParser
+ player : Player
+ musicXMLParttern : Pattern
+ stringInstrument : String
+ drumSet : String
+ stepToNoteMap : String[]
+ instrument_type : int
+ tempoSpeed : int
+ sheet : SheetScore

+ musicPlayer(Score, SheetScore, String)
+ setNoteList(): void
+ getTempo(): int
+ play(String): void
+ isPaused(): boolean
+ isPlaying(): boolean
+ resume(): void
+ pause(): void
+ exit(): void
+ isFinished(): boolean
+ finish(): void
+ SetInstrumentType(): void
+ setInstrument(): void
+ setStringInstrument(): void
+ setDrumSet(): void
+ mapToNote(char, int): int
+ mapToDuration(int): String
+ developSlideString(Note, Note): String
+ getInstrumentType(): int
+ toString(): String
+ getNote():List<Note>
+ generateTimeModString(HashMap<String, Integer>): String

Music features are saved in this file

High-Level Sheet Music Graphics Generation:

Music Note Graphics Generation:

Group

NoteTail
width: double
+ NoteTail(double height, int num, boolean stemDown)

NoteUnit
- width: double
- spacingType: double
+ highlighted: boolean
- grace: boolean
- noteNum: int
- measure: int
+ pressed: NoteUnit
+ getNoteNum(): int
+ setNoteNum(int num): void
+ getWidth(): double
+ setWidth(double width): void
+ getMeasure(): int
+ setMeasure(int measure): void
+ getGrace(): boolean
+ setGrace(boolean grace): void
+ getSpacingType(): double
+ setSpacingType(double spacingType): void
+ *toggleHighlight(): void*

DisplayUnit
- top: double
- bottom: double
- height: double
- grace: double
- position: int
+ getPosition(): int
+ setPosition(int position): void
+ getTop(): double
+ setTop(double top): void
+ getBottom(): double
+ setBottom(double bottom): void
+ getHeight(): double
+ setHeight(double height): void
+ *extendStaff(int positions, double height): void*
+ addTails(double height, boolean stemDown): void

BoxedUnit
+ currMeasureNoteNum: int = 0

BoxedText
PADDING_LEFT: final int = 8
label: Text
container: Rectangle
+ BoxedText(String text, double size, double type, boolean grace, boolean chord, int measure)
+ toggleHighlight(): void

BoxedChord
frets: List<BoxedText>
+ BoxedChord(double size, List<Note> notes, int measure, boolean isGrace)
+ toggleHighlight(): void

DisplayChord
hasFlip: boolean
max_position: int
min_position: int
displayNotes: List<DisplayNote>
+ DisplayChord(double height, List<Note> notes)
+ sortNotes(List<Note> notes): void
+ isANoteFlipped(List<Note> notes): boolean
+ extendStaff(int positions, double heights): void
+ addTails(double height, boolean stemDown): void
+ toggleHighlight(): void

DisplayNote
preceding: double
trailing: double
parenthesesDisplacement: double
isChord: boolean
isNormalSide: boolean
dotScale: double = 3
dotDistanceScale: double = 1.5
noteHeadWidth: double
+ DisplayNote(double height, Note note, boolean hasFlip, boolean isFlip)
- addParentheses(int type, double height): void
+ addDots(double height, Note note): void
+ extendStaff(int positions, double heights): void
+ addTails(double height, boolean stemDown): void
+ toggleHighlight(): void

The following 2 Diagrams are GUI elements used by the DisplayNote class:

(Notehead)   **(Rest)**

CrossNoteHead
width: double
stemPosition: double
+ CrossNoteHead(double height, Color color, double strokeScale)

BreveNoteHead
width: double
+ BreveNoteHead(double height)

NoteHead
width: double
height: double
stemPosition: double
+ NoteHead(double height, int type, String noteHead)
+ getHeight(): double
+ getWidth(): double
+ getStemPosition(): double

Group

WholeNoteHead
width: double
+ WholeNoteHead(double height)

FullNoteHead
width: double
stemPosition: double
+ FullNoteHead(double height)

HalfNoteHead
width: double
stemPosition: double
+ HalfNoteHead(double height)

EighthRest
width: double
height: double
+ EighthRest(double height)

SmallRest
width: double
height: double
+ SmallRest(double height, int type)

Rest
width: double
height: double
+ Rest(double height, int type)
+ getHeight(): double
+ getWidth(): double

Group

HalfRest
+ HalfRest(double height)

QuarterRest
width: double
height: double
+ QuarterRest(double height)

WholeRest
width: double
height: double
+ WholeRest(double height)

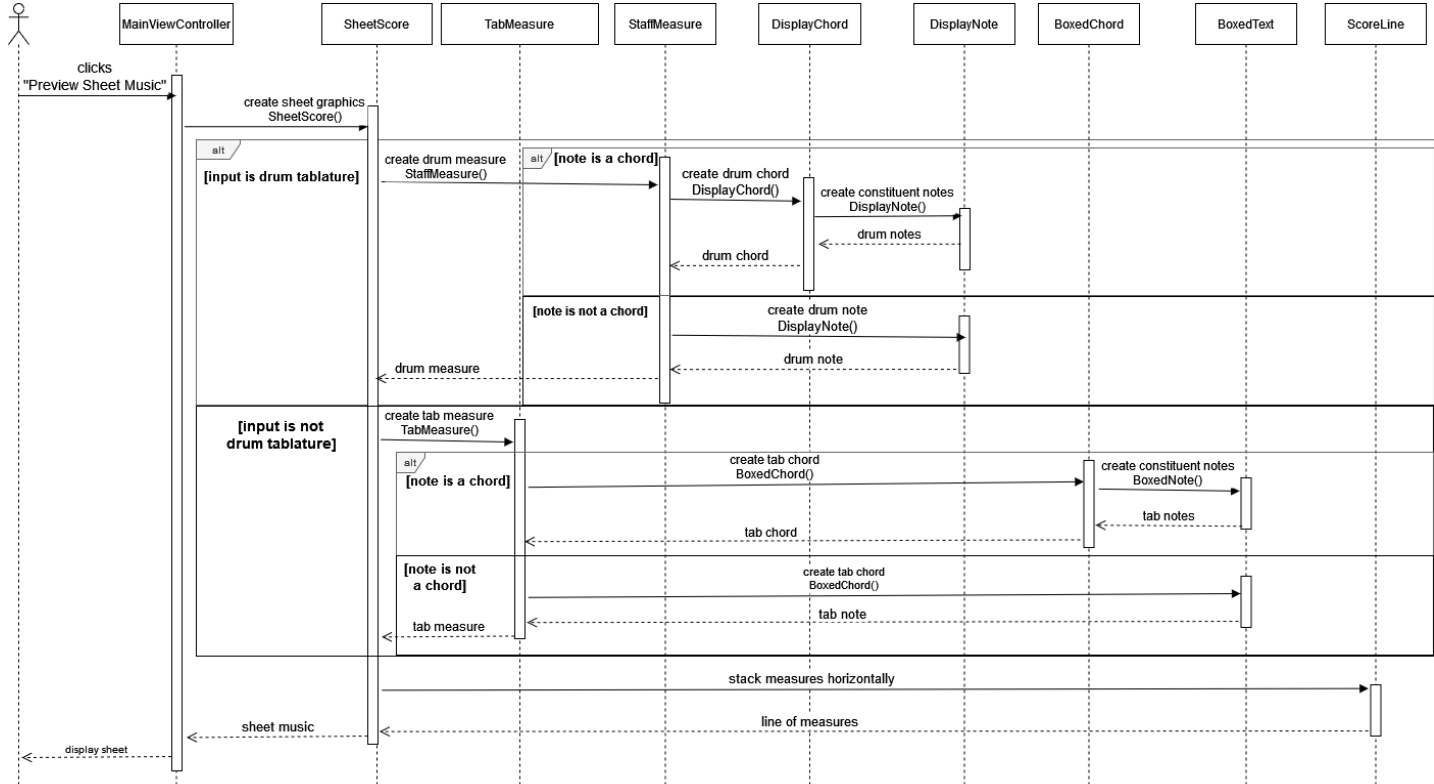# Sequence Diagram

Playing music

Customize Display

Export PDF

Generation of Sheet Music Graphics:

Maintenance Scenarios

I. Play function scenarios

All play features are included in the musicPlayer.java file which is saved in the custom_player package.

    A. If we want to add a new notification about playing music, only need to add a new notification in the musicPlayer.java class.

    B. If we want to change the condition for the play, pause or exit feature, only need to change musicPlayer.java class.

    C. If we want to support new features like playing music from the measure that the user input or the notion user clicked, also the musicPlayer.java is the only class that needs to be changed.

    D. If we want to support more instruments such as piano, celesta, clavinet and so on, we only need to change musicPlayer.java class.

II. Modification of XML Parsing Unit

    A. (1) If a new MusicXML element must be supported, simply create a new class *X* which has the attributes corresponding to the content/attributes of this element. (2) Then, create an object of this element within the corresponding class for the new element's parent. (3) Make sure to store this object within the class it is initialized in, so it can be referenced via an object of this class. (4) The initialization of the attributes of class *X* must be handled by the constructor the class itself which has the single parameter(**doc. Element)** containing the XML element in the tag: <new-supported-element-tag-name>.

B. If previously unsupported attributes/contents of music XML elements must be
supported, then add these attributes to the corresponding Java class in the
custom_component_data package. Make sure to initialize them within the class's
constructor.

III.   Updating the Music Sheet Graphics Generation

A. If a new graphical music element must be supported, and this element applies to
musical notes, then follow the steps below:

1. Create a new class simply generating the graphical component without any
translations. Make sure its constructor has the appropriate arguments and
it extends a JavaFX class.

2. Set and initialize relevant data that may be used (such as its width, or its
enumerated occurrence within a measure)

3. Accordingly, initialize an object of this note using appropriate parameters
within the DisplayNote or DisplayChord class for drums, and the
BoxedText or BoxedChord class for guitars. Then add this object to the
display by "this.getChildren().add(element);" for the new element created.

4. Translate the element horizontally and horizontally based on its position
on the note (The top-left corner of the square around the notehead is the
origin (0,0)).

5. Modify the relevant attributes of the note (if this new element causes the
note to take up a greater width, then increment the width accordingly).
Such values can be "spacing type, width, preceding, trailing" which are
further documented in the project code.