Journal of The Korea Society of Computer and Information
Vol. 19, No. 2, February 2014
www.ksci.re.kr
http://dx.doi.org/10.9708/jksci.2014.19.2.077

MusicXML 기반의 악보 편집기 개발

칸 나지브 울라*, 이 정 철*

Development of a Music Score Editor based on MusicXML

Najeeb Ullah Khan*, Jung-Chul Lee *

요 약

과거 작곡가들은 피아노, 바이올린, 기타, 플룻, 드럼과 같은 고전적 악기나 일반적인 도구들을 이용하여 작곡하였다. 그러나 디지털 기술의 발전으로 많은 소프트웨어 응용 프로그램이 개발되었으며 이제 음악가들은 개인용 컴퓨터를 이용하여 곡들을 제작할 수 있게 되었다. NIFF, SMDL, 그리고 MIDI와 같은 다양한 악보 표현 형식이 제안되었으나 MIDI 형식이 성공적으로 사용되어 왔다. 최근에는 MusicXML 형식이 컴퓨터 음악을 표현하는 사실상 표준으로 부상하였다. 본 논문에서는 MusicXML 형식의 구조에 대한 개요를 기술하고 C# 언어로 MusicXML 기반 악보 편집 소프트웨어를 구현하는 방법을 제안한다. 본 논문에서 제안하는 방법을 이용하여 악보 편집 소프트웨어를를 구현함으로써 구현의 효율성을 보였으며 활용 가능성을 확인하였다.

▸ Keywords : MusicXML, 악보 편집기, 기보

Abstract

In the past composers used to make music with classical instruments such as piano, violin, guitar, flute, drums, and other well-known tools. With the advent of digital technology many software programs were developed which allow musicians to compose tunes using personal computers. Many file formats were introduced such as NIFF, SMDL and MIDI but none besides MIDI has been successful. Recently MusicXML has emerged as a de-facto standard for the computer representation of music. This paper presents a brief description of the structure of the MusicXML format and describes the development of a music score editor based on MusicXML. We implemented a MusicXML-based score editing software using C# language and a feasibility test showed the efficiency of our proposed method.

▸ Keywords : MusicXML, Score Editor, Music Notation

•제1저자 : 나집 울라한 •교신저자 : 이정철
•투고일: 2013. 12. 24, 심사일 :2014. 1. 21, 게재확정일 : 2014. 2. 12.
* 울산대학교 전기공학부(School of Electrical Engineering, University of Ulsan)

I. Introduction

With the advent of digital technology, many commercial tools have been developed for the electronic composition and production of music. Notation editing softwares for professional publishing such as Sibelius and Finale use their own proprietary file formats for the representation of music. Conventionally MIDI was used as an interchange format between these proprietary formats but commercial score editors such as the one mentioned above use about 300 music symbols while MIDI is able to represent about 40 elements[1]. Thus interchange between these softwares used to be a problem. Programs for scanning music scores were able to work only with one proprietary format e.g. scores scanned with SmartScore could only be exported to Finale. Sheet Music available on the internet was also limited to Portable Document Format PDF or to their proprietary formats such as Sunhawk SOLERO and MusicNotes. In the past many attempts have been made to develop a standard music notation format for the interchange of music between these softwares such as Notation Interchange File Format NIFF and Standard Music Description Language SMDL.

NIFF integrates the visual and logical organization of the music into one model. For example the pitch of a note is represented by its position on the staff rather than on a number or alphabet such as a 'C'. Such a format is not suitable for applications such as sequencing. The emergence of markup languages such as the SGML lead to the development of music interchange languages based on the markup languages. The Standard Music Description Language SDML was the first markup language for music. SMDL was so complex that none of the popular commercial softwares adopted it and thus it failed as a music interchange format. Attempts to develop XML based formats were made such as MNML, MusiML, MusiXML and Wedelmusic but none of them succeeded.

Recently MusicXML has emerged as a standard for the interchange of music among notation programs. MusicXML was developed by Recordare LLC in 2000. MusicXML is based on MuseData and Humdrum file format. Currently MusicXML is supported by more than 160 music softwares. The Dolet plug-in can be used to import and export MusicXML files to Finale and Sibelius. The purpose of this paper is twofold. First to give a detailed description of MusicXML and second to describe the development of a music score editor based on MusicXML format.

An introduction to MusicXML can be found in [2], [3] and [4] with descriptions of sample MusicXML files and applications of MusicXML in musical notation, analysis and performance applications. [5] explains the facts behind the success of MusicXML in becoming a de-facto standard for music notation interchange and distribution. Music analysis such as distribution of note durations and analysis of correlation between pitch and duration in a music score using Document Object Model DOM and XQuery are described in [6] and [7].

A complete reference of the MusicXML standard explaining each element of MusicXML can be found on the MusicXML website[8]. However no published work is available which describes the overall structure of MusicXML. The MusicXML XSDs are large enough to be analysed easily. This paper describes the overall structure of MusicXML version 3.0 in detail highlighting the relationships of the most commonly used elements of MusicXML.

There are many commercial and a few open-source music score editors available that supports MusicXML such as MuseScore. However very little published work is available on the development of music score editors. In [9], the authors have developed a MusicXML based music score reader for display of score on a Tablet PC for the visually impaired Musicians. [10] also describes sheet music display on an android Tablet PC using

MusicXML. [11] describes Museflash, a web based music score editor developed using Macromedia Flash. However MuseFlash support an inadequate number of symbols even for an armature musician. Museflash supports only notes from whole to eighth note a flat, and a sharp. It lacks even the basic music elements such as rests, dots and other basic music notation symbols. In this paper we describe the development of a music score editor based on the Windows Presentation Foundation WPF which is Microsoft's next generation graphics interface. Our music score editor is capable of creating, displaying, editing and playing back music using the MusicXML file format supporting a modest number of music notation elements.

The rest of the paper is organized as follows. A description of the MusicXML standard is given in section II. Section III describes the implementation details of the score editor followed by experimental results in section IV. Conclusion is given in section V.

II. MusicXML

1. Introduction

MusicXML is a digital sheet music interchange and distribution format for common Western music notation based on eXtensible Markup Language XML. It was developed by Recordare LLC. The goal of MusicXML is to interchange data between different music software programs such as score editors, optical music recognition softwares and digital music stands etc. The XML schema Definition for MusicXML is freely available. MusicXML is based on two existing music file formats; MuseData which is a general purpose, software independent file format and Humdrum file format designed for research, analysis and music studies[2]. These two file formats have large repertoires available. Basing MusicXML on these formats gives it a strong technical foundation. MusicXML's element and attribute names are based on the English-language musical terms used in the USA[12]. MusicXML is based on XML and thus is internet friendly and non binary format. Although raw MusicXML files are large, they compress well. MusicXML files 7 times larger than Musedata files are only 2 times larger when compressed[6]. Since XML tools are available for every major platform, MusicXML application developer has the freedom to chose the development platform of choice.

2. Structure

Music notation can be divided into three domains; the logical domain which includes the undelying musical representation such as melodies, rythms etc, the visual domain which includes the visual elements such as the stem directions and fonts etc, and the performance domain which includes things such as tempo and loudness of music etc. MusicXML integrate these three domains in such a way that the internal representation is not affected by the visual and performance domain settings. MusicXML uses XML elements for the logical domain while attributes for the visual and performance domain. This is shown in figure 1. Dedicated elements for visual <print> and performance <sound> domains are also included where the attributes are not sufficient.

The Humdrum format on which the design of MusicXML is based, represents music files as a two dimensional plane with succession of events (notes in the same part) proceeds vertically down and concurrent events (corresponding notes in different parts) extends horizontally[13] as shown in figure 2. For this reason MusicXML have two kind of basic structures for music score, one is the timewise score and the other is partwise score as shown in figure 3.

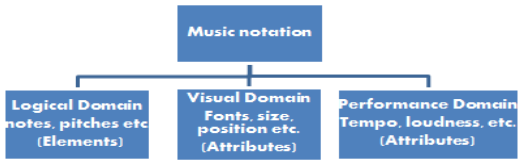

Fig. 1. Domains of Music Notation

	Successive events			
concurrent events	A	K	V	etc
	B	L	W	etc
	C	M	X	etc
	D	N	Y	etc
	etc	etc	etc	

Fig. 2. Humdrum Format

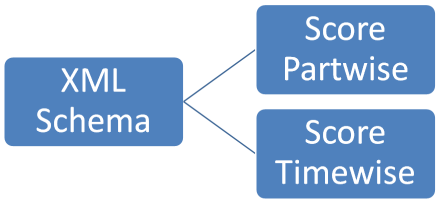

Fig. 3. The two structures of MusicXML score

In the timewise score the measure is the primary element and parts are contained within each measure. In the partwise score the part is the primary element and measures are contained within each part. Automatic convertors are available for the two structures and the contents of both the structures is the same thus we only describe the partwise score in the coming discussion.

The root element of a partwise score is a 〈score-partwise〉 element. It has a version attribute which is the version of MusicXML the score complies to. The 〈score-partwise〉 element contains meta data about the score such as the titles, identification (which include information such as the software with which the score was created etc), it also contain information regarding the default values for the appearance of the score such as the staff size and fonts etc. There is a credit element which specifies typical information to be displayed on the first page of a score. There is a partlist element which enlists all the parts present in the score, it also contains virtual instrument and other MIDI related information for each of the part listed. Finally in the partwise score, there are one or more musical parts present sequentially. All the elements in the Partwise score are optional except the partlist and part elements. The root element of a partwise score is shown in figure 4.

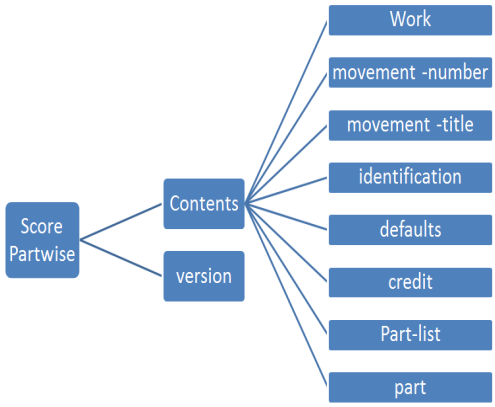

Fig. 4. The Partwise Score

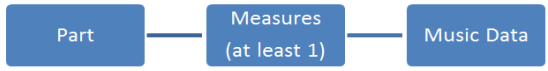

Fig. 5. Measures within Parts

Each part contains measures as shown in figure 5, which in turn contain music data. A measure can contain a variety of music data. Table 1 shows the elements representing the music data inside a measure with brief descriptions. Describing all the elements in detail is beyond the scope of this paper and the interested reader is referred to the MusicXML reference available at the MusicXML website[8]. The note and attributes elements are essential even for the simplest of music applications and will be described in some detail.

A note is the most common music data in the measure. The most common elements inside a note element are shown in table 2 with brief descriptions. The notation element contains elements for

representing graphical symbols associated with a note such as ornaments, articulations, ties, slurs and tuplets etc. The note also contain elements for the representation of cue and grace notes and information about stem directions and the identification of the staff on which the note should be placed in a multi-staff part such as a piano.

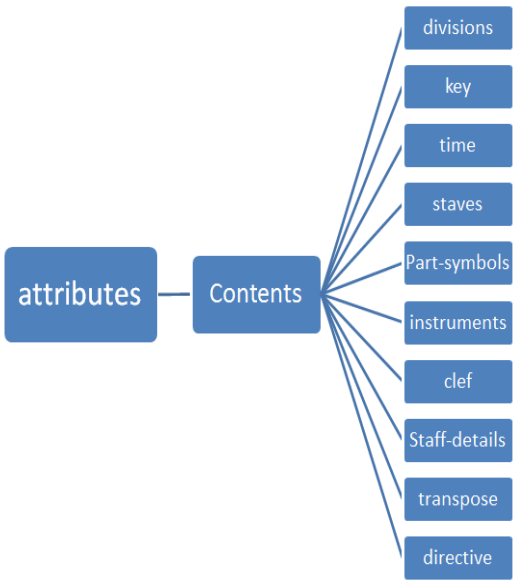

Fig. 6. Contents of attributes element

The attributes element is shown in figure 6. An attributes element is placed inside a measure for specifying information such as key and time signatures, clef, and the unit for the duration element of a note. In MusicXML, the duration of a note or rest is specified by the number of divisions for which the note will be played. The divisions element in the attributes determine the number of divisions per quarter note. Other information such as the number of staffs, graphical details of the staffs, measure styles and transposition of the part to the concert pitch is also encoded in the attributes element. The attribute element is optional and if the current measure do not have an attribute element, then the latest attributes specified will be applicable.

Table 1. Music data inside a measure

Element Name	Description
note	Represents notes
attributes	Contains information that usually change on measure boundaries such as key and time signatures
forward	Coordinates multiple voices in one part
backup	Coordinates multiple voices in one part
direction	A musical indication not attached to a specific note such as a rehearsal mark
harmony	Represents harmony
figured bass	Figured-bass notation
print	General printing parameters
sound	General playback parameters
barline	Represents special barlines
grouping	Used for analysis purposes
link	Serves as an outgoing simple XLink
bookmark	Serves as target for an incoming simple XLink

Table 2. Common note element contents

Element Name	Description
type	Represents the type of note e.g half, quarter
duration	the actual duration of time for which the note will be played
dot	Represents a dot
rest	Indicates a rest
pitch	represents pitch as step octave and alteration due to accidentals
accidental	Represents a graphical accidental e.g. flat, sharp
chord	Represents that the present note is a chord tone to the preceeding note
time modification	represents changes in duration due to tuplet notes
beam	Represents beaming of the note
notations	represents a wide variety of graphical notation elements e.g tied, slur etc
Lyric	Represents the lyric associated with the note

3. Example

Figure 7 shows a simplified MusicXML file with a partwise score containing a single part with one measure and two notes. The corresponding music notation is shown in figure 8. The partwise score contains an identification element followed by the partlist. The score contains only one part which contains only one measure. The measure contains an

attributes element and two notes. A value of 4 for the divisions in the attributes element means there are four divisions in a quarter note so a quarter note has a duration of 4 divisions, a half note has a duration of 8 divisions and an eighth note has a duration of 2 divisions. Inside the Key element the fifths element specifies the number of accidentals in the key signature. The flats are represented by a negative number and the sharps are represented by a positive number. In the example there are no flats or sharps in the key signature thus the fifths element is zero. The mode element specifies whether the scale is major or minor. In the time element, the beats element specifies the number of beats per measure while the beat type element is used to specify which note type constitutes a beat. The value 4 signifies a quarter note. The clef sign is the type of clef while the line number determines at which line of the staff the clef is placed counting from the bottom line of the staff. The listing shows a clef sign of G placed at the second staff line.

Figure 8 shows a duration of 12 divisions for the first note. The type element specifies that this is a half note which according to the defined divisions in attributes should have a duration of 8 divisions. This addition of 4 divisions is because of the reason that this is a dotted note. The dot adds half of its original duration (4) to its duration (8) resulting in a value of 12. The dot is specified by an empty dot element. The second note in figure 7 is a quarter note with a sharp accidental.

```
<score-partwise>
  <part-list>
    <score-part id="P1">
      <part-name>Violin</part-name>
      <part-abbreviation>Vln.</part-abbreviation>
      <score-instrument id="P1-I3">
        <instrument-name>Violin</instrument-name>
      </score-instrument>
      <midi-instrument id="P1-I3">
        <midi-channel>1</midi-channel>
        <midi-program>41</midi-program>
      </midi-instrument>
    </score-part>
  </part-list>
  <part id="P1">
    <measure number="1" width="152.09">
      <attributes>
        <divisions>4</divisions>
        <key>
          <fifths>0</fifths>
          <mode>major</mode>
        </key>
        <time>
          <beats>4</beats>
          <beat-type>4</beat-type>
        </time>
        <clef>
          <sign>G</sign>
          <line>2</line>
        </clef>
      </attributes>
      <note>
        <pitch>
          <step>A</step>
          <octave>4</octave>
        </pitch>
        <duration>12</duration>
        <voice>1</voice>
        <type>half</type>
        <dot/>
      </note>
      <note>
        <pitch>
          <step>C</step>
          <alter>1</alter>
          <octave>5</octave>
        </pitch>
        <duration>4</duration>
        <voice>1</voice>
        <type>quarter</type>
        <accidental>sharp</accidental>
      </note>
    </measure>
  </part>
</score-partwise>
```

Fig. 7. MusicXML example file

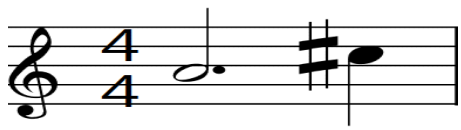

Fig. 8. Representation of MusicXML file in figure 7

Table 3. Symbols

Symbol	Description
Notes	whole to 32nd
Rests	whole to 32nd
dot	single
Accidental	Flat, sharp, natural
Lyrics	Yes
Slur	Yes
Tie	Yes
Clefs	G, F, C
Time sign	Yes
Key sign	Yes

III. Implementation

Music notation involves a lot of symbols. Although MusicXML represents common western music notation from the seventeenth century onwards, it has more than 400 elements. Only the most commonly used symbols shown in table 3 were selected for the music score editor.

A music score editor or reader involves heavy use of custom graphics (such as displaying shapes for slurs, beams, ties etc). In contrast to just displaying music, score editors involve user interaction (a staff line is not just a graphic shape but should respond to a user click). The score editor was implemented in Microsoft C#.Net. There are two options available for Graphical user interfaces in C#; one is Windows Forms and the other is Windows Presentation Foundation WPF. Windows Forms provides a lot of easy to use controls such as buttons and textboxes etc. It also provides GDI+ graphics, which is based on Windows Graphics Device Interface. GDI+ contains classes to create graphics, draw text, and manipulate graphical images as objects[14]. Pixels are used as the units in Windows Forms and thus size is dependent on the display device resolution. The core of WPF is a resolution-independent and vector-based rendering engine that is built to take advantage of modern graphics hardware. WPF extends the core with a comprehensive set of application-development features that include Extensible Application Markup Language (XAML), controls, data binding, layout, 2-D and 3-D graphics, animation, styles, templates, documents, media, text, and typography[15]. The shape objects in WPF such as lines, elipses etc are interactive i.e. they have events (such as mouse button click events etc). A comparison of Windows Forms and WPF is given in table 4. A music score editor involves interactive graphics which should be zoom-able as well. It turns out that WPF is the ideal choice for such an application. Based on this discussion we have chosen WPF as the platform for the development of the music score editor.

Table 4. WinForms Vs WPF

	WPF	WinForms
Graphics	DirectX	GDI
Units	Device Independent	Pixels
Scalable	Yes	No
Shapes	Interactive	Static
Browser Hosting	Yes	No

Figure 9 shows the different modules used in the music score editor. Each module consists of one or more classes.

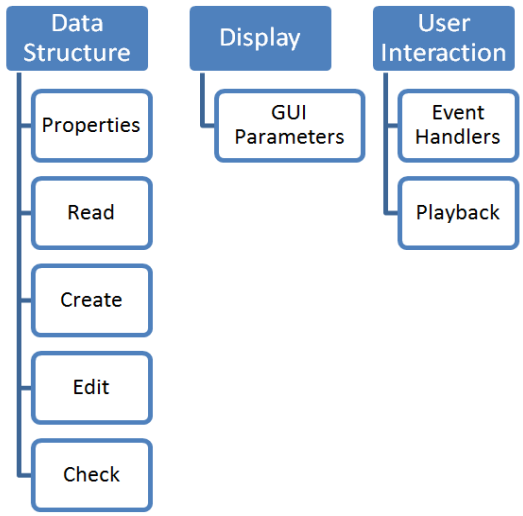

Fig. 9. Music Score Editor Modules

1. Data Structure

To develop a score editor based on MusicXML, we first need to be able to parse MusicXML files. Microsoft .Net provides support for reading and writing XML document using the Document Object Model DOM. Although the DOM approach is very flexible conforming to the schema of MusicXML is a difficult task when writing new MusicXML files. An Alternate method which utilizes the XML serialization can be used to serialize and deserialize C# objects to XML and XML into C# objects. This method requires availability of a class whose objects we will convert to MusicXML and vice versa. Microsoft Visual Studio provides a tool called XSD Tool to convert the XML Schema Definition file into a class[16]. Since MusicXML XSD is freely available, we used the XSD tool to generate the C# classes corresponding to the elements of MusicXML. This process is shown in figure 10 below. However, there was one problem with the data types in the automatically generated classes. The sequence of elements in MusicXML XSD was translated to C# array type which cannot be dynamically resized; however for score editor we need a dynamic data structure such as a list. The problem discussed above can be solved by using a java library named ProxyMusic. ProxyMusic provides a binding between Java objects in memory and data formatted according to MusicXML[17]. This library solves the data structure problems and contains all the classes corresponding to MusicXML elements. ProxyMusic provides utility methods for reading and writing MusicXML files.

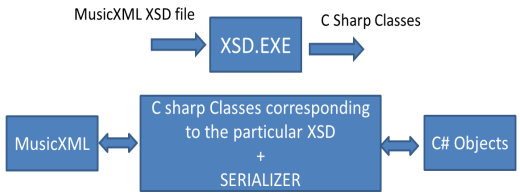

Fig. 10. MusicXML Reading and Writting using C# XML serialization

To use ProxyMusic (being a java library) in C# we used the IKVM.NET tool. IKVM.NET is an implementation of Java for Mono and the Microsoft .NET Framework. It includes a Java Virtual Machine implemented in .NET [18]. There are two methods the IKVM.NET can be used. ProxyMusic is statically compiled to a DLL and is used directly in C#. Since the classes and methods in ProxyMusic has inputs and outputs of Java data types thus the Java classes and data types are used dynamically for input output operations with ProxyMusic.

The ProxyMusic since generated from the MusicXML XSD has the same nested structure as MusicXML. This makes it very difficult to use ProxyMusic directly for the internal representation of notes and measures in the score editor. Figure 11 shows an example of accessing information about the first note in the first measure of a part. As can be seen in figure 11, the measure contains various music data objects such as notes, attributes etc, the list of these objects is accessed by the function getNoteOrBackupOrForward(). So there is no explicit indexing for notes in a measure using ProxyMusic. We search for the first occurrence of a note object in the list and then access its different properties.

```
//Access the first part in the score
ScorePartwise.Part FirstPart =
   (ScorePartwise.Part)scorePartwise.getPart().get(0);

//Access the first measure in the Part
ScorePartwise.Part.Measure FirstMeasure =
   (ScorePartwise.Part.Measure)FirstPart.getMeasure().get(0);

//Check the Music data inside the measure and look for the first note
Note FirstNote=null;
for(int i =0; i < FirstMeasure.getNoteOrBackupOrForward().size(); i++)
{
   if (FirstMeasure.getNoteOrBackupOrForward().get(i) is Note)
   {
      FirstNote = (Note)FirstMeasure.getNoteOrBackupOrForward().get(i);
      break;
   }

}
//Get the pitch of the note
string step = FirstNote.getPitch().getStep().value();
//Get the duration of the note
int duration = FirstNote.getDuration().intValue();
```

Fig. 11. ProxyMusic Example

```
public class AttributeProperties
{
    public string fifths = "";
    public string mode = "";
    public string TimeBeats = "";
    public string BeatType = "";
    public string ClefSign = "";
    public string ClefLine = "";
    public int ClefStaff = 1;
    public int divisions = 0;
    public int staves = 1;
}
```

Fig. 12. Class for Attributes Element

Many of the elements in MusicXML and consequently in ProxyMusic are optional and thus before reading the properties of an element, we first need to check if the properties are present or not. For example in a note there may or may not be a duration property and we may need to determine the duration of the note based on the divisions and note type. For the music score editor we need an indexed data structure with some level of uniformity. Developing such a data structure allowed us to separate the content of MusicXML from the display aspects of the music score editor. Several C# classes were written each representing the important properties of elements such as the score itself, the measures within the score, the notes and attributes within the measure. Figure 12 shows the class to represent the 〈attributes〉 element of MusicXML. Classes were developed to read these properties from ProxyMusic data structure (ReadDS.cs), create new elements based on the given properties (CreateDS.cs) and edit the properties of the existing elements (EditDS.cs).

A class was created for checking and correcting the score for errors if there are more notes than allowed by the time signature in a measure.

2. Display

WPF uses an automatic layout system using panels, such as the Grid panel in which child controls are positioned by rows and columns, the StackPanel in which child controls are stacked either vertically or horizontally, the WrapPanel in which the child controls are positioned in left-to-right order and wrapped to the next line when there are more controls on the current line than space allows and the Canvas in which child controls provide their own layout[15]. To display the different elements of the music score and at the same time make them editable, we have adopted a layered approach to display the score.

Figure 13 shows the different panels to display the score. Inside the root Grid there are two rows containing the ribbon control for the user interaction and the ScrollViewer. The ViewBox inside the ScrollViewer contains all the graphic data of the music score. The ViewBox can scale its child element when it's size is changed, thus ViewBox provide us the zooming functionality for the score while the ScrollViewer allows us to scroll the score. A vertical StackPanel is used as the panel for displaying the staffs. Above which a WrapPanel is used for the display of notes, rests and other associated symbols such as dots and accidentals. A third layer that is a Canvas, is used to display the inter-note symbols such as the beams, ties and slurs etc. To display the music symbols, we used the Bach Musicological Font[19] which is a TrueType Font. Bach font do not represent the pitch information of the notes thus the vertical position of each note is controled by the pitch value of the note.

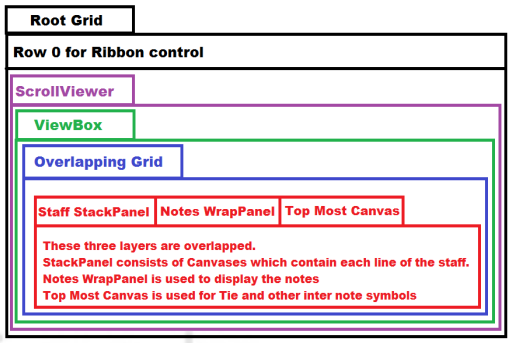

Fig. 13. GUI Architecture

However, with different clef signs, the names of the staff lines changes for example the top line of staff with a G clef is an "F" but with a C clef it correspond to a "G". This positioning of notes is performed by the GUIPara class.

3. User Interaction

The music score editor uses the Ribbon Control for user commands rather than the traditional menu bar and toolbars. The ribbon is a command bar that organizes the features of an application into a series of tabs at the top of the application window. The ribbon user interface (UI) increases discoverability of features and functions, enables quicker learning

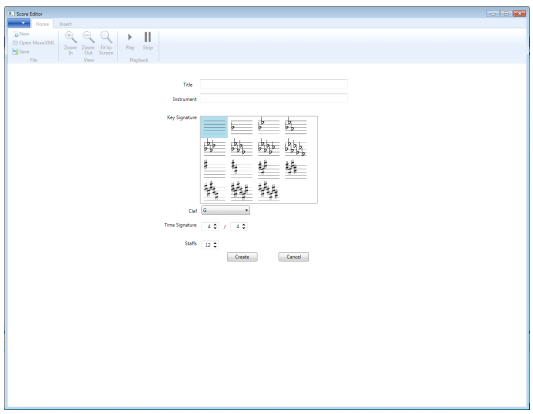

Figure 14. Creating a new score

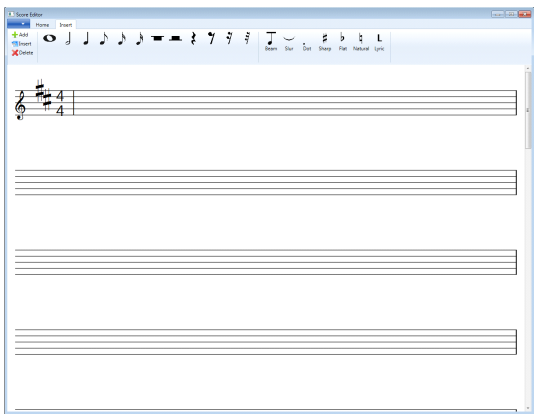

Figure 15. A new score displayed

of the application, and makes users feel more in control of experience with the application[20-22]. Figure 14 shows the window for creating a new music score. Figure 15 shows the music score editor with a new score opened. The ribbon control has two tabs. Home and Insert tab. Home tab contains buttons for operations such as creating a new score, opening an existing MusicXML file and saving a score to MusicXML file as well as zooming and playback operations. The Insert tab contains buttons for notes, rests and other symbols as well as for editing these symbols. Event handlers were used to respond to user actions such as entering new notes, editing and deleting the existing ones etc. Each of the actions first makes the changes to the underlying data structure and then to the display. Currently only mouse events are implemented. Notes are created by selecting the note symbol from the Insert Tab in the ribbon control and clicking the desired staff line. Each of the staff line has its associated event handler which identifies the staff line clicked. Using the staff line clicked, the selected note type, the clef sign and the time signature, the properties for the note are generated and added to the ProxyMusic data structure and the display is updated.

The playback class uses the Windows Multimedia API winmm to playback the notes. The playback class has methods to extract pitch, duration and velocity information from MusicXML score and play it.

IV. Results

We implemented a music score editor based on MusicXML 3.0 using visual C# windows WPF on PC. Figure 16 shows a MusicXML file displayed by our music score editor. The music socre editor formats the music according to the rules of music notation. Each staff line starts with the clef sign followed by the key signature(if any). The number of measures per staff depends on the notes in the measures. As can be seen in figure 16, the first two staffs contains 5 measures each while the third staff contain 4

measures. The same file displayed using the MuseScore editor is shown in figure 17. Figure 16 has better appearance as compared to figure 17, however it lacks certain symbols such as the up and down bows. The spacing between notes in figure 16 is more uniform as compared to that of figure 17.

To demonstrate the creation of MusicXML file and the editing capabilities of the score editor, figure 18 shows a score with four notes added, followed by figure 19 in which the first note in the measure is deleted. Figure 20 finally shows the insertion of a half note after the eighth note. Parts of the MusicXML files for figures 18, 19, 20 are shown in figures 21, 22, 23 respectively.

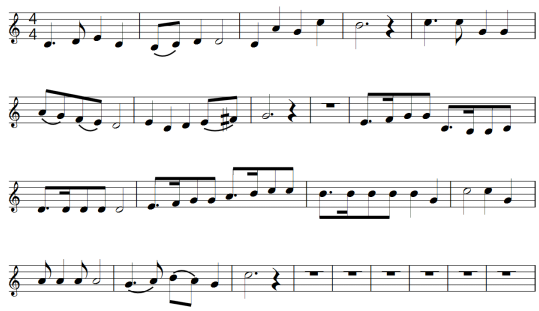

Figure 16. Displaying score using the music score editor

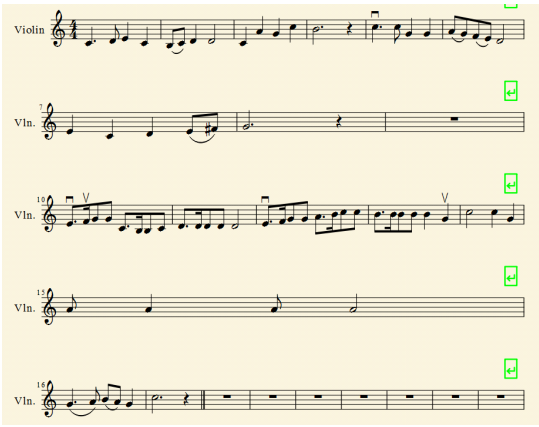

Figure 17. Displaying score using MuseScore

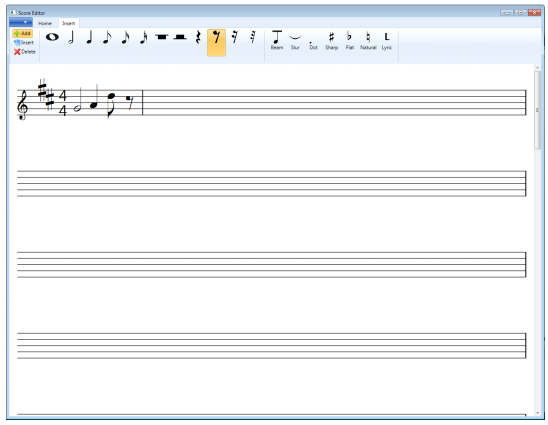

Figure 18. Notes added to the score

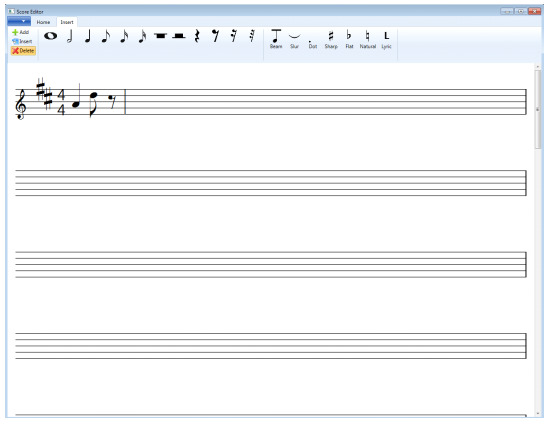

Figure 19. The first note deleted

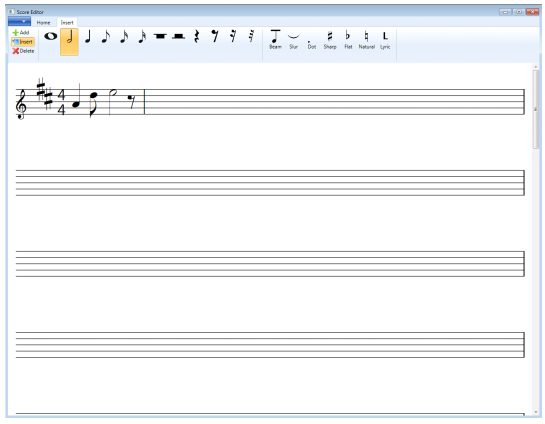

Figure 20. A half note inserted after the eighth note

```
<part id="P1">
    <measure number="1">
        <attributes>
            <divisions>8</divisions>
            <key>
                <fifths>2</fifths>
                <mode>major</mode>
            </key>
            <time>
                <beats>4</beats>
                <beat-type>4</beat-type>
            </time>
            <staves>1</staves>
            <clef number="1">
                <sign>G</sign>
                <line>2</line>
            </clef>
        </attributes>
        <note>
            <pitch>
                <step>G</step>
                <octave>4</octave>
            </pitch>
            <duration>16</duration>
            <voice>1</voice>
            <type>half</type>
            <staff>1</staff>
        </note>
        <note>
            <pitch>
                <step>A</step>
                <octave>4</octave>
            </pitch>
            <duration>8</duration>
            <voice>1</voice>
            <type>quarter</type>
            <staff>1</staff>
        </note>
        <note>
            <pitch>
                <step>D</step>
                <octave>5</octave>
            </pitch>
            <duration>4</duration>
            <voice>1</voice>
            <type>eighth</type>
            <staff>1</staff>
        </note>
        <note>
            <rest/>
            <duration>4</duration>
            <voice>1</voice>
            <type>eighth</type>
            <staff>1</staff>
        </note>
    </measure>
</part>
```

Figure 21. MusicXML file made by the score editor for the score shown in figure 18

```
<part id="P1">
    <measure number="1">
        <attributes>
            <divisions>8</divisions>
            <key>
                <fifths>2</fifths>
                <mode>major</mode>
            </key>
            <time>
                <beats>4</beats>
                <beat-type>4</beat-type>
            </time>
            <staves>1</staves>
            <clef number="1">
                <sign>G</sign>
                <line>2</line>
            </clef>
        </attributes>
        <note>
            <pitch>
                <step>A</step>
                <octave>4</octave>
            </pitch>
            <duration>8</duration>
            <voice>1</voice>
            <type>quarter</type>
            <staff>1</staff>
        </note>
        <note>
            <pitch>
                <step>D</step>
                <octave>5</octave>
            </pitch>
            <duration>4</duration>
            <voice>1</voice>
            <type>eighth</type>
            <staff>1</staff>
        </note>
        <note>
            <rest/>
            <duration>4</duration>
            <voice>1</voice>
            <type>eighth</type>
            <staff>1</staff>
        </note>
    </measure>
</part>
```

Figure 22. MusicXML file made by the score editor for the score shown in figure 19

```
<part id="P1">
    <measure number="1">
        <attributes>
            <divisions>8</divisions>
            <key>
                <fifths>2</fifths>
                <mode>major</mode>
            </key>
            <time>
                <beats>4</beats>
                <beat-type>4</beat-type>
            </time>
            <staves>1</staves>
            <clef number="1">
                <sign>G</sign>
                <line>2</line>
            </clef>
        </attributes>
        <note>
            <pitch>
                <step>A</step>
                <octave>4</octave>
            </pitch>
            <duration>8</duration>
            <voice>1</voice>
            <type>quarter</type>
            <staff>1</staff>
        </note>
        <note>
            <pitch>
                <step>D</step>
                <octave>5</octave>
            </pitch>
            <duration>4</duration>
            <voice>1</voice>
            <type>eighth</type>
            <staff>1</staff>
        </note>
        <note>
            <pitch>
                <step>E</step>
                <octave>5</octave>
            </pitch>
            <duration>16</duration>
            <voice>1</voice>
            <type>half</type>
            <staff>1</staff>
        </note>
        <note>
            <rest/>
            <duration>4</duration>
            <voice>1</voice>
            <type>eighth</type>
            <staff>1</staff>
        </note>
    </measure>
</part>
```

Figure 23. MusicXML file made by the score editor for the score shown in figure 20

V. Conclusion

MusicXML has emerged as a de-facto standard for music notation interchange and distribution. We have provided a broad overview of the MusicXML elements. The ease of using ProxyMusic rather than a parser using DOM is demonstrated. The use of WPF for the development of music score editor turned out to be a scalable solution. The automatic layout system of WPF allows for efficient editing as opposed to the bitmap based approach in which the whole music score must be repainted if a single note is to be changed. Although the music score editor has only the most commonly used symbols, it can readily be extended given the complete coverage of MusicXML by ProxyMusic and the power of WPF. **The support for multiple part scores as well as advanced notation elements such as articulations and ornaments will be added in a future version.** The score editor has enough symbols to be used by researchers, amateur musicians and educators.

VI. Acknowledgment

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (2011-0015125)

References

[1] P. Bellini and P. Nesi, "WEDELMUSIC format: An XML music notation format for emerging applications." Proc. WEDELMUSIC 2001, pp. 79 - 86, Nov. 2001.

[2] S. Cunningham, "Suitability of MusicXML as a Format for Computer Music Notation and Interchange." Proceedings of IADIS Applied Computing 2004 International Conference, Lisbon, Portugal. pp. III-7, 2004.

〔3〕 M. Good, "MusicXML in Commercial Applications." *Music Analysis East and West: Computing in Musicology 14,* edited by W. B. Hewlett and E. Selfridge-Field, MIT Press, pp. 9-20. 2006,
〔4〕 M. Good, "MusicXML: An Internet-Friendly Format for Sheet Music." *Proceedings of XML 2001 Conference,* pp. 9-14, Dec, 2001
〔5〕 M. Good, "Lessons from the Adoption of MusicXML as an Interchange Standard." *Proceedings of XML 2006 Conference*, pp. 1-13, Dec. 2006
〔6〕 M. Good, "MusicXML in Practice: Issues in Translation and Analysis." Proceedings of First International Conference MAX 2002, pp. 47-54, Sep. 2002
〔7〕 M. Good. "MusicXML for notation and analysis." *The virtual score: representation, retrieval, restoration,* edited by W. B. Hewlett and E. Selfridge-Field, MIT Press, pp. 113-124, 2001
〔8〕 "MusicXML XSD Schema Reference" available at *http://www.musicxml.com/*
〔9〕 L. Housley, T. Lynch, R, Ramnath,P.F. Rogers, J. and Ramanathan, J, "Implementation Considerations in Enabling Visually Impaired Musicians to Read Sheet Music Using a Tablet." Proceedings of IEEE COMPSAC, pp. 678 - 683, Jul. 2013.
〔10〕 L.L. Housley, *Dynamic Generation of Musical Notation from MusicXML Input on an Android Tablet*. Doctoral Dissertation. Ohio State University, 2012.
〔11〕 Cunningham, Stuart, et al. "Web-based Music Notation Editing." Proceedings of IADIS- International Conference on WWW/Internet, Murcia, Spain. 2006.
〔12〕 M.D. Good, "MusicXML: The First Decade." *Structuring Music Through Markup Language: Designs and Architectures*, MakeMusic, Inc., pp. 187-192, 2012
〔13〕 "The Humdrum Syntax" available at http://www.music-cog.ohio-state.edu/Humdrum/guide05.html
〔14〕 "Windows Forms Overview" available at *http://msdn.microsoft.com/en-us/library/8bxxy49h(v=vs.110).aspx*
〔15〕 "Introduction to WPF" available at http://msdn.microsoft.com/en-us/library/aa970268(v=vs.110).aspx
〔16〕 "XML Schema Definition Tool (Xsd.exe)" available at *http://msdn.microsoft.com/en-us/library/x6c1kb0s(v=vs.110).aspx*
〔17〕 ProxyMusic *https://proxymusic.kenai.com/*
〔18〕 J. Frijters, "IKVM, an implementation of Java for Mono and the. NET Framework," Available at *http://www.ikvm.net*
〔19〕 Yo Tomita. "Bach, Musicological Font" available at *http://www.mu.qub.ac.uk/tomita/bachfont/*
〔20〕 "Ribbon (WPF)" available at *http://msdn.microsoft.com/en-us/library/*
〔21〕 G.J. Lim and J.C. Lee, "An Electronic Keyboard Instrument Using PC MIDI and USB Interface," Journal of The Korean Society of Computer and Information, vol.16, no.11, p85-93, Nov. 2011.
〔22〕 J.Y. Kim, J.C. Lee, and H.S. Jun, "Development of PC based flute performance learning software," Journal of The Korean Society of Computer and Information, vol.18, no.2, p95-105, Feb. 2013.

저 자 소 개

Najeeb Ullah Khan
2012년 : International Islamic University, Islamabad 학사
2013~현재: 울산대학교 전기공학부 석사과정
관심분야: Digital signal processing, Speech recognition/synthesis,
Email : electronicengr@gmail.com

이 정 철
1984년 : 서울대학교 전자공학과 공학사
1988년 : 서울대학교 전자공학과 공학석사
1998년 : 서울대학교 전자공학과 공학박사
현 재: 울산대학교 전기공학부 부교수
관심분야 : 디지털신호처리, 음성신호처리, 음성합성
Email : jungclee@ulsan.ac.kr