8-8-2005

ScoreSVG: A New Software Framework for Capturing the Semantic Meaning and Graphical Representation of Musical Scores Using JAVA2D, XML, and SVG

Geoffrey Alan Bays

SCORESVG: A NEW SOFTWARE FRAMEWORK FOR CAPTURING THE SEMANTIC MEANING AND GRAPHICAL REPRESENTATION OF MUSICAL SCORES USING JAVA2D, XML, AND SVG

by

GEOFFREY BAYS

Under the Direction of Ying Zhu

ABSTRACT

ScoreSVG implements a three-tiered software architecture that generates musical scores in scalable vector graphics (SVG), something no other known music editor has done. SVG is non-proprietary XML-based format that renders graphical content into a web browser equipped with a SVG plugin. Scores in SVG can be scripted to produce interactive music theory examples, or make other graphical score changes impossible in any other format. Large music score editors such as Finale and Sibelius output to proprietary file formats, or Postscript (.ps) files that are not interchangeable or modifiable once created. Open source efforts such as LilyPond or Guido require the user to learn a new text-based music format to get a PostScript music score. ScoreSVG converts the data from a user-friendly visual front end to GuidoXML, capturing the semantic meaning of the score, and then outputs the result in SVG using an XSL stylesheet and the Saxon 8.4 XSLT processor.

INDEX WORDS: SVG, Music Notation, Music Editing, Java 2D Graphics, XSLT, Guido, XML

SCORESVG: A NEW SOFTWARE FRAMEWORK FOR CAPTURING THE SEMANTIC MEANING AND GRAPHICAL REPRESENTATION OF MUSICAL SCORES USING JAVA2D, XML, AND SVG

by

GEOFFREY BAYS

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of

Master of Science

Georgia State University

August 2005

SCORESVG: A NEW SOFTWARE FRAMEWORK FOR CAPTURING THE SEMANTIC MEANING AND GRAPHICAL REPRESENTATION OF MUSICAL SCORES USING JAVA2D, XML, AND SVG

by

GEOFFREY BAYS

Major Professor:	Ying Zhu
Committee:	Rajsekhar Sunderaman
	Xaolin Hu

Electronic Version Approved:

Office of Graduate Studies
College of Arts and Sciences
Georgia State University
August 2005

DEDICATION:

In loving memory of my father, Robert A. Bays, who exemplified a passion for wide-ranging learning that propelled me in this endeavor.

ACKNOWLEDMENTS:

Firstly to my thesis advisor, Ying Zhu for his advice, encouragement, and for pointing me toward the SVGOpen 2005 conference where I will present a paper based on this thesis.

Secondly, I would like to thank Xaolin Hu for looking over the software engineering aspects of ScoreSVG and making helpful suggestions, and Rajsekhar Sunderraman for his general graduate advice through my years at Georgia State University.

Lastly, but by no means least, I would like to thank my wife, Deborah, for her great patience during my years of computer science study.

TABLE OF CONTENTS

Page

LIST OF FIGURES:

LIST OF EXAMPLES:

LIST OF ABBREVIATIONS:

GUI	Graphical User Interface
SVG	Scalable Vector Graphics
UI	User Interface
XML	eXtensible Markup Language
XSL	eXtensible Stylesheet Language
XSLT	eXtensible Stylesheet Language:Transformations
W3C	World Wide Web Consortium
WYSIWYG	What You See Is What You Get

CHAPTER 1: Introduction

The impetus for creating another music score editor came from frustration as a relatively impoverished graduate student in music some years ago, upon finding out that commercial score editors such as Finale and Sibelius were around $300 in cost even for the student or educators version. This alone would not have sufficed for work on a new software framework, though, had SVG not arrived and started to gain acceptance and visibility as a great open source standard for web-based graphics. The advent of SVG combined with the lack of any music score editing software that output scores in SVG led directly to the development of Score SVG. ScoreSVG aims to provide a user-friendly, open source music score editor and software framework that records the musical semantics of a score as well as the graphical output in the non-proprietary and configurable SVG format.[29]

As XML continued to become ubiquitous and all-pervasive, researches into music and XML followed, which brought in turn exposure to other non-XML text-based representations of music scores, Guido music notation and LilyPond in particular.[5] [8] The completeness and compactness of the Guido system was impressive, and the GuidoXML developed from it was similarly compact and human-readable, unlike the more widespread MusicXML, which is grossly verbose and solely machine-readable. [14]

The key rationale for incorporating this XML into the ScoreSVG framework at all, lay in the XML format capacity to record the semantic meaning of the musical score. One could just output the score in SVG, but then only the graphical representation of the score is captured. With the semantics of the score output into XML, analyses of all sorts could be unleashed upon the score, perhaps even an automated Schenkerian analysis.

Music theoretical problems and analyses were definitely considered for the development of ScoreSVG. The capability of SVG to yield interactive scores through scripting is one of the most attractive features of the format. This will make possible music theory problems that can be solved right in a browser. For performance or presentation purposes, scores could have particular notes or other elements colored, and the colors changed by the user. These kinds of possibilities make ScoreSVG a uniquely capable framework to build upon.

In addition to music theory and pedagogical purposes, another rationale for ScoreSVG was to create an open source score editor that is easy to use. Thus a WYSIWYG editor written in Java was created for the front end to ScoreSVG. It is easy to use, and does not require the user to learn a new text-based music notation system as do LilyPond and Guido.

Having decided to create a music score editor that was user-friendly on the front end, captured the semantics of the score with XML, and output to the highly configurable and non-proprietary SVG standard, the development process began. In the following chapters each aspect of ScoreSVG is investigated in turn, from the front end to the backend. Chapter 2 describes the background to ScoreSVG, explaining the nature of SVG, how it works, how it is viewed, and the rationale for outputting musical scores in it. A number of music score editors and music XML formats are reviewed as related research. Chapter 3 presents the architecture of ScoreSVG. First, the overall architecture is considered, then the software engineering process with UML diagrams for each unique software system. Chapter 4 presents the user interface and its design, without describing all the user interactions with it. This facet is saved until chapter 8, the user guide. Chapter 4 also explains the development strategy and programming features of the front end. Chapter 5 relates how a musical font is embedded in

both the front end and backend of ScoreSVG and the software and processes necessary to make that happen. Chapter 6 investigates the way that the system generates XML from the Java objects and gives examples of the modified GuidoXML used by ScoreSVG. Chapter 7 explains the XSL stylesheet used to generate an SVG output and how it was designed and additional possibilities with it. The use, embedding, and possibilities with the Saxon 8.4 XSLT processor are discussed as well. Chapter 8 describes how the user can interact with ScoreSVG at its current state of development, and likely additions to ScoreSVG are mentioned here as well. Chapter 9 briefly details how ScoreSVG was assembled for delivery and how both the musical font and Saxon were embedded in it. Finally, in chapter 10, various possibilities for ScoreSVG are discussed, from obvious immediately necessary features to more challenging avenues of exploration.

Chapter 2: Background

2.1 The SVG Format: Characteristics and Capabilities

SVG is a very attractive graphics format because it rests on a non-proprietary standard defined by the W3C, because it is a vector-based graphics format, and because it is a kind of XML format with a DOM tree that is navigable with many scripting languages. The W3C defines SVG as follows: “SVG is a language for describing two dimensional graphics and graphical applications in XML”. [23] SVG includes a great variety of features that are probably not needed for music scores, such as layering, filter effects, and animation. In this section the features of SVG needed for music scores will be examined as well as general characteristics that distinguish SVG from other graphics formats. [1]

As a vector-based graphics format, SVG allows images to be stored in compact text-based files, and allows those image to be transformed, especially enlarged, without any deterioration of image quality since by definition, the images are delineated by describing vectors, or points between which lines or shapes are drawn. A common file format used by music score editors like Finale and Sibelius is PostScript (.ps). PostScript images are textual vector-based graphics, but PostScript is low level and very complex, and thus unfriendly to manipulate directly. In contrast, SVG files are easy for people to write and change; even graphic artists with no programming background can easily learn to create SVG images without any other software. The alternative graphics formats, such as JPEG, PNG, TIFF, are bit-mapped, and are weighty binary formats good for photographs and print media, but less optimal when compressed for web usage. These images deteriorate when enlarged and compressed.

The SVG specification also includes a built-in event handling system for mouse events that is much easier to use than JavaScript. Though this does not add any capabilities that JavaScript cannot accomplish, it enables many likely changes to an SVG image to be done quite easily. The capacity to make user-interactive changes to an SVG score through scripting remains one of SVG's most salient features. [1] [25]

Support and viewability of SVG are now at a rather high point due to Adobe Software's widely distributed SVG browser plugin which ships with Acrobat Reader and other Adobe software.[18] Many graphics programs now support SVG in some capacity, even if only as a export option. Being viewable in a browser means that specialized or expensive software is not needed to view SVG. The Apache Batik project supplies an open source SVG viewer, conversion tools for TrueType fonts to SVG fonts, and conversion classes that

convert Java2D graphics into SVG. [19] All of these facts are indicators that SVG will continue to grow in popularity, making it a logical choice for music score output.

Features of SVG that are used by ScoreSVG are not great in number, but it is instructive to cover them here. At the most basic level, SVG is capable of rendering basic geometric shapes such as lines, circles, ellipses and rectangles. These are simply specified by coordinates relative to a coordinate system with the upper left corner given as (0,0). For example, a black line with stroke-width 3 and an ellipse with a green outline and red fill, 20 pixels wide and 10 pixels high are shown below, first as SVG code, then in the browser :

```
<svg width="200px" height="200px" viewBox = "0 0 200 200">
    <line x1="30" y1="10" x2="120" y2="10" style="stroke-width: 3;  stroke: black;"/>
   <ellipse cx="70" cy="80" rx="40" ry="20" style="stroke: green;stroke-width: 3;fill: red;"/>
</svg>
```

Example 1: SVG line and ellipse

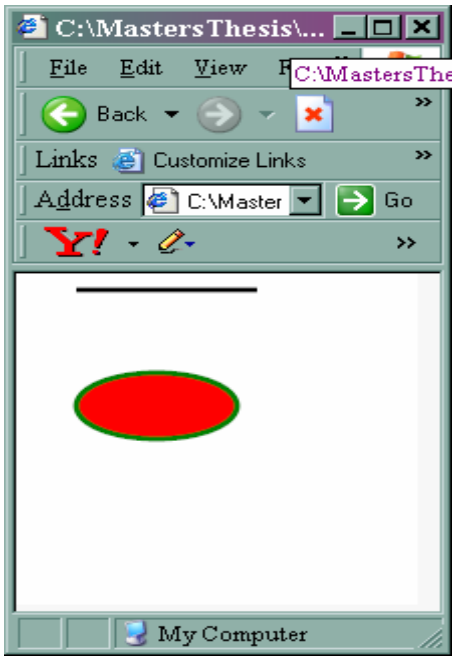

Figure 1: SVG Line and Ellipse from Example 1

Next, consider the “scalable” in scalable vector graphics. Any SVG image, in whole or in part can be manipulated with two transform attributes, “translate” and “scale”. The translate attribute moves the SVG item to a different coordinate, and the scale attribute resizes the item. Since scale resizes the item by altering the coordinate system, using scale or translate and scale in combination requires adjusting the coordinates. To double the size of the ellipse in example 1 above and keep it centered at (30,80) would require SVG code that halves the x and y center coordinates of the ellipse:

```
<svg width=”200px” height=”200px” viewBox = “0 0 200 200”>
   <ellipse cx=”15” cy=”40” rx=”20” ry=”10” style=”stroke:green
   fill:red;”transform=”scale(2.0)”/>
```

Example 2: Scaling an SVG ellipse

The translate attribute could be used to adjust the coordinates of the ellipse as well. As will be seen below, the XSL stylesheet that ScoreSVG uses must adjust the coordinates that place the musical font characters to counteract the scaling factor demanded by the correct size of the font. The scalability factor of SVG makes it possible to create a script that will resize any music score right in the browser using the translate attribute to adjust the coordinates. [1] [25]

Moving beyond simple lines and shapes, SVG is capable of describing elliptical arcs and also very complex curves through quadratic and cubic Bezier curves. These graphics are achieved with the ‘path’ tag. Within the ‘path’ tag, there are number of descriptive letters. ‘M’ means move to, ‘L’ means draw a line, ‘H’ means draw a horizontal line, ‘V’ means draw a vertical line, ‘A’ is for an elliptical arc, ‘Q’ for a quadratic Bezier curve, ‘C’ for a cubic Bezier curve, ‘D’ indicates the starting coordinates. In this list of path descriptors, uppercase defines an absolute coordinate system, and lowercase defines a relative coordinate system.

Quadratic Bezier curves have only one control point, whereas cubic Bezier curves can have two control points and are thus capable of describing more complex curves than the quadratic Bezier curves. [1]

In ScoreSVG, the path tags are used to describe the complex curves of font objects such as treble clefs, eighth note stems and similar things. An example of one of these would not be especially pleasant to look at, but it can be said that the entire Maestro musical font of some two hundred characters can be embedded in an SVG file of only 58 KB. Further discussion of embedding the font in the XSL stylesheet can be found in chapter 5. Staff lines, bar lines, double bar lines, and rectangular rests such as half rests and whole rests are drawn using simple SVG elements, while everything else is rendered using path objects.

To reference the path objects that delineate the font characters, SVG uses XLink, a W3C standard for specifying links in XML documents. [20] A namespace reference is added to the stylesheet document tag to include xlink: xmlns:xlink=http://www.w3.org/1999/xlink. An SVG document will typically have a <defs> tag near the top, designating a section that contains references to objects that will be referred to later in the document. In the ScoreSVG XSL stylesheet, this <defs> section contains all musical font symbols that are used in the score. These are referenced with the <use> tag in this manner throughout the stylesheet: <svg:use xlink:href="#quarterrest" x=”55” y=”43”/>. This makes for a much more compact stylesheet than would result without the <use> tag. Additional details of SVG creation with the styesheet will be discussed in chapter 7.

2.2 Related Research:Music Score Editors

As any student of music knows, the primary music score editors, Finale and Sibelius, while full-featured and capable of delivering print-ready scores, are large, complex and expensive software packages. [21] [22] Output of these programs is to a proprietary file format, or PostScript, or now to MusicXML format. [14] (See next section). Web-viewable scores are rare, but do now exist. Recently, Sibelius has implemented Scorch, which allows web viewing and playing of Sibelius music files and capella, a German company, has implemented html output of music scores. Both of these commercial editors are fairly expensive. Other attempts at making music scores viewable on the web are difficult to find.

A perusal of Acadia University's website on music notation software at *http://ace.acadiau.ca/score/others.htm* reveals a large number of specialized music editors for musical niches such as guitar tablature, early music notation and Braille music. [10] Brahms software at brahms.sourceforge.net even has a piano roll music editor in it! LilyPond is the most notable open source music editor, outputting to .ly files and PostScript. [8] Results from LilyPond are very aesthetically satisfying, as the development team has worked hard to achieve an engraver's quality score. Output to html is achieved by specifying image tags to PostScript images, a process automated by the command-line tool, lilypond-book. Unfortunately, input to LilyPond must be in the lilypond notation format, a text-based format the user must learn. There is no visual front end to LilyPond, though ScoreSVG could perhaps be extended to accomplish this task.

What appears to be missing from the extensive list of editors is an inexpensive or free editor that is general purpose, outputs to a non-proprietary file format, and allows for easy web viewing. ScoreSVG was conceived to fill this apparent gap in music editors, and to

provide a starting place for further musical researches and transformations made possible by XML and SVG.

2.3 Related Research: Music XML Formats and Guido Music Notation

A music XML format was deemed necessary for ScoreSVG to record and preserve the semantic meaning of the music score for possible analysis, transformation, or some additional parsing. Without this XML at its center, ScoreSVG would just be a transformation of graphics from one format to another. As with music editors, there are multifarious music XML formats. No less than seventeen music XML formats can be found at *http://xml.coverpages.org/xmlMusic.html* , and there are many more as well. [9] The most widely used of these formats, indeed the only one to have achieved any widespread use, is the MusicXML format from Recordare. [14] It is very complete, and is now used as a translation agent for Finale and Sibelius, with other third party software hooking up to LilyPond as well. Unfortunately, it is extremely verbose and therefore only machine readable. SMDL from the ISO/IEC group in 1995 was based on SGML, was vastly complex, and seems not to be widely accepted. ABC is a text-based non-XML format, but not altogether complete and missed the XML development explosion. Other music XML projects, such as 4ML and MusiXML, seem to be isolated efforts, as was the project that caught the author's attention: Guido music notation format, discussed below.[12]

The only known attempt to convert music XML to SVG lies in Laura O'Shea's 2003 paper for the SVGOpen2003, MusicML2SVG. She presents an incomplete attempt to convert MusicXML to SVG using XSLT, but the author encounters difficulties with fonts and

positioning of musical elements. These are the same difficulties that will be discussed in chapter 5. [13]

Guido music notation, found at *http://www.informatik.tu-darmstadt.de/AFS/GUIDO/* is a non-XML text-based music notation system that is truly compact and human-readable as well as fairly complete for Western music notation.[5] It was developed by a team of faculty and students at the Technical University of Darmstadt (TUD), and continues to be worked on up to the present. This system is remarkably concise and complete, designed to be written by humans, not machines. A parser and viewer for Guido music notation, the Guido NoteServer was created and runs as a Java applet from the Guido web site. [6] It outputs to .gif and EPS formats. To use it, of course, one must learn Guido.

When the XML explosion occurred (2001), a group of students at TUD developed a GuidoXML format that retained the original system's compactness and completeness and combined this with a guido2xml parser and an xml2guido parser, both written in C++. [4] A subset of GuidoXML based on the DTD developed at TUD was used for ScoreSVG, a subset that will become more complete as more features are implemented in the future. See the appendix for the GuidoXML subset used in ScoreSVG. The GuidoXML subset was amended to include some additional attributes to facilitate processing by the XSL stylesheet, notably x and y coordinates for most XML elements, and the type of clef for the <key> element. Changes made to guidoXML will be discussed further in chapter 6 . Nonetheless, GuidoXML provides a compact vehicle for the semantic meanings of any score created.

Chapter3: ScoreSVG Architecture

3.1 Overall Architecture

The overall architecture of ScoreSVG is a three-tiered architecture: a Java-based WYSIWYG editor on the front end, that allows for easy score creation, an XML file in the middle, capturing the semantics of the score, and an XSL stylesheet to generate an SVG file for the final output. The XML file and the XSL stylesheet are inputs for the saxon 8.4 XSLT processor from Michael Kay and Saxonica. [15] This processor is embedded as a Java .jar file in ScoreSVG. Figure 2 shows the overall architecture of ScoreSVG.

Several tools for producing a SVG output were possible, including a number of imperative programming languages such as Java or Python, but XSLT is a powerful and fast method of transforming XML into many formats provided the number of calculations and decisions are not too great or complex. Initial work was done on the XSL stylesheet and even with advanced XPath 2.0 and XSLT 2.0 features, arriving at proper x coordinates for the various musical elements was deemed exceedingly difficult. Upon deciding to include x- and

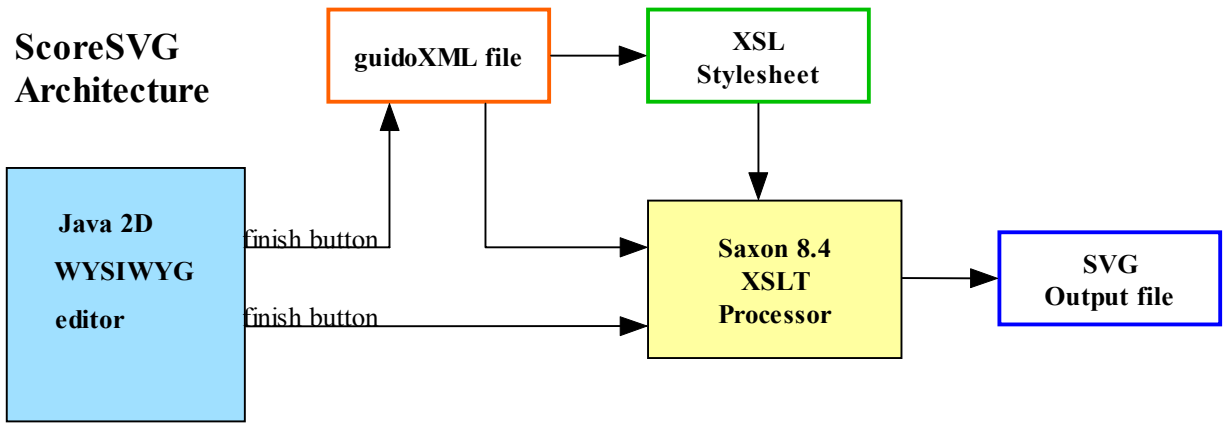

Figure 2: ScoreSVG Overall Architecture

y-coordinates into GuidoXML, then the task of creating SVG with the XSL stylesheet became much more manageable as will be explained further in chapter 7.

When the user activates the finish button in the front end editor, the data collected in Java is converted into XML and written to a file. Then, in a separate command, the XSLT processor is called and takes the GuidoXML file and the XSL stylesheet as inputs and directs output to the SVG output file. At present, this call to the Saxon processor is accomplished through a system call using the Java Runtime class. While this finish process is not extremely fast, it is only done once when the user has finished inputting a music score. Furthermore, parameters can be passed to the stylesheet through Saxon, which could allow for several alternative SVG outputs with, for example, different scripts added. Chapter 10 will discuss future work that will offer the user multiple options for outputting the score.

3.2 Java WYSIWYG Editor Architecture

The design of the front end editor is loosely based on the standard Model-View-Controller (MVC) architecture, with the primary difference being that in ScoreSVG, two different controller subsystems are needed, one for the user's front end view, and the other to generate modified GuidoXML These two controllers are named the Java Controller and the XML Controller in Figure 3, which shows all the subsystems in ScoreSVG. Here the model part of the architecture, where the entity objects reside, is the Data subsystem consisting of a single class, the Storage class. This class is implemented as a Singleton pattern [27], and all of the basic constants for a score are stored here. These parameters include the distance between

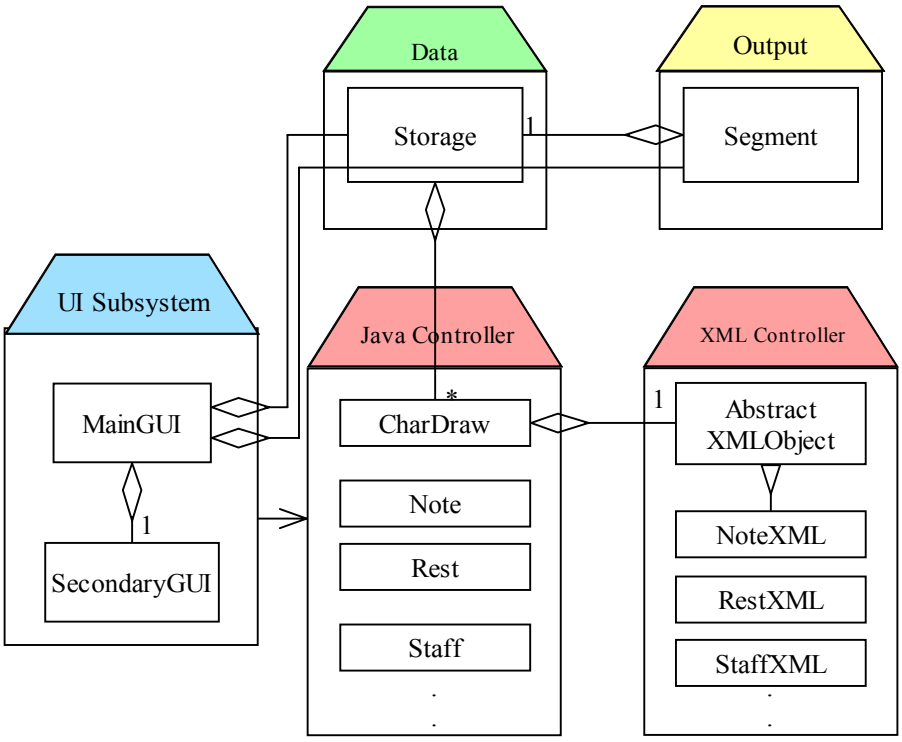

Figure 3: ScoreSVG Front End Subsystems

lines in a staff, the distance between different staffs, the width of the staffs and parameters of this nature which are used by many classes throughout the software. The Storage class also holds a Java LinkedList of CharDraw objects which contain all the data for the user interface classes, and all the data for the XML generation. The CharDraw objects in the linked list contain all the XML data because they in turn have various concrete classes derived from the XMLObject class through composition. The linkage between the Java controller classes and the XML controller classes through class CharDraw became necessary during the development process because the user is able to move notes and other objects in the score editor and when this occurs, the updated positions must be reflected in the XML generated.

The Output subsystem consists of single class, Segment, which is also implemented as a Singleton pattern. This class includes a method that extracts the necessary data from the

linked list of CharDraw objects by calling the toString() methods of the concrete XMLObject classes that are contained in the CharDraw objects. This process is done at the end so that objects that have been moved or deleted by the user will be changed or removed from the linked list of CharDraw objects.The resulting output string is used in a method that writes the modified GuidoXML to a file, completing the process of the XML Controller subsystem. An instance of Segment is also found in the MainGUI class so that its methods can be called when the finish button is activated by the user.

The Java Controller subsystem is where the bulk of the calculations occur in ScoreSVG, because it is here that all the user input data from the UI subsystem, such as what object to draw at what coordinates, arrives. Figure 4 shows the full Java Controller subsystem, revealing that all the classes in it other than CharDraw, contain an instance of the Storage class, perhaps several CharDraw objects, and a concrete class derived from the XMLObject class. Essentially, there is one class for most types of musical elements in modified GuidoXML. All classes in this subsystem except for CharDraw have a getCharDraw() method that processes the user data from the view classes. The getCharDraw() method takes a HashMap object as an input parameter, because this is the type of object that keeps track of the state information in the MainGUI class. Coordinates and other user data are obtained form the HashMap, and the resulting CharDraw objects are added to the linked list in the Storage class. At the same time, data necessary to generate the XML is added to the concrete subclass of XMLObject that corresponds to the particular class in the Java controller subsystem For the Note class this would be NoteXML, for the Rest class, it would be RestXML, and so on.

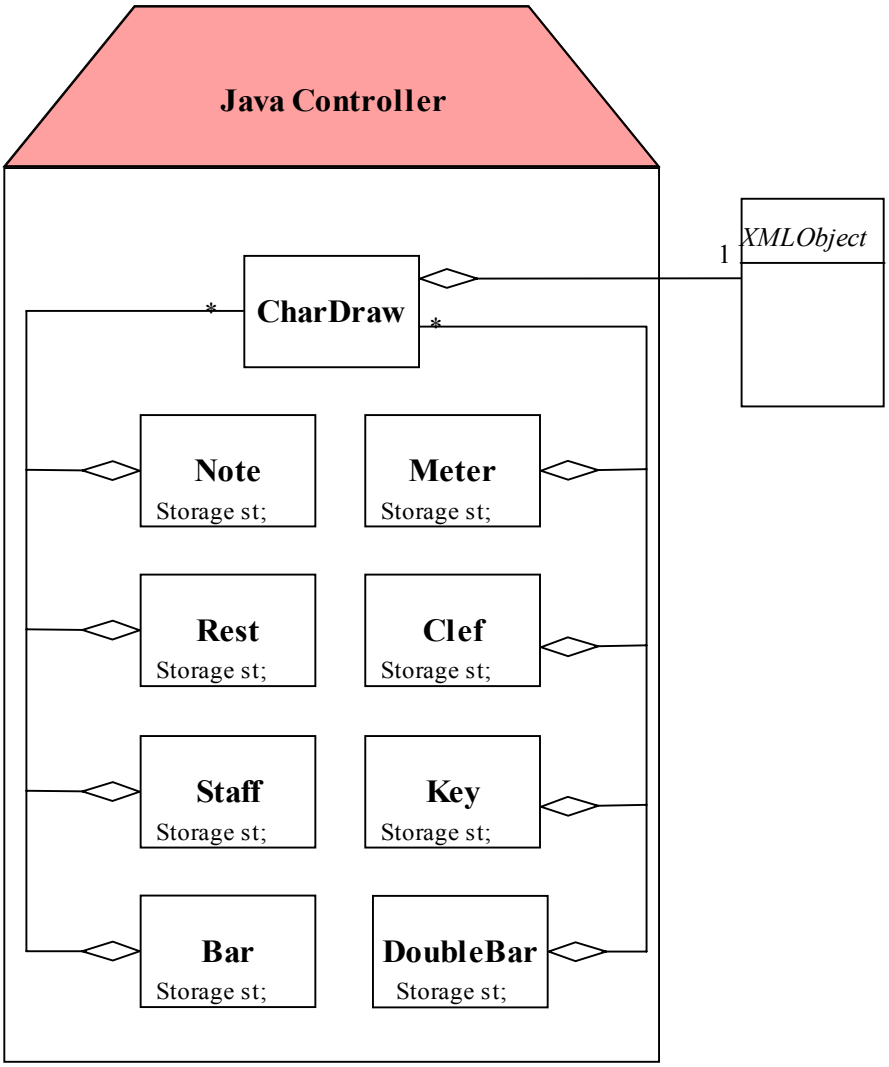

Figure 4: Java Controller Subsystem

More details on the XML process can be found in chapter 6. Instances of the Storage class are necessary in all classes to enable access the size parameters of the score and the linked list of CharDraw objects.

Chapter 4: The ScoreSVG User Interface

4.1 The Main User Interface and the MainGUI class

The initial concept for the ScoreSVG GUI design was to give the user access to basic, frequently used features, without the necessity of dealing with drop-down menus or any other time-consuming search. Since ScoreSVG at present only implements basic score editing features, this was not difficult to achieve. A glance at Figure 5, showing the main GUI for ScoreSVG, reveals two rows of buttons at the top that provide access to most of the musical items to be added to a score. Here are buttons for notes and rests of varying durations with accompanying possibilities of dots and accidentals (sharps, flats, and natural signs). The dot and accidental buttons toggle and have a different background color when activated. The last note entered in Figure 5 has a dot, and hence the dot button in the note row is toggled ‘on’ and orange in color. Buttons to control the stem direction of notes, barlines and double barlines are also found in these two rows. By far the majority of items the user will add to a score can all be found in these two rows of buttons without any need on the part of the user to scroll through drop-down menus.

On the left side of the main GUI are buttons involving the setup and creation of the score as well as items that require a secondary GUI for user input. The seven buttons on the left bar are the Setup button, the Staff button, the Meter button, the Key button, the Delete button, the Slur button and the FINISH button. The Delete button and the Slur button have not yet been implemented in ScoreSVG , but are on the list of features to add to ScoreSVG. More detail can be found in chapter 9, where future plans are discussed. The setup button allows the

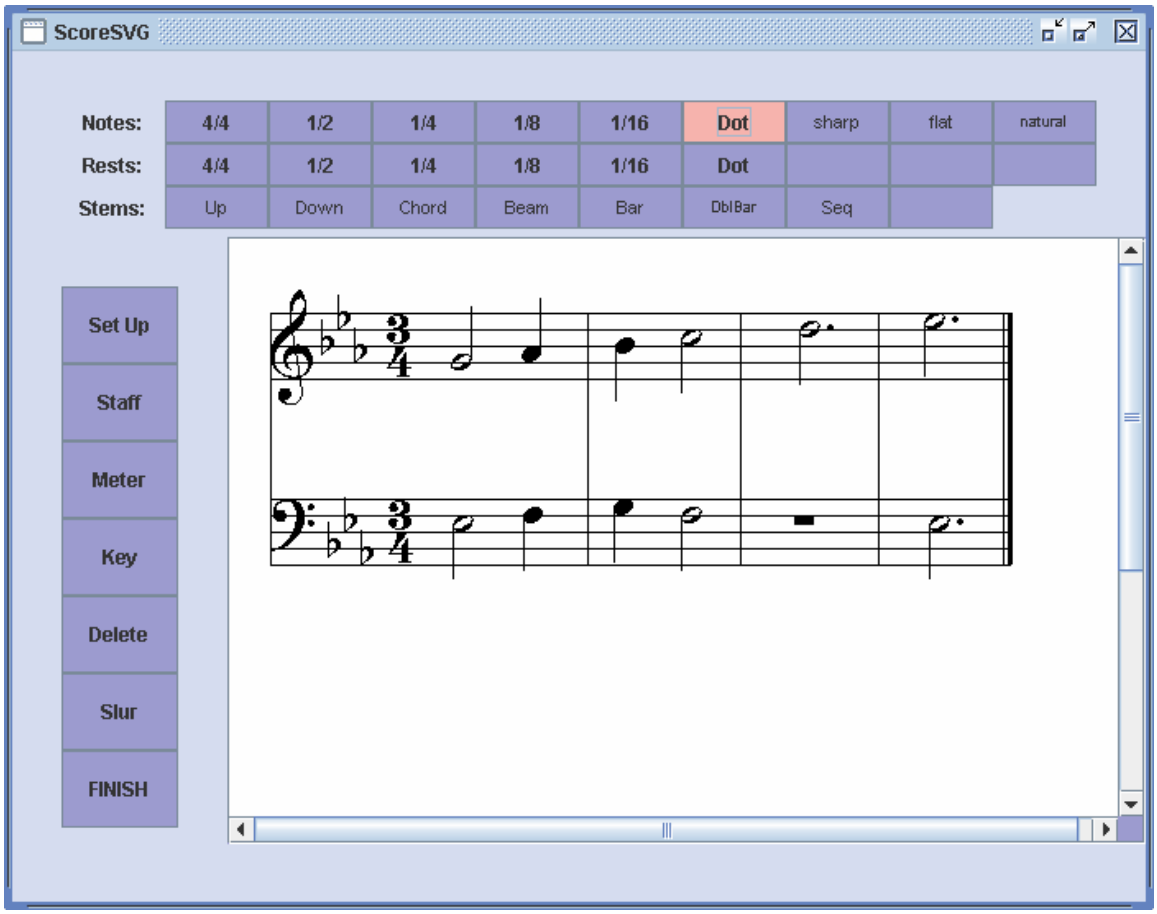

Figure 5: ScoreSVG Main GUI

user to control the size of the score by choosing from a number of possibilities discussed in the next section. The staff button allows the user to choose the clef that the staff will contain, at this point either a treble clef or a bass clef. Similarly, the meter and key buttons summon a secondary GUI allowing the user to choose a time signature and a key for the musical score. The Delete button has not yet been implemented yet; it is intended to let the user delete an entire section of the score at once. At present, the user may delete an item by double-clicking on it. If the item is a note with a sharp and a dot, for instance, three double clicks are necessary to remove the entire item from the score. The Slur button, when implemented, will

allow the user to add slurs and ties to the score in the necessary places. The Finish button calls all the procedures necessary to output the score in an SVG file: calling methods in the Segment class to iterate through the list of CharDraw items and extract the XML strings, write the XML to a file, and call on Saxon 8.4 with the XML file and the XSL stylesheet as parameters to yield the final SVG file.

Later, another bar of buttons added to the bottom of the main GUI might enable the user to select various different scripts to be added to the outcome, or different XML formats to produce. The design of ScoreSVG was conceived to allow easy extensibility for XML formats. By adding another abstract class on top of XMLObjects, a several sets of derived XML classes could be invoked by the system using a classic builder pattern. [27]

4.2 The SecondaryGUI Class

The SecondaryGUI class contains al the code for the secondary, or subsidiary GUIs invoked by the user when activating buttons on the left side of the man GUI. There are secondary GUIs for the setup, staff, key and meter buttons on the left side of the main panel. Figure 6 shows the Staff Setup GUI, Figure 7 the Staff GUI, figure 8 the Meter GUI and figure 9 the Key GUI. Their functions with regard to user input follow logically from musical necessities. The Staff Setup GUI is especially important because it allows the user to set in in advance the vertical and horizontal size of the staffs as well as the number of staffs per system. This latter feature enables the user to draw barlines and double barlines one, two, three, four or more staffs high in scores with multiple staffs per system such as piano music, which has two staffs per system, a treble and a bass staff.

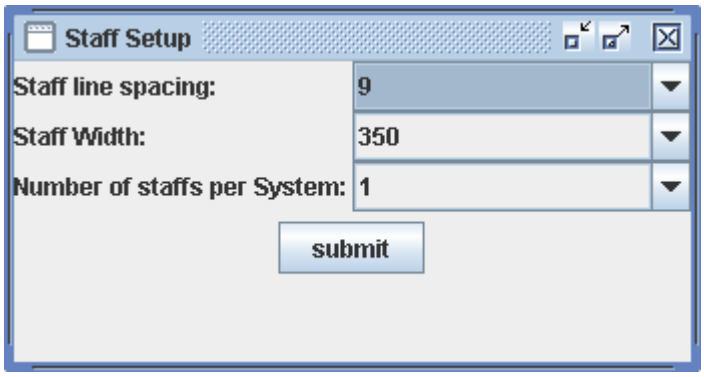

Figure 6: Staff Setup GUI

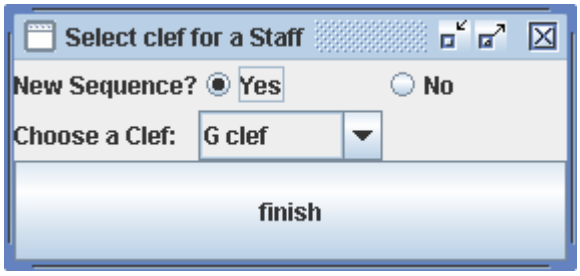

Figure 7: Staff GUI

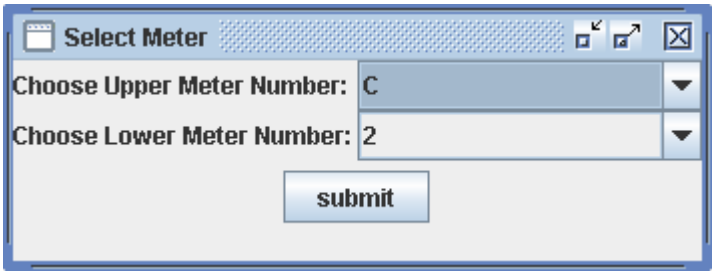

Figure 8: Meter GUI

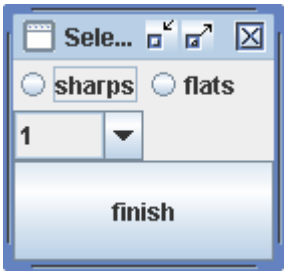

Figure 9: Key GUI

4.3 Implementation Details of the MainGUI Class

While it would be impossible to detail all of the procedures necessary to develop the Java code for the MainGUI class, some salient features will be outlined here. When the user wishes to use the top two bars of buttons for notes and rests, the input of several buttons must be stored to achieve the end result desired. If, for instance, the user wishes to put a dotted quarter note, stem up, with a sharp in front of it, then the note button marked '1/4', the dot button on the same row, the stemsup button and the sharp button must all be activated somehow. A Java HashMap variable called stateHash is used to keep track of the state of these buttons. Example 3 shows the lines of code initializing the stateHash variable. If the user puts a certain kind of note in the score and then switches to a rest by clicking on a rest button, all of the remainder of the previous note information is retained in stateHash. If the user has put the aforementioned dotted quarter note, stem up and sharp on the staff, then paced a quarter rest on the staff, when the user again clicks on a note button, the sharp, dot and stem up features will still be there. When the user clicks the mouse on the score panel, the mouse event coordinates are registered by the MouseListener class and placed into the stateHash variable. Then, one or another of the Java Controller subsystem classes is instantiated with the stateHash variable being passed to the constructor of the class. Thus the stateHash variable is the primary means of passing information from the user interface class and the Java controller classes.

Another interesting feature of ScoreSVG lies in the users ability to move notes and rests to new locations and have the new location reflected in the XML. The dragging of items is handled by the MouseMotionHandler class which implements the MouseMotionListener

interface. If it is determined that the mouse is over an previously drawn item, the the x- and y-coordinates of the item are reset in the CharDraw class, and the x- and y-coordinates of the

```
stateHash = new HashMap(12);
stateHash.put("name","ZZ");
stateHash.put("noteduration","ZZ");
stateHash.put("restduration","ZZ");
stateHash.put("notedot","ZZ");
stateHash.put("restdot","ZZ");
stateHash.put("stems","stemsup");
stateHash.put("accidental","ZZ");
stateHash.put("xcoord",0);
stateHash.put("ycoord",0);
stateHash.put("uppernum","ZZ");
stateHash.put("lowernum","ZZ");
stateHash.put("keynumber","ZZ");
stateHash.put("seqnumber",0);
```

Example 3 : Initialization of stateHash Variable

CharDraws object's XMLObject are reset as well. Additionally, the setRectangle() method of CharDraw is called, which resets the area around the item that determines its position for movement or deletion.In this manner, the resulting XML reflects any movements the user made by dragging items on the score panel.

The deletion of score items is accomplished via the mouseClicked() method in the MouseHandler class which implements the MouseAdapter interface. If the mouse is over a previously drawn item and the mouse has been clicked twice, the item is removed from the CharDraw linked list in the Storage class. Later, ScoreSVG may be updated to delete several items at once that have the same or overlapping rectangle areas. This would mean that a single double click would delete a note, its dot, sharp and ledger lines all at once.

Chapter 5: Embedding a Musical Font in ScoreSVG

5.1 Embedding the Maestro Font in the Java Front End

A number of musical fonts have been created over the past twenty years for use in the printing of scores, for typesetting, and for use in music score editors. The venerable Sonata font from Adobe was one of the first, and is still available today in several formats. Trying to draw the more difficult, curvy musical symbols and have them look good as a task for a seasoned graphic artist, not a student of computer science. This fact combined with some digging in the Java Font class API resulted in excitement over the possibility of embedding the Maestro font, freely available in a TrueType format, in the ScoreSVG Java front end.

Example 4 shows the few lines of Java code needed to embed a font into a Java application in such a way that the font need not be installed on the client system. This code is for Java 1.4; in the new Java 1.5, the font can be directly created from the font file without using any stream objects at all. [28]

```
FileInputStream fin = new      FileInputStream("C:\\ScoreSVG\\Fonts\\MAESTRO_.TTF");
    DataInputStream din = new DataInputStream(fin);
    musicFont2 = Font.createFont(Font.PLAIN,din);
      musicFont = musicFont2.deriveFont((float)size);
```

Example 4: Embedding a Font in Java

The createFont() method allows a font to be embedded in a Java application as a TrueType font file. It creates a font of size 1 which is then resized with the deriveFont() method. Theoretically this should have been the solution for ScoreSVG, especially since it was known that the Maestro font was a Unicode mapped font, the type of font that Java is

designed to display. The solution above, however was not sufficient; no musical characters would appear in the main GUI no matter which Unicode numbers were tried.

Further research revealed that the maestro font, like the WingDings and WebDings fonts on most Windows systems was a symbol encoded font which would not appear in a Java application. Persistence led to the discovery that all symbol encoded fonts use the range of hexadecimal numbers from F020 to F0FB. The only remaining question was: what musical symbols were mapped to which of these hexadecimal numbers? Fortunately, FontCreator 5.0 from WebLogic provided the answers, as one of its features allows one to read which characters are mapped to which hexadecimal numbers. [7]

Once the mapping of characters to Unicode numbers was known then any musical symbol could be displayed using the Java Unicode escape sequence in this manner: char c = '\uF026' (displays a treble clef in Maestro) followed by this line in the paint() method of class ScorePanel (extends JPanel): g2.drawstring("" + c, xcoord,ycoord), where c is the Unicode character, and xcoord and ycoord are the x- and y-coordinates of the location where the character should be displayed. With this solution in hand, all of the Maestro characters appeared in the Java front end.

The only remaining question was how large the font would need to be to fit correctly into a musical staff of a certain size. Experimentation revealed that a font multiplier factor of 3.867 worked well. Multiply the number of pixels between the staff lines by this factor and the necessary font size would be correctly calculated.

5.2 Embedding the Maestro Font in the XSL Stylesheet

On the backend of ScoreSVG the challenge was to find a way to transform the Maestro TrueType font into SVG. Each character would need to be represented as a path object in SVG as explained earlier in chapter 2. The Batik project from Apache contains not only an SVG viewer, but also a command line utility called ttf2svg, which is designed to transform a TrueType font into SVG. Unfortunately, ttf2svg does not work for symbol encoded fonts such as maestro. Adobe Illustrator and PhotoImpact from Ulead both have the capability of exporting fonts into SVG. [11] The Maestro characters needed for ScoreSVG were displayed in 24 point size in PhotoImpact 7, and exported as SVG. The result was a series of path objects that could be used in the <defs> section of an SVG file. The entire Maestro font of over two hundred characters could be exported as SVG in a file of 58 KB, and the characters used in ScoreSVG take up only around 25 KB.

As with the front end embedding, the remaining question was the scaling factor for the font. Experimentation proved that the exported 24 point font needed a scaling factor equal to the number of pixels between the staff lines divided by 6.67.

Chapter 6: ScoreSVG GuidoXML Generation

6.1 Modifying GuidoXML

As mentioned earlier, certain limitations in the capacity of XSLT to effect calculations and decisions led to the necessity of modifying GuidoXML to make the backend of ScoreSVG more readily implemented. To understand the issues here, consider Example 5 below, which is a sample of unmodified GuidoXML. The most troubling elements in GuidoXML for an XSL stylesheet are the <stemsup/> and <stemsdown/> tags. These are

empty tags that function solely by their position in the XMl document. For the functional and declarative programming paradigm of XSLT, which does not allow for side effects such as updated variables, these tags are a great difficulty. Initially, work was done to display the stems correctly without any other data, and this was ultimately achieved with use of complex

```
<guido>
  <segment>
    <sequence>
      <clef s = "f"/>
      <staff i = "1"/>
      <key i = "-3"/>
      <meter s = "4/4"/>
      <stemsdown/>
      <note name = "f" octave = "0" duration = "1/8" accidentals = "flat"/>
      <note name = "g" octave = "0" duration = "1/4" />
      <note name = "d" octave = "1" duration = "1/8" accidentals = "sharp"/>
      <stemsup/>
      <note name = "e" octave = "-1" duration = "2/4"/>
      <bar/>
      <rest duration = "1/8"/>
      <note name = "b" octave = "0" duration = "3/8" accidentals = "natural"/>
      <rest duration = "1/4"/>
      <note name = "c" octave = "-1" duration = "1/4" accidentals = "natural"/>
      <bar/>
      <rest duration = "4/4"/>
      <doublebar/>
    </sequence>
  </segment>
</guido>
```

Example 5: Original GuidoXML

XSLT 2.0 and XPath 2.0 features. The correct y coordinates of all notes could also be calculated, but getting the correct x coordinates would require a difficult and complex algorithm really not suitable for XSLT. Therefore that decision was made to use a little ‘creative laziness’ and modify GuidoXML to enable the backend processing to be done without undue strain.

Before displaying the modified GuidoXML, it should be mentioned that after the top level element, <guido>, there is a single element per score, and that the <sequence> tags refer to lines of music that are supposed to be played simultaneously. Thus to have the GuidoXML emerge correctly, one should enter a score a use the 'Seq' button on the second row at the top of the main GUI to declare a new sequence. This causes a new <sequence> tag to be written into the GuidoXML.

Example 6 shows the modified GuidoXML. The lineSpace attribute in the segment tag is the most fundamental parameter for sizing a score, and as an attribute in the segment made it easy to reference in the XSL stylesheet. X- and y-coordinates were added to all elements, and the applicable clef was designated in the 's' attribute of the key element. In addition, some features needed a second y-coordinate, such as the barlines and double barlines, and the staffs needed a second x-coordinate to delineate the width of a staff. These were all added to the GuidoXML, and currently, the stemsup and stemsdown tags have been eliminated, and a stem attribute added to all notes to indicate whether the stem is up or down for each individual note. This greatly simplifies the XSL stylesheet, though there are still a number of calculations happening there as will be seen in the next chapter.

6.2 Generating Modified GuidoXML in the XML Controller Classes

The basic mechanism for generating the modified GuidoXML lies in the XML Controller subsystem classes which all inherit from the abstract XMLObject class and override its methods. As mentioned earlier, each CharDraw object that contains the data necessary for drawing the musical font characters on the front end also contains an XMLObject object of the relevant type. In the Note class, for instance, the main CharDraw

object will contain an instance of the concrete NoteXML class, and this class' methods will be invoked in code in the Note class. Similarly, in the Rest class there will be an instance of the RestXML class, and its methods will be invoked.

By far the most effort at generating the modified GuidoXML occurs in the Note class using methods of the NoteXML class. As can be seen from Examples 5 and 6, in either

```
<guido>
  <segment lineSpace=”20”>
     <sequence>
       <clef s = "f" x=”100” y=”100” />
       <staff i = "1" x=”100” y=”100” x2=”850”/>
       <key s=”f” i = "-3" x=”100” y=”100”/>
       <meter s = "4/4" x=”235” y=”100”/>
       <note name = "f" octave = "0" duration = "1/8" accidentals = "flat" stem=”down” x=”312 y=”120”/>
       <note name = "g" octave = "0" duration = "1/4" stem=”down” x=”364” y=”110”  />
      <note name="d" octave ="1" duration = "1/8" accidentals = "sharp" stem=”down” x=”421” y=”70”/>
       <note name = "e" octave = "-1" duration = "1/2" stem=”up” x=”470” y=”200”/>
       <bar x=”502” y=”100” y2=”180”/>
       <rest duration = "1/8" x=”514” y=”125”/>
       <note name="b" octave="0" duration="3/8" accidentals = "natural" stem=”up” x=”577” y=”160”/>
      <rest duration = "1/4" x=”623” y=”126”/>
     <note name="c" octave = "-1" duration = "1/4" accidentals ="natural" stem=”up” x=”660” y=”220”/>
       <bar x=”692” y=”100” y2=”180”/>
       <rest duration = "4/4" x=”744” y=”120”/>
        <doublebar x=”850” y=”100” y2=”180”/>
    </sequence>
  </segment>
</guido>
```

Example 6: Modified GuidoXML

version of the XML, the note elements contain the note names and the octave of the note. These pieces of data are essential for GuidoXML and the semantic meaning of the score. They are not, however, essential for the display of the notes in SVG. The note name and the octave must be calculated from the y position of the note. This is accomplished by keeping track of which clef is drawn on each staff, and referring to one of two Java HashMaps in the Storage class that define the note name and octave for the treble and bass clefs. The methods

in NoteXML that accomplish this determine the number of the staff the note is on, and then determine the clef that the staff contains, and finally look up the value of the note name and octave in the relevant HashMap. ScoreSVG uses an octave designation of '1' for the octave from middle C to the B above that. Higher octaves are numbered 2,3, and so on, and lower octaves descend to 0, -1, -2, and so on. If the note contains a dot, then the duration must be refigured as it will be different from the string value sent by the note button in the main GUI. If the value of the note button is '1/4', and the dot is present, then the duration of the note becomes '3/8' in GuidoXML. The durations for the rests with dots are handled in the same manner.

New sequences are created using the 'Seq' button on the second row of buttons in the main GUI. A sequence number is kept in the stateHash variable that is passed from the main GUI to the Java Controller classes. Clicking the 'Seq' button will increase this number, and update a variable in the various concrete XMLObject classes. Finally, in the Segment class which forms the output subsystem, if the sequence number is incremented, then <sequence> tags are added to the modified GuidoXML output string.

Other XML elements are easily handled. Producing the necessary attribute for the meter elements involves merely concatenating the upper meter number, a forward slash and the lower meter number. The key attribute is exactly the same number the secondary GUI for the key has returned. X- and y-coordinates as well as y2- or x2-coordinates are extracted form the values in the stateHash variable and inserted into the XMLObject classes in all cases. All derived XMLObject classes have a setXY() method and a toString() method. The setXY() method allows for position changes when notes or rests are dragged in the main GUI, and the toString() method allows the XML to be output to a file.

The Segment class is where the CharDraw list is treated through, the XMLObject classes referenced, and the toString() method of each such object called and the result added to the XML output string. Lastly, the XML output string is written to a file.

Chapter 7: ScoreSVG XSL Stylesheet

7.1 Overview of the Stylesheet

The overall design of the ScoreSVG XSl stylesheet is what author Micahel Kay calls a 'rule-based' stylesheet. [2] In such a stylesheet, an unpredictable input source can be handled by specifying a template match for each type of element encountered. The ScoreSVG is essentially this type of stylesheet, but some calculations remain to be done, specifically the drawing of ledger lines for the notes. The ledgerDraw template and the accidentalDraw template are called from the note template, with x- and y-coordinates passed as parameters, and represent a departure from a straightforward rule-based stylesheet.

The header of the ScoreSVG XSL stylesheet , shown in Example 7, indicates that the stylesheet conforms to version 2.0 of the W3C XSLY specification. Since the Saxon 8.4 XSLT processor incorporates version 2.0, this heading avoids warnings from the Saxon compiler. Namespaces for XSL,SVG and XLink are declared. Xlink is necessary for the XSL <use> tag to link to an item in the <defs> section of the stylesheet. The xs namespace from the XMLSchema specification is necessary to assign explicit types to certain variables, or to cast variables to different type. Finally, the output type of the stylesheet is given as XML, since SVG is a type of XML.

The ScoreSVG XSL stylesheet continues with the declarations of global variables used by many templates. In XSLT, variables are not really variables, they are constants and

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="2.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:svg="http://www.w3.org/2000/svg"
   xmlns:xlink="http://www.w3.org/1999/xlink"
   xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xsl:output  method="xml" version="1.0" encoding="ISO-8859-1"/>
```

Example 7: XSL Stylesheet Header

not subject to change at all. Here, in Example 8, are the essential constants necessary for rendering the score: the space between staff lines, the space between staffs, the 'fontFactor' for determining the correct font size, and the starting y-coordinate of the first staff. From these constants, all further calculations are possible. The space between staff lines is extracted from the segment tag at the top of the GuidoXML document.

```
<xsl:variable name="staffLineSpace" select="/guido/segment/@lineSpace"></xsl:variable>
<xsl:variable name="fontFactor" select="$staffLineSpace div 6.67"></xsl:variable>
<xsl:variable name="spaceBetweenStaffs" select="$staffLineSpace*11 +
ceiling($staffLineSpace*0.25)"/>
<xsl:variable name="startY" select="./guido/segment/sequence[1]/staff[1]/@y"/>
```

Example 8: Global Variables in the XSL Stylesheet

After the global variables are declared the top level template match follows setting the start of the SVG output document. The match is for 'guido/segment'. There is only a single segment in any Guido document, and the segment element is the child of the topmost guido element. The start of the SVG document is declared, and the <defs> section with all the path information of the musical font characters follows. An XSL apply-templates element calls on all the templates to start the transformation.

A sizeable portion of the stylesheet is devoted to checking for the value of some attribute in an XML tag, and based upon its value, referencing the correct item in the <defs> section with the SVG <use> tag and placing it at the x- and y-coordinates specified. As an example of this operation, consider the placement of a ‘g’ or treble clef in Example 9:

```
<xsl:template match="sequence/clef">

 <xsl:param name="xPos" select="./@x"></xsl:param>
 <xsl:param name="yPos" select="./@y"></xsl:param>

 <xsl:if test="./@s eq 'g'">
<svg:use xlink:href="#trebleclef" x="{$xPos div $fontFactor}" y="{$yPos div $fontFactor}"
    transform="scale({$fontFactor})"/>
 </xsl:if>
```

Example 9: Placing a Treble Clef in the XSL Stylesheet

The x- and y-coordinates are extracted from the clef element attributes, the s attribute of the clef element tested with an XSL ‘if’ to see which clef is indicated. The contents of the SVG <use> tag are the most important to consider. XLink makes the reference back to the treble clef character defined in the <defs> section. The clef is properly placed by dividing the x- and y-coordinates by the fontFactor, and then scaling the clef using the transform attribute with the fontFactor as the scaling factor. This works because when an item is scaled in SVG, the whole coordinate system is scaled to achieve the scaling.

The ledgerDraw template in the XSL stylesheet does considerable calculations to draw the correct number of ledger lines. First, the distance of the y coordinate of the note from the top and bottom of the staff is calculated, and this distance is divided by one half of the space between the staff lines to yield the number of ledger lines to draw. Then an XSL for-each loop

is used to draw the lines, with the XPath function position() used to determine which iteration of the loop is happening, and at which y coordinate to draw the lines.[24]

Whole rests, half rests, bar lines, double bar lines and staff lines are drawn 'from scratch' in the stylesheet, that is, they are drawn using SVG basic lines and rectangles and not the result of placing music font characters. This method is easier and more flexible than using the font characters, and completely necessary for the staff lines which really have no viable font alternative.

Nothing of great import occurs in the other templates in the stylesheet, namely, the meter, key, accidentalDraw, or rest templates, that has not already been mentioned. The next chapter will describe the process of using ScoreSVG from a user's perspective.

Chapter 8: ScoreSVG User Guide

8.1 Set Up and Staffs

To begin a score in ScoreSVG, it is best to click the Set Up button on the left side of the main panel. The Set up GUI will appear and offer the user choices about the size of the staffs for the score. The user can choose a space between the staff lines of 9,10,12,15, or 18 pixels, with the default set to 12 pixels. Similarly, the width of the staffs can be chosen from 350,450,550,650 or 750 pixels in width. The default is set to 750 pixels here. If the user does not use the Set Up button, scores will be 750 pixels wide with a space of 12 pixels between staff lines. Then the user should select how many staffs per system the score will have. This would be two for a piano score, three for a score for voice and piano, or four for a string quartet (although no C clef has yet been implemented in ScoreSVG). Clicking on the submit button causes the GUI to vanish, and the set up data to be recorded in the software.

After using the Set Up feature, the user can proceed to the Staff button. This will invoke a small GUI where the user can choose either a G (or treble) clef or an F (or bass) clef. The new sequence question is unnecessary, as ScoreSVG keeps track of sequences with the 'Seq' button. Clicking finish will cause the staff with the clef to appear in the main panel. If desired, the user may draw another or several more staffs on the panel in the same manner.

The 'Seq' button on the third row of buttons at the top of the main panel can be used to indicate a new sequence. This is not required for the visual output in SVG, but for proper GuidoXML notation. It is up to the user to decide if this is a necessary task. If not used, the resulting XML file will contain just one sequence element.

8.2 Key Signatures and Time Signatures

Clicking on the Key button on the left side of the panel reveals the key signature GUI. Here the user chooses sharps or flats and how many. After clicking finish, nothing will happen, but when the user clicks the mouse on the panel, the sharps or flats will be drawn in the correct position for the clef the staff contains, with the mouse click defining the left most side of the accidentals drawn. If more than one staff is drawn, the user may keep clicking on staffs, and the sharps or flats will appear. Changes of key signature can be placed at any point on any staff.

The Meter button on the left side of the main panel lets the user choose a time signature for the piece. The meter GUI asks the user for the upper number and the lower number. If 'C' is chosen, no lower number need be selected. After clicking the submit button, the meter GUI will disappear and the user may click on any staff and the correct meter will

appear. Several time signatures on several staffs may be drawn successively, and changes of time signature can be made throughout the score.

8.3 Notes, Rests and Barlines

Notes, rests and barlines require the use of the three upper rows of buttons in the main GUI. To get the desired type of note, the user first clicks on the duration of the note desired, then the dot button if a dotted note is wanted, an accidental if called for and then the Up or Down button by the Stems row to indicate an up or down stem on the note. The order in which the buttons other than the duration button (it should be clicked first) are clicked does not matter. The dot and accidental buttons toggle and are orange when on. Similarly, to get the type of rest desired, the user clicks on the duration first and then adds a dot if desired. After choosing the type of note or rest, by clicking on the main panel, the note or rest will appear at the point of the mouse click, with the exception of half rests and whole rests which appear at the standard vertical placement on the staff. Ledger lines will be drawn automatically above and below the staff as needed. Three ledger lines plus a space are possible currently in ScoreSVG. Notes with four ledger lines are not yet possible.

Notes and rests can be moved by clicking the mouse on them and while holding the mouse button depressed, dragging them across the panel. Dots, accidentals, and ledger lines have to be dragged separately, but only the note head has any effect on the final result in the XML file. Clefs, time signature numbers, and key signature accidentals may also be moved.

Notes, rests, dots and accidentals may be deleted by double clicking them. If a note has a dot and an accidental, double clicking may first cause the dot or accidental to vanish, and then two more double clicks will be required to delete the remaining two items. NOTE:

only deleting the note itself removes it from the XML. Deleting a dot or an accidental by itself will not change the outcome.

Barlines and double barlines can be drawn by using the Bar and DblBar buttons on the third row of buttons at the top of the main panel. Once the button is clicked, the user may click on any staff, and a barline or double barline long enough to cover the number of staffs per system will be drawn. If fewer than that number of staffs have been drawn, they can be added using the Staff button. Unfortunately, barlines and double barlines cannot yet be moved or deleted in ScoreSVG.

At present, the 16th note and rest features are not implemented in ScoreSVG, and the Chord and beam buttons are not implemented either. The delete button is also awaiting future development as is the slur button on the left side of the main panel. Further discussion of development plans can be found in chapter 10.

8.4 Finishing and Output

At the present stage of development, outputting a score in Score SVG is extremely simple. Once all elements of the score have been input on the main panel, the user simply clicks on the FINISH button at the bottom of the button bar on the left of the main panel. One click does it all: the modified GuidoXML file is written in the same directory as the source files, the Saxon 8.4 XSLT processor is invoked, and the stylesheet transformation produces the SVG file which is named score1.svg. Currently, the user will have to double click the score1.svg file and view it in the system default browser, which will need the Adobe SVG plugin. Later, it may be possible to preview the SVG file right in the Java Swing panel.

8.5 ScoreSVG Delivery

The system call made by ScoreSVG to the Saxon XSLT processor using the Java Runtime class makes the packaging of the software to run on different platforms somewhat challenging. The system call, made using the exec() method does not work with relative path names. If a system has a main C: \ drive, one can just put the Maestro font in a folder inside the main folder, the Saxon .jar file in the same folder and run ScoreSVG from the command line. This approach would work on a Linux platform as well.

In an effort to make ScoreSVG run more easily on a Windows platform, that is, without resorting to the command line, a piece of software called JavaLauncher from SyncEdit software was employed to create a Windows executable for ScoreSVG. [16] JavaLauncher even allows the Saxon .jar file and the Maestro folder to be referenced in the Windows .exe file created. Using JavaLauncher to create a Windows executable and Icon-Lover software from Aha-Soft [17] to make a custom icon (.ico file) made ScoreSVG much easier and more attractive to run on a Windows system. Just a click on the .exe file and ScoreSVG will start up immediately. Most Windows users are not comfortable with a command line environment hence the .exe file helps the user-friendly mission of ScoreSVG greatly. Linux users are likely to be quite comfortable using the command line, so that running ScoreSVG from the command on Linux should not be difficult at all. The only other requirement, of course, is to download and install the Adobe SVG plugin. Adobe has made this an extremely painless and quick process that should not deter any computer literate individual. [18]

Chapter 9: Conclusion and Future Work

9.1 Conclusion

This paper has examined ScoreSVG from the Java WSIWYG front end which allows the user to easily enter musical elements into a score, to the modified GuidoXML in the middle, and finally to the XSL stylesheet which produces the SVG file at the end. The three-tiered architecture of the software does indeed offer a way to record the musical semantics of a score in the GuidoXML, and a graphical representation of the score in the SVG output. ScoreSVG does not use proprietary file formats, and its output is easily viewable in browsers that everyone has on their system. Furthermore, the SVG format will allow transformations and possibilities through scripting that cannot be achieved in any other format. Finally, SVG graphics, being vector-based, are small in size and remain very clear even when enlarged to billboard size.

Referring to ScoreSVG as a software framework is justified because it can be extended to include other XML formats and to offer a variety of preformatted scripts for various purposes, and with additional XSL stylesheets, produce output in different formats, even audio files. Analytical methods to apply to the resulting XML could be added to the framework as well. Lastly, ScoreSVG can be configured to run over the web using Java WebStart technology.

9.2 Short-Term Feature Improvements to ScoreSVG

A considerable laundry list of features comes to mind when contemplating the features that ScoreSVG needs to be really useful. But this is the short-term list, in rough order of priority at the current date:

1. Implement chords
2. Implement beamed notes: even groups of two, three, or four eighth notes and 16th notes would make a large difference
3. Implement slurs and ties
4. Make barlines and double barlines moveable and deletable.
5. Make notes with many items delete with a single double click.
6. Enable the user to put text on the score for expressive markings, titles, etc.
7. Add a C clef
8. Add 16th notes
9. Add a collection of symbols: fermatas, accents, staccatos, tenutos, crescendos and diminuendos.
10. Implement the Delete button to enable the user to delete whole sections of items.
11. Set up ScoreSVG with the new Java WebStart technology so that it can run on the Web.
12. Enable the user to select the name and location of the output file using a JFileChooser Swing GUI.
13. Enable the user to print the contents of the main panel directly from the main panel.
14. Import the necessary packages from Saxon so that the XSLT processor can be used without resorting to a system call.
15. From the Apache Batik project, use the JSVGCanvas API to make previews of the SVG output visible in a Java Swing GUI.

9.3 Extending the ScoreSVG Framework

Score SVG has numerous possibilities for future development. It could be easily extended to write different kinds of music XML from the front end. Music XML comes to mind immediately, because this would allow ScoreSVG scores to be output in other editors. If instead, ScoreSVG wrote the front end musical data in LilyPond script, then it would become the user-friendly front end for LilyPond.

An entire bottom bar of buttons could allow the user to select from various preset scripts to be added to the final SVG file. One such script would allow the user to select the size of the score from buttons viewable in the browser. In Figure 10, the output of Example 6 is shown in a browser with JavaScript added that lets the user choose the size of the score: S for small, M for medium, and L for large. Another possible script would allow the user to highlight or change the color of chosen elements. Accidentals could be made red, ends of phrases could be colored, accented appoggiaturas could be highlighted, particular intervals or chords could be highlighted. This could be one of the selections on the bottom button bar. Parameters could be passes to the top level of the stylesheet through the command line invocation of Saxon, which allows for such parameter passing. Scripting could also allow for the interactive scores, where various elements would be present, but not in the correct place, and students could complete the score by dragging notes and rests to right spot before submitting their answers.

ScoreSVG could include a number of ways to parse and analyze the XML generated for various music theory purposes, and the means to hear the score could be achieved with a different XSL stylesheet, one that would create a .wav or .mp3 file, both of which are readily playable using the Java sound API.

Making ScoreSVG web-accessible with Java WebStart technology would add greatly to its usability. WebStart technology allows the Web application to read and write to files on a client system with a carefully constructed protocol. Users could access ScoreSVG on the web,

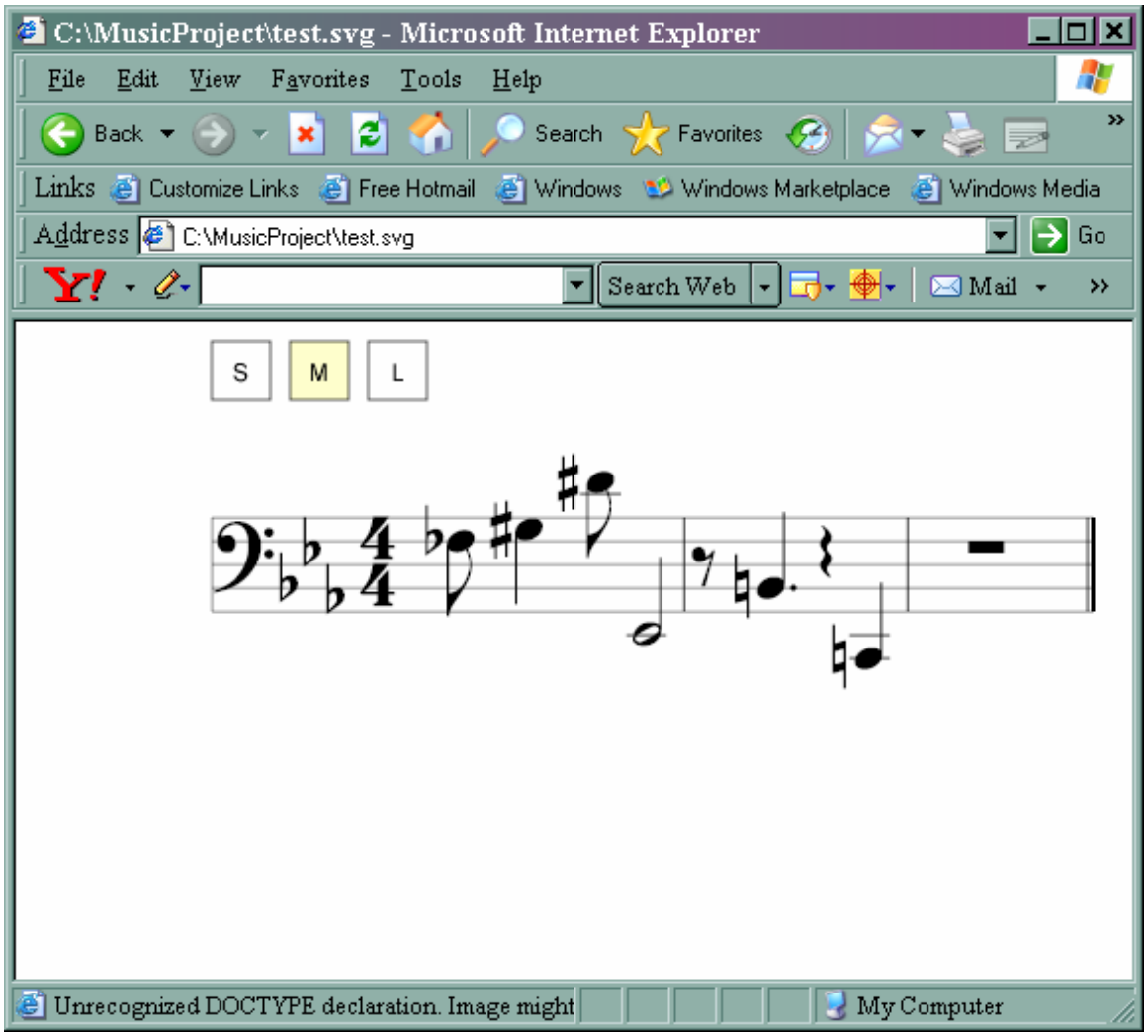

Figure 10: Example 6 With JavaScript

and create SVG files on their own systems. This feature will be pursued as a short-term goal for ScoreSVG, especially since there are uses for it as a means of preserving public domain music sores digitally.

Further consultation with musical associates both known and unknown will undoubtedly bring more excellent ideas forward that can be implemented on the general software framework of ScoreSVG. It is also possible that ScoreSVG will be placed in SourceForge or some other repository as an open source project, so that other talents may be brought to bear upon it and a wealth of features added with effort and delight spread far and wide.

REFERENCES:

[1] Kurt Cagle, SVG Programming: The Graphical Web, Apress, New York,N.Y. 2002

[2] Michael Kay, XSLT 2.0 Third Edition, Wiley Publishing, Indianapolis, 2004

[3] Sal Mangano, XSLT Cookbook, O'Reilly media, Sebastopol, CA 2003

[4] H. Merten, S. Simon, A. Berndt, Guido to XML: Einbettung des Guido Formats in ein XML konformes Modell, Technishe Universitat Darmstadt, 2001.

[5] http://www.informatik.tu-darmstadt.de/AFS/GUIDO/
Guido Homepage

[6] http://www.noteserver.org/noteserver.html
Guido NoteServer

[7] http://www.high-logic.com/fcp.html
FontCreator 5.0

[8] http://lilypond.org/web/
LilyPond Music Editor

[9] http://xml.coverpages.org/xmlMusic.html
Music XML formats

[10] http://ace.acadiau.ca/score/others.htm
Acadia University: music score editors

[11] http://www.ulead.com/
home of PhotoImpact

[12] http://www.musicmarkup.info/scope/markuplanguages.html#4ML
more music XML

[13] Laura O'Shea, Stirring XML: Visualizations in SVG No. 2: MusicML2SVG, Proceedings,SVGOpen2003 Conference, 2003
http://www.svgopen.org/2003/papers/StirringXml-VisualisationsInSVG/#S2.

[14] http://www.recordare.com/
Recordare, home of MusicXML

[15] http://saxon.sourceforge.net
and http://www.saxonica.com/ Home of the Saxon XSLT processor

[16] http://www.syncedit.com/software/javalauncher/
Home of JavaLauncher software

[17] http://www.aha-soft.com/
Aha-soft, home of Icon-Lover

[18] http://www.adobe.com/svg/main.html
Home of Adobe SVG plugin and information

[19] http://xml.apache.org/batik/
Home of the Apache Batik project, which includes the Batik SVG viewer.

[20] http://www.w3.org/TR/xlink/
The W3C XLink specification

[21] http://www.finalemusic.com/
Home of the Finale Score Editor

[22] http://www.sibelius.com/
Home of the Sibelius Score Editor

[23] http://www.w3.org/TR/SVG/intro.html
The W3C SVG specification.

[24] Michael Kay, *XPath 2.0.* Third Edition. Indianapolis: Wiley Publishing, 2004.

[25] J.David Eisenberg, *SVG Essentials*, Sebastopol,CA: O'Reilly, 2002.

[26] Martin Fowler, Kendall Scott. UML Distilled. 2nd ed. Indianapolis: Addison-Wesley, 2000.

[27] Eric Gamma, Richard Helm, Ralph Johnson, John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Indianapolis: Addison-Wesley, 1995.

[28] Cay S. Horstmann, Gary Cornell, *Core Java 2*. 2 vols. Santa Clara, CA: Prentice Hall, 2005.

[29] Geoffrey Bays, Ying Zhu, *ScoreSVG: A Web-Based Music Editor for Creating Scores in SVG*, Proceedings,SVGOpen2005 Conference, Entschede, Netherlands, 2005. http://www.svgopen.org/2005/paperAbstracts/ScoreSVG2.html

APPENDIX A:

DTD of the Subset of GuidoXML used in ScoreSVG:
(NOT the modified GuidoXML)

```
<?xml version="1.0" encoding="UTF-8"?>
<!--System-->
<!ELEMENT guido (segment)>
<!ELEMENT segment (sequence+)>
<!ELEMENT sequence ANY>
<!ELEMENT clef EMPTY>
<!ATTLIST clef
      s CDATA #REQUIRED
>
<!ELEMENT key EMPTY>
<!ATTLIST key
      s CDATA #IMPLIED
      i CDATA #IMPLIED
```

```
>
<!ELEMENT meter EMPTY>
<!ATTLIST meter
      s CDATA #REQUIRED
>
<!ELEMENT note EMPTY>
<!ATTLIST note
      name CDATA #REQUIRED
      octave CDATA #IMPLIED
      duration CDATA #IMPLIED
      accidentals CDATA #IMPLIED
>
<!ELEMENT rest EMPTY>
<!ATTLIST rest
      duration CDATA #IMPLIED
      dot CDATA #IMPLIED
>

<!--Stems-->
<!ELEMENT stemsUp EMPTY>
<!ATTLIST stemsUp
      dy CDATA #IMPLIED
>
<!ELEMENT stemsDown EMPTY>
<!ATTLIST stemsDown
      dy CDATA #IMPLIED
>
<!--Bars-->
<!ELEMENT bar EMPTY>
<!ATTLIST bar
      i CDATA #IMPLIED
>
<!ELEMENT doubleBar EMPTY>
<!ATTLIST doubleBar
      i CDATA #IMPLIED
>



      .
```

APPENDIX B: ScoreSVG Source Code

```
package ScoreSVG;

import java.io.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.font.*;
import javax.swing.*;
import java.util.*;
import java.awt.geom.*;
import java.awt.Graphics;
import java.lang.Runtime;

public class MainGUI extends JFrame {

   protected JFrame jframe;
   protected Container content;
   protected JPanel topPanel;
   protected JPanel noteRestPanel;
   protected JPanel sidePanel;
   protected Canvas canvas;
   protected JScrollPane scroller;
   protected JButton chordB;
   protected JButton beamsetB;
   protected JButton wholeNoteB;
   protected JButton halfNoteB;
   protected JButton quarterNoteB;
   protected JButton eighthNoteB;
   protected JButton sxthNoteB;
   protected JButton noteDotB;
   protected JButton sharpB;
   protected JButton flatB;
   protected JButton naturalB;
   protected JButton wholeRestB;
   protected JButton halfRestB;
   protected JButton quarterRestB;
   protected JButton eighthRestB;
   protected JButton sxthRestB;
   protected JButton restDotB;
   protected JButton deleteB;
   protected JButton staffB;
   protected JButton meterB;
   protected JButton barB;
   protected JButton doubleBarB;
   protected JButton slurB;
   protected JButton finishB;
   protected JButton space1B;
   protected JButton space2B;
   protected JButton newseqB;
   protected JButton space4B;
   protected JButton space5B;
   protected JButton space6B;
```

```
 protected JButton space7B;
 protected JButton stemsupB;
 protected JButton stemsdownB;
 protected JButton keyB;
 protected JButton setupB;
 protected JButton[] noteRestButtonList;
 protected JButton[] sideButtonList;
 protected JButton[] topButtonList;
 protected ActionEventHandler handler;
 protected JRadioButton newSeq;
 protected JRadioButton oldSeq;
 private Font font;
 private Font smallFont;
 private Font musicFont;
 private Font musicFont2;
 private String clefStr;
// private boolean isNewSeq;
 private boolean dotOn;
 private boolean restdotOn;
 private boolean sharpOn;
 private boolean flatOn;
 private boolean naturalOn;
 private RadioButtonHandler rbHandler;
 private Graphics g;
 protected ScorePanel  scP;
 protected LinkedList shapeList;
 protected LinkedList  charDrawList;
 private Dimension area;
 private int width;
 private int height;
 private String noteType;
 private CharDraw current = null;
 private int lineSpace;
 private String stateArray[];
 protected static HashMap stateHash;
 private Color OnColor;
 private Color color;
 private int fontSize;
 protected Storage storage;
 private Segment segment;
 private SecondaryGUI sgu;

  /** Creates new form MainGUI */
  public MainGUI() {

      sgu = new SecondaryGUI();
      segment = Segment.getInstance();
      storage = Storage.getInstance();
      lineSpace = storage.getLineSpace();
      shapeList = storage.getShapeList();
      charDrawList = storage.getCharDrawList();
      fontSize = storage.getFontSize();

      this.createMusicFont();

      stateArray = new String[]{"ZZ","ZZ","ZZ","ZZ","ZZ","ZZ"};
```

```
        stateHash = new HashMap(12);
        stateHash.put("name","ZZ");
        stateHash.put("noteduration","ZZ");
        stateHash.put("restduration","ZZ");
        stateHash.put("notedot","ZZ");
        stateHash.put("restdot","ZZ");
        stateHash.put("stems","stemsup");
        stateHash.put("accidental","ZZ");
        stateHash.put("xcoord",new Integer(0));
        stateHash.put("ycoord",new Integer(0));
        stateHash.put("uppernum","ZZ");
        stateHash.put("lowernum","ZZ");
        stateHash.put("keynumber","ZZ");
        stateHash.put("seqnumber",new Integer(0));

        noteType = "";
        dotOn = false;
        restdotOn = false;
        sharpOn = false;
        flatOn = false;
        naturalOn = false;

        //shapeList = new LinkedList();
       // charDrawList = new LinkedList();
        font = new Font("Arial",Font.PLAIN,11);
        smallFont = new Font("Arial", Font.PLAIN, 9);
        JFrame.setDefaultLookAndFeelDecorated(true);
        jframe = new JFrame("ScoreSVG");

        jframe.setLocation(250,30);
        jframe.addWindowListener(
              new WindowAdapter(){
                public void windowClosing(WindowEvent we)
                {
                  jframe.setVisible(false);
                  System.exit(0);
                }

           }

        );


        color = new Color(0x9c,0x9b,0xCF);
        OnColor = new Color(0xF6,0xB3,0xAD);

        area = new Dimension(550,600);
        scP = new ScorePanel();
        scP.setPreferredSize(area);
        scP.setVisible(true);
        scP.setBackground(Color.WHITE);
        //scP.setForeground(color);

        scroller = new
JScrollPane(scP,JScrollPane.VERTICAL_SCROLLBAR_ALWAYS,
                                    JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
```

```
        scroller.setBackground(color);
       // scroller.setForeground(color);
        handler = new ActionEventHandler();

        topPanel = new JPanel(new GridLayout(1,2));

        topPanel.setBackground(new Color(213,220,239));
        topPanel.setBorder(BorderFactory.createEmptyBorder(30,30,5,30));
        noteRestPanel = new JPanel(new GridLayout(3,10));

        noteRestPanel.setBackground(new Color(213,220,239));

noteRestPanel.setBorder(BorderFactory.createEmptyBorder(30,30,5,30));
        sidePanel = new JPanel(new GridLayout(7,1));
        sidePanel.setBackground(new Color(213,220,239));
        sidePanel.setBorder(BorderFactory.createEmptyBorder(30,30,5,30));

        chordB = new JButton("Chord");
        chordB.addActionListener(handler);
        chordB.setBackground(color);
        chordB.setFont(font);
        keyB = new JButton("Key");
        keyB.addActionListener(handler);
        keyB.setBackground(color);
        setupB = new JButton("Set Up");
        setupB.addActionListener(handler);
        setupB.setBackground(color);
        beamsetB = new JButton("Beam");
        beamsetB.addActionListener(handler);
        beamsetB.setBackground(color);
        beamsetB.setFont(font);
        noteDotB = new JButton("Dot");
        noteDotB.addActionListener(handler);
        noteDotB.setBackground(color);
        restDotB = new JButton("Dot");
        restDotB.addActionListener(handler);
        restDotB.setBackground(color);
        deleteB = new JButton("Delete");
        deleteB.addActionListener(handler);
        deleteB.setBackground(color);
        staffB = new JButton("Staff");
        staffB.addActionListener(handler);
        staffB.setBackground(color);
        meterB = new JButton("Meter");
        meterB.addActionListener(handler);
        meterB.setBackground(color);
        barB = new JButton("Bar");
        barB.addActionListener(handler);
        barB.setBackground(color);
        barB.setFont(font);
        doubleBarB = new JButton("DblBar");
        doubleBarB.addActionListener(handler);
        doubleBarB.setBackground(color);
        doubleBarB.setFont(smallFont);
        slurB = new JButton("Slur");
```

```
slurB.addActionListener(handler);
slurB.setBackground(color);
finishB = new JButton("FINISH");
finishB.addActionListener(handler);
finishB.setBackground(color);
space1B = new JButton("");
space1B.addActionListener(handler);
space1B.setBackground(color);
space2B = new JButton("");
space2B.addActionListener(handler);
space2B.setBackground(color);
newseqB = new JButton("Seq");
newseqB.addActionListener(handler);
newseqB.setBackground(color);
newseqB.setFont(font);
space4B = new JButton("");
space4B.addActionListener(handler);
space4B.setBackground(color);
space5B = new JButton("");
space5B.addActionListener(handler);
space5B.setBackground(color);
space6B = new JButton("");
space6B.addActionListener(handler);
space6B.setBackground(color);
space7B = new JButton("");
space7B.addActionListener(handler);
space7B.setBackground(color);
stemsupB = new JButton("Up");
stemsupB.addActionListener(handler);
stemsupB.setBackground(color);
stemsupB.setFont(font);
stemsdownB = new JButton("Down");
stemsdownB.addActionListener(handler);
stemsdownB.setBackground(color);
stemsdownB.setFont(font);


wholeRestB = new JButton("4/4");
wholeRestB.addActionListener(handler);
wholeRestB.setBackground(color);
halfRestB = new JButton("1/2");
halfRestB.addActionListener(handler);
halfRestB.setBackground(color);
quarterRestB = new JButton("1/4");
quarterRestB.addActionListener(handler);
quarterRestB.setBackground(color);
eighthRestB = new JButton("1/8");
eighthRestB.addActionListener(handler);
eighthRestB.setBackground(color);
sxthRestB = new JButton("1/16");
sxthRestB.addActionListener(handler);
sxthRestB.setBackground(color);
wholeNoteB = new JButton("4/4");
wholeNoteB.addActionListener(handler);
wholeNoteB.setBackground(color);
halfNoteB = new JButton("1/2");
```

```
        halfNoteB.addActionListener(handler);
        halfNoteB.setBackground(color);
        quarterNoteB = new JButton("1/4");
        quarterNoteB.addActionListener(handler);
        quarterNoteB.setBackground(color);
        eighthNoteB = new JButton("1/8");
        eighthNoteB.addActionListener(handler);
        eighthNoteB.setBackground(color);
        sxthNoteB = new JButton("1/16");
        sxthNoteB.addActionListener(handler);
        sxthNoteB.setBackground(color);
        sharpB = new JButton("sharp");
        sharpB.addActionListener(handler);
        sharpB.setBackground(color);
        sharpB.setFont(font);
        flatB = new JButton("flat");
        flatB.addActionListener(handler);
        flatB.setBackground(color);
        flatB.setFont(font);
        naturalB = new JButton("natural");
        naturalB.addActionListener(handler);
        naturalB.setBackground(color);
        naturalB.setFont(smallFont);

        content = jframe.getContentPane();

      /*  topButtonList = new JButton[]{chordB,beamsetB};
        for(int i = 0; i < topButtonList.length; i++){
           topPanel.add(topButtonList[i]);
        }
      */
        noteRestPanel.add(new JLabel("Notes:",JLabel.CENTER));

        noteRestButtonList = new
JButton[]{wholeNoteB,halfNoteB,quarterNoteB,
                                        eighthNoteB,sxthNoteB,noteDotB,
                                        sharpB,flatB,naturalB};


        for(int i = 0; i < noteRestButtonList.length; i++){
           noteRestPanel.add(noteRestButtonList[i]);
        }

        noteRestPanel.add(new JLabel("Rests:",JLabel.CENTER));
        noteRestPanel.add(wholeRestB);
        noteRestPanel.add(halfRestB);
        noteRestPanel.add(quarterRestB);
        noteRestPanel.add(eighthRestB);
        noteRestPanel.add(sxthRestB);
        noteRestPanel.add(restDotB);
       // noteRestPanel.add(space1B);
       // noteRestPanel.add(space2B);
        noteRestPanel.add(space7B);
        noteRestPanel.add(space5B);
        noteRestPanel.add(space6B);
```

```
        noteRestPanel.add(new JLabel("Stems:",JLabel.CENTER));
        noteRestPanel.add(stemsupB);
        noteRestPanel.add(stemsdownB);
        noteRestPanel.add(chordB);
        noteRestPanel.add(beamsetB);
        noteRestPanel.add(barB);
        noteRestPanel.add(doubleBarB);
        noteRestPanel.add(newseqB);
        noteRestPanel.add(space4B);

        sideButtonList = new JButton[]{setupB,staffB,meterB,keyB,deleteB,
                                       slurB,finishB};

        for(int i = 0; i < sideButtonList.length; i++){
           sidePanel.add(sideButtonList[i]);
        }

       //topPanel.add(noteRestPanel);
      // content.add(topPanel,BorderLayout.NORTH);
       content.add(noteRestPanel,BorderLayout.NORTH);
       content.add(topPanel,BorderLayout.SOUTH);
       content.add(scroller,BorderLayout.CENTER);
         content.add(sidePanel,BorderLayout.WEST);


        initComponents();
        jframe.setSize(700,550);
        jframe.setVisible(true);


    }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    private void initComponents() {//GEN-BEGIN:initComponents

        addWindowListener(new java.awt.event.WindowAdapter() {
            public void windowClosing(java.awt.event.WindowEvent evt) {
                exitForm(evt);
            }
        });

        pack();
    }//GEN-END:initComponents

    /** Exit the Application */
    private void exitForm(java.awt.event.WindowEvent evt) {//GEN-
FIRST:event_exitForm
        System.exit(0);
    }//GEN-LAST:event_exitForm

 private void viewStaffGUI(){
```

```
 rbHandler = new RadioButtonHandler();
 String clefNames[] = new String[]{"G clef","F clef"};
 final JFrame frame = new JFrame("Select clef for a Staff");
 final JPanel panel = new JPanel(new GridLayout(2,3));
 Container c = frame.getContentPane();
 c.setLayout(new GridLayout(2,1));
 //c.setLayout(new GridLayout(3,3));
 JComboBox clefCB = new JComboBox(clefNames);

 clefCB.addActionListener(
    new ActionListener(){
       public void actionPerformed(ActionEvent ae)
       {
           JComboBox cb = (JComboBox)ae.getSource();
           clefStr = (String)cb.getSelectedItem();
         //  System.out.println(clefStr);
       }

     }
 );

 ButtonGroup bg = new ButtonGroup();
 JLabel seqLabel = new JLabel("New Sequence?");
 JLabel chooseClef = new JLabel("Choose a Clef: ");

// JButton clearB = new JButton("clear");
 JButton finishB = new JButton("finish");

 finishB.addActionListener(
     new ActionListener(){
       public void actionPerformed(ActionEvent ae)
       {
           Staff staff = new Staff(clefStr,stateHash);
          Shape s  = staff.getPath();
          CharDraw cd = staff.getCharDraw();
         // LinkedList ll = new LinkedList();
         // ll.add(cd);
         // scP.getCharDraw(ll);
         // scP.getShape(s);
          int newY = staff.getMaxYDimension();
           if(newY > area.height){
               Dimension dim = new Dimension(area.width, newY);
               scP.setPreferredSize(dim);
           }

           scP.revalidate();
           scP.repaint();


           frame.setVisible(false);

       }

 }

 );
```

```
    newSeq = new JRadioButton("Yes",true);
    newSeq.addActionListener(rbHandler);
    oldSeq = new JRadioButton("No",true);
    oldSeq.addActionListener(rbHandler);

    bg.add(newSeq);
    bg.add(oldSeq);
    panel.add(seqLabel);
    panel.add(newSeq);
    panel.add(oldSeq);
    panel.add(chooseClef);
    panel.add(clefCB);
   //panel.add(finishB);

    c.add(panel);
    c.add(finishB);
    frame.pack();
    frame.setVisible(true);
    frame.setLocation(250, 35);
}



public void createMusicFont(){

        try{

       FileInputStream fin = new
FileInputStream("C:\\SVGProject\\Maestro\\MAESTRO_.TTF");

       DataInputStream din = new DataInputStream(fin);
       musicFont2 = Font.createFont(Font.PLAIN,din);



        }catch(IOException io){
            System.out.println(io.getMessage());
        }catch(FontFormatException ffe){
            System.out.println(ffe.getMessage());
        }
}

public Font setMusicFont(int size){



      musicFont = musicFont2.deriveFont((float)size);

        return musicFont;
}

public void setNoteType(String note){

     noteType = new String(note);
```

```
}
public String getNoteType(){
    return noteType;
}
public CharDraw find(Point2D point){
   /*  ListIterator shpItr = shapeList.listIterator();
     while(shpItr.hasNext()){
         Shape s = (Shape)shpItr.next();

         if(s.contains(point)){
            return s;
         }

     }   */

    ListIterator cdItr = charDrawList.listIterator();
    while(cdItr.hasNext()){
        CharDraw cdr = (CharDraw)cdItr.next();
        if(cdr.getRectangle().contains(point)){

            return cdr;
        }

    }
   return null;
}

public void remove(CharDraw s){
    if(s == null) return;
    if(s == current) current = null;
    s.getNoteXML().finalize();
  //  int sizeb = charDrawList.size();
   // System.out.println("list size before removal: " +
Integer.toString(sizeb));
    charDrawList.remove(s);
   // int sizea = charDrawList.size();
   // System.out.println("list size after removal: " +
Integer.toString(sizea));
    scP.repaint();

}

class ScorePanel extends JPanel{
    private Shape shp;
    private CharDraw cd;
    private Storage storage;

    public ScorePanel(){
       addMouseListener(new MouseHandler());
       addMouseMotionListener(new MouseMotionHandler());
       current = null;
       storage = Storage.getInstance();

    }
```

```
 public Shape getShape(Shape s){
    shp = s;
    shapeList.add(s);
    return shp;

 }

 public void getRectangle(Rectangle2D rec){
     shapeList.add(rec);

 }

 public void getCharDraw(LinkedList cdP){
    // cd = cdP;
     charDrawList.addAll(cdP);
   //  return cd;
 }

 public void paintComponent(Graphics g){


    super.paintComponent(g);
     fontSize = storage.getFontSize();
     musicFont = musicFont2.deriveFont((float)fontSize);
    Graphics2D g2 = (Graphics2D) g;
    g2.setFont(musicFont);
    g2.setPaint(Color.black);
    ListIterator shpItr = shapeList.listIterator();
    ListIterator charItr = charDrawList.listIterator();
    try{
      while(shpItr.hasNext()){
          Object ob = shpItr.next();
          if(ob instanceof Rectangle2D.Double){
              g2.fill((Rectangle2D.Double)ob);
          }
          else{
             g2.draw((Shape)ob);
          }
      }
      while(charItr.hasNext()){
          CharDraw cds = (CharDraw)charItr.next();
          char c = cds.getChar();
          double array[] = cds.getXY();
         g2.drawString("" +
c,(int)Math.round(array[0]),(int)Math.round(array[1]));
      }
    }catch (Exception e){

    }


 }
```

```
}

private class RadioButtonHandler implements ActionListener{
    public void actionPerformed(ActionEvent e)
    {
      if(e.getSource() == newSeq)
         System.out.println("New System");
      else if(e.getSource() == oldSeq)
          System.out.println("Old System");

    }

}
 private class ActionEventHandler implements ActionListener{

       public void actionPerformed(ActionEvent ae)
         {
        if(ae.getSource() == staffB){
            viewStaffGUI();

         }
        else if(ae.getSource() == setupB){
            sgu.viewSetupGUI();
        }
        else if(ae.getSource() == wholeNoteB){
            setNoteType("whole");
            stateHash.put("name","note");
            stateHash.put("noteduration","4/4");
        }
        else if(ae.getSource() == halfNoteB){
            stateHash.put("name","note");
            stateHash.put("noteduration","1/2");
        }
        else if(ae.getSource() == quarterNoteB){
            stateHash.put("name","note");
            stateHash.put("noteduration","1/4");
        }
        else if(ae.getSource() == eighthNoteB){
            stateHash.put("name","note");
            stateHash.put("noteduration","1/8");
        }
        else if(ae.getSource() == sxthNoteB){
            stateHash.put("name","note");
            stateHash.put("noteduration","1/16");
        }
        else if(ae.getSource() == wholeRestB){
            stateHash.put("name","rest");
            stateHash.put("restduration","4/4");
        }
        else if(ae.getSource() == halfRestB){
            stateHash.put("name","rest");
            stateHash.put("restduration","1/2");
        }
         else if(ae.getSource() == quarterRestB){
            stateHash.put("name","rest");
            stateHash.put("restduration","1/4");
```

```
}
else if(ae.getSource() == eighthRestB){
    stateHash.put("name","rest");
    stateHash.put("restduration","1/8");
}
else if(ae.getSource() == sxthRestB){
    stateHash.put("name","rest");
    stateHash.put("restduration","1/16");
}
else if(ae.getSource() == noteDotB){
    stateHash.put("name","note");
    if(dotOn == false){
    stateHash.put("notedot","yes");
    dotOn = true;
    noteDotB.setBackground(OnColor);
    }
    else if(dotOn == true){
    stateHash.put("notedot","no");
    dotOn = false;
    noteDotB.setBackground(color);
    }

}
else if(ae.getSource() == restDotB){
    stateHash.put("name","rest");
    if(restdotOn == false){
    stateHash.put("restdot","yes");
    restdotOn = true;
    restDotB.setBackground(OnColor);
    }
    else if(restdotOn == true){
    stateHash.put("restdot","no");
    restdotOn = false;
    restDotB.setBackground(color);
    }
}
else if(ae.getSource() == sharpB){
    flatOn = false;
    naturalOn = false;
    stateHash.put("name","note");
    if(sharpOn == false){
    stateHash.put("accidental","sharp");
    sharpOn = true;
    sharpB.setBackground(OnColor);
    }
    else if(sharpOn == true){
     stateHash.put("accidental","ZZ");
     sharpOn = false;
     sharpB.setBackground(color);
    }
}
else if(ae.getSource() == flatB){
    sharpOn = false;
    naturalOn = false;
    stateHash.put("name","note");
    if(flatOn == false){
```

```
    stateHash.put("accidental","flat");
    flatOn = true;
    flatB.setBackground(OnColor);
    }
    else if(flatOn == true){
      stateHash.put("accidental","ZZ");
      flatOn = false;
      flatB.setBackground(color);
    }

}
else if(ae.getSource() == naturalB){
    sharpOn = false;
    flatOn = false;
    stateHash.put("name","note");
    if(naturalOn == false){
    stateHash.put("accidental","natural");
    naturalOn = true;
    naturalB.setBackground(OnColor);
    }
    else if(naturalOn == true){
    stateHash.put("accidental","ZZ");
    naturalOn = false;
    naturalB.setBackground(color);
    }
}
else if(ae.getSource() == stemsupB){
    stateHash.put("name","note");
    stateHash.put("stems","stemsup");
}
else if(ae.getSource() == stemsdownB){
    stateHash.put("name","note");
    stateHash.put("stems","stemsdown");
}
else if(ae.getSource() == barB){
    stateHash.put("name","bar");
}
else if(ae.getSource() == doubleBarB){
    stateHash.put("name","doublebar");
}
else if(ae.getSource() == meterB){
    sgu.viewMeterGUI(stateHash);
}
else if(ae.getSource() == keyB){
    sgu.viewKeyGUI(stateHash);
}
else if(ae.getSource() == newseqB){
    int sn = ((Integer)stateHash.get("seqnumber")).intValue();
    stateHash.put("seqnumber",new Integer(sn + 1));

}
else if(ae.getSource() == finishB){
    segment.getXMLObjects();
    segment.processXYChange();
    segment.toString();
    segment.writeXML();
```

```
           // String cmd[] = {"cmd.exe","cd","C:\\SVGProject"};
            String cmd2[] = {"java","-jar","C:\\SVGProject\\saxon8.jar","-
o","C:\\SVGProject\\score1.svg","C:\\SVGProject\\xoutput.xml","C:\\SVGProje
ct\\ScoreXSL.xsl"};
            try{
            //Process p = Runtime.getRuntime().exec(cmd);
            Process p2 = Runtime.getRuntime().exec(cmd2);
            }catch(IOException io){
                System.out.println("Unable to execute XSLT" +
io.getMessage());
                System.err.println(io);
            }
        }
    }


}

 private class MouseHandler extends MouseAdapter{

     public void mousePressed(MouseEvent me){
        Point2D point = me.getPoint();
        current = find(point);
        if (current == null){
        double x = point.getX();
        double y = point.getY();
        int xI = (int)x;
        int yI = (int)y;
        stateHash.put("xcoord",new Integer(xI));
        stateHash.put("ycoord",new Integer(yI));
        if(stateHash.get("name").equals("note")){
        Note note = new Note(stateHash);
        LinkedList cdf = note.getCharDraw();
       // scP.getCharDraw(cdf);
        }
        else if(stateHash.get("name").equals("rest")){
            Rest rest = new Rest(stateHash);
        LinkedList cdf = rest.getCharDraw();
       // scP.getCharDraw(cdf);
        }
        else if(stateHash.get("name").equals("bar")){
            Bar bar = new Bar(stateHash);
          //  LinkedList cdf = bar.getCharDraw();
            // scP.getCharDraw(cdf);
            Rectangle2D s = bar.getPath();
          //   LinkedList ll = new LinkedList();
            //  ll.add(s);
          //   scP.getRectangle(s);
        }
        else if(stateHash.get("name").equals("doublebar")){
           DoubleBar db = new DoubleBar(stateHash);
         //  LinkedList cdf = db.getCharDraw();
          //  scP.getCharDraw(cdf);
           Rectangle2D r = db.getPath();
        }
        else if(stateHash.get("name").equals("meter")){
```

```
            Meter meter = new Meter(stateHash);
            LinkedList ll = meter.getCharDraw();
           // scP.getCharDraw(ll);

        }

        else if(stateHash.get("name").equals("key")){
            Key key = new Key(stateHash);
            LinkedList ll = key.getCharDraw();
        }
       // Shape shp = note.getPath(x,y);
       // scP.getShape(shp);
        scP.repaint();


      // System.out.println(stateHash.toString());
        }
    }

    public void mouseClicked(MouseEvent me){

        current = find(me.getPoint());
        if(current != null && me.getClickCount() >= 2){
            remove(current);
        }

    }


}

private class MouseMotionHandler implements MouseMotionListener{

    public void mouseDragged(MouseEvent e) {
        int halflineS = (int)Math.ceil(lineSpace*0.5);
        if(current != null){

          double x = e.getX();
          double y = e.getY();
          int xI = (int)x;
          int yI = (int)y;
          current.setXY(x - halflineS,y + halflineS);
        /*  if(current.getNoteXML() != null){
             current.getNoteXML().setXY(x,y);

          }   */

          if(current.getXMLObject() != null){
             current.getXMLObject().setXY(x,y) ;
          }
         // current.setXY(xI-halflineS,yI + halflineS);
          current.setRectangle(xI,yI,xI-halflineS,yI - halflineS);
         // Rectangle2D rect = new Rectangle2D.Double();
        //  Rectangle2D rect = current.getBounds2D();
        //  rect.setFrame(x-0.5*lineSpace,y-
0.5*lineSpace,1.4*lineSpace,0.95*lineSpace);
```

```
            scP.repaint();
        }

    }

    public void mouseMoved(MouseEvent e) {
    /*
        if(find(e.getPoint()) == null){
            setCursor(Cursor.getDefaultCursor());
        }
        else{
            setCursor(Cursor.getPredefinedCursor(Cursor.CROSSHAIR_CURSOR));
        }  */
    }

}

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
       MainGUI mg =  new MainGUI();


    }


    // Variables declaration - do not modify//GEN-BEGIN:variables
    // End of variables declaration//GEN-END:variables

}
```

```
package ScoreSVG;

import java.awt.*;
import java.awt.Graphics;
import java.awt.geom.*;
import javax.swing.*;
import java.util.*;
import java.lang.Math;

public class Note {
  /*  private String name;
    private String octave;
    private String duration;
    private String accidentals;
    private String output; */

    private String clefStr;
    private String dotString;
    private String duration;
```

```
 private String accidentals;
 private int lineSpace;
 private HashMap hash;
 private int xcoord;
 private int ycoord;
 private int halflineS;
 private CharDraw cd;
 private CharDraw cdDot;
 private LinkedList charList;
 private LinkedList charDrawList;
 private double startY;
 private double posDouble;
 private double halflineD;
 private int posFactor;
 private int factor;
 private double finalY;
 private int spaceBetweenStaffs;
 private int upperLeftY;
 private int seqNumber;
 private Storage storage;
 private Segment segment;
 private HashMap noteHashG;
 private HashMap noteHashF;
// private XMLObject nxml;


 public Note(HashMap hashP) {

     storage = Storage.getInstance();
     lineSpace = storage.getLineSpace();
     halflineS = storage.getHalfLineS();

     upperLeftY = storage.getUpperLeftY();
     spaceBetweenStaffs = storage.getSpaceBetweenStaffs();

     noteHashG = storage.getNoteHashG();
     noteHashF = storage.getNoteHashF();
     charDrawList = storage.getCharDrawList();
     segment = Segment.getInstance();



     hash = new HashMap();
     hash.putAll(hashP);
     xcoord = ((Integer)hash.get("xcoord")).intValue();
     ycoord = ((Integer)hash.get("ycoord")).intValue();
     dotString = (String)hash.get("notedot");
     duration = (String)hash.get("noteduration");
     accidentals = (String)hash.get("accidental");
     seqNumber = ((Integer)hash.get("seqnumber")).intValue();


     cd = new CharDraw("noname",'z',3,3);
     charList = new LinkedList();
     startY = 0;
     posDouble = 0;
```

```
        posFactor = 0;
        factor = -99;
        finalY = 0;
        halflineD = lineSpace*0.5;



    }


 public LinkedList getCharDraw(){

     int count = 0;
     //factor = (int)Math.floor((ycoord)/spaceBetweenStaffs);
     factor = (int)Math.floor((ycoord - upperLeftY + ((spaceBetweenStaffs -
4*lineSpace)/2))/spaceBetweenStaffs);
     startY = ycoord  - upperLeftY - factor*spaceBetweenStaffs;
      posDouble = startY/halflineD;
      posFactor = (int)Math.rint(posDouble);
      finalY = posFactor*halflineD  + upperLeftY +
factor*spaceBetweenStaffs;
      double upperStaffCornerY = upperLeftY + factor*spaceBetweenStaffs;
      double lowerStaffCornerY = upperStaffCornerY + 4*lineSpace;

      if(posFactor < -1){
         for(int i = -1; i > posFactor; i = i - 2){
             count++;
             char cl = '\uF05F';
            CharDraw cline = new CharDraw("ledgerline",cl,xcoord -
1.5*halflineS,upperStaffCornerY - count*lineSpace);
            cline.setRectangle(xcoord -  1.5*halflineS,upperStaffCornerY +
count*lineSpace,xcoord- 2.5*lineSpace,upperStaffCornerY - (count +
1)*lineSpace);
            charList.add(cline);
        }

      }
      else if(posFactor > 9){
          for(int j = posFactor - 9; j > 0; j = j - 2){
             count++;
             char cl = '\uF05F';
            CharDraw cline = new CharDraw("ledgerline",cl,xcoord -
1.5*halflineS,lowerStaffCornerY + count*lineSpace);
            cline.setRectangle(xcoord -  1.5*halflineS,lowerStaffCornerY +
count*lineSpace,xcoord - 2.5*lineSpace,lowerStaffCornerY + (count -
1)*lineSpace);
            charList.add(cline);
         }

      }

       char c = '\uF071';
       if(duration.equals("4/4")){
           c = '\uF077';
           cd = new CharDraw("wholenote",c,xcoord - halflineD,finalY);
         //  cd = new CharDraw("wholenote",c,xcoord - halflineS,ycoord);
```

```
            cd.setRectangle(xcoord,ycoord,xcoord - halflineS,ycoord -
halflineS);


            XMLObject nxml = new NoteXML();
            nxml.setNoteNameOctave(posFactor, factor);
            nxml.setXY(xcoord,finalY);
            nxml.setDuration(duration,dotString);
            nxml.setAccidentals(accidentals);
            nxml.setSeqNumber(seqNumber);
            cd.setXMLObject(nxml);


        }
        else if(duration.equals("1/2") &&
hash.get("stems").equals("stemsup")){
            c = '\uF068';
            cd = new CharDraw("halfnote",c,xcoord  - halflineS,finalY  +
halflineD);
           // cd = new CharDraw("halfnote",c,xcoord  - halflineS,ycoord  +
halflineS);
             cd.setRectangle(xcoord,ycoord,xcoord - halflineS,ycoord -
halflineS);

           XMLObject nxml = new NoteXML();
            nxml.setXY(xcoord,finalY);
            nxml.setNoteNameOctave(posFactor, factor);
            nxml.setDuration(duration,dotString);
            nxml.setAccidentals(accidentals);
            nxml.setSeqNumber(seqNumber);
            nxml.setStem("up");
           cd.setXMLObject(nxml);
        }
        else if(duration.equals("1/2") &&
hash.get("stems").equals("stemsdown")){
            c = '\uF048';
            cd = new CharDraw("halfnote",c,xcoord  - halflineS,finalY  +
halflineD);
            // cd = new CharDraw("halfnote",c,xcoord  - halflineS,ycoord +
halflineS);
              cd.setRectangle(xcoord,ycoord,xcoord - halflineS,ycoord -
halflineS);

            XMLObject nxml = new NoteXML();
            nxml.setXY(xcoord,finalY);
            nxml.setNoteNameOctave(posFactor, factor);
            nxml.setDuration(duration,dotString);
            nxml.setAccidentals(accidentals);
            nxml.setSeqNumber(seqNumber);
            nxml.setStem("down");
            cd.setXMLObject(nxml);

        }
       else if(duration.equals("1/4") &&
hash.get("stems").equals("stemsup")){
            c = '\uF071';
```

```
           cd = new CharDraw("quarternote",c,xcoord  - halflineS,finalY  +
halflineD);
          //  cd = new CharDraw("quarternote",c,xcoord  - halflineS,ycoord
+ halflineS);
             cd.setRectangle(xcoord,ycoord,xcoord - halflineS,ycoord -
halflineS);

          XMLObject nxml = new NoteXML();
           nxml.setXY(xcoord,finalY);
           nxml.setNoteNameOctave(posFactor, factor);
           nxml.setDuration(duration,dotString);
           nxml.setAccidentals(accidentals);
           nxml.setSeqNumber(seqNumber);
           nxml.setStem("up");
           cd.setXMLObject(nxml);

       }
       else if(duration.equals("1/4") &&
hash.get("stems").equals("stemsdown")){
           c = '\uF051';
           cd = new CharDraw("quarternote",c,xcoord  - halflineS,finalY  +
halflineD);
          // cd = new CharDraw("quarternote",c,xcoord  - halflineS,ycoord +
halflineS);
             cd.setRectangle(xcoord,ycoord,xcoord - halflineS,ycoord -
halflineS);

          XMLObject nxml = new NoteXML();
           nxml.setXY(xcoord,finalY);
           nxml.setNoteNameOctave(posFactor, factor);
           nxml.setDuration(duration,dotString);
           nxml.setAccidentals(accidentals);
           nxml.setSeqNumber(seqNumber);
           nxml.setStem("down");
          cd.setXMLObject(nxml);
       }
       else if(duration.equals("1/8") &&
hash.get("stems").equals("stemsup")){
           c = '\uF065';
           cd = new CharDraw("eighthnote",c,xcoord  - halflineS,finalY  +
halflineD);
          //  cd = new CharDraw("eighthnote",c,xcoord  - halflineS,ycoord +
halflineS);
            cd.setRectangle(xcoord,ycoord,xcoord - halflineS,ycoord -
halflineS);

           XMLObject nxml = new NoteXML();
           nxml.setXY(xcoord,finalY);
          nxml.setNoteNameOctave(posFactor, factor);
           nxml.setDuration(duration,dotString);
           nxml.setAccidentals(accidentals);
           nxml.setSeqNumber(seqNumber);
           nxml.setStem("up");
           cd.setXMLObject(nxml);
       }
```

```
       else if(duration.equals("1/8") &&
hash.get("stems").equals("stemsdown")){
           c = '\uF045';
           cd = new CharDraw("eighthnote",c,xcoord  - halflineS,finalY  +
halflineD);
         //  cd = new CharDraw("eighthnote",c,xcoord  - halflineS,ycoord +
halflineS);
             cd.setRectangle(xcoord,ycoord,xcoord - halflineS,ycoord -
halflineS);

           XMLObject nxml = new NoteXML();
           nxml.setXY(xcoord,finalY);
           nxml.setNoteNameOctave(posFactor, factor);
           nxml.setDuration(duration,dotString);
           nxml.setAccidentals(accidentals);
           nxml.setSeqNumber(seqNumber);
           nxml.setStem("down");
           cd.setXMLObject(nxml);
       }


       if(dotString.equals("yes")){
          char ch = '\uF02E';
          cdDot = new CharDraw("notedot",ch,xcoord + lineSpace,finalY);
         // cdDot = new CharDraw("notedot",ch,xcoord + lineSpace,ycoord);
          cdDot.setRectangle(xcoord,ycoord,xcoord - halflineS,ycoord -
halflineS);
         // cd.setRectangle(xcoord,ycoord,xcoord - 2*lineSpace,ycoord -
2*lineSpace);
          charList.add(cdDot);
       }

       if(accidentals.equals("sharp")){
           char ch = '\uF023';
           CharDraw cdSharp = new CharDraw("sharp",ch,xcoord - lineSpace -
halflineS,finalY);
         // CharDraw cdSharp = new CharDraw("sharp",ch,xcoord - lineSpace -
halflineS,ycoord);
          cdSharp.setRectangle(xcoord,ycoord,xcoord - halflineS,ycoord -
halflineS);
          charList.add(cdSharp);
       }
       else if(accidentals.equals("flat")){
           char ch = '\uF062';
           CharDraw cdFlat = new CharDraw("sharp",ch,xcoord - lineSpace -
halflineS,finalY);
         // CharDraw cdFlat = new CharDraw("sharp",ch,xcoord - lineSpace -
halflineS,ycoord);
          cdFlat.setRectangle(xcoord,ycoord,xcoord - halflineS,ycoord -
halflineS);
          charList.add(cdFlat);
       }
       else if(accidentals.equals("natural")){
           char ch = '\uF06E';
           CharDraw cdNatural = new CharDraw("sharp",ch,xcoord - lineSpace
- halflineS,finalY);
```

```
         // CharDraw cdNatural = new CharDraw("sharp",ch,xcoord - lineSpace
- halflineS,ycoord);
          cdNatural.setRectangle(xcoord,ycoord,xcoord - halflineS,ycoord -
halflineS);
          charList.add(cdNatural);
       }
       charList.add(cd);
       charDrawList.addAll(charList);

       return charList;
 }



  public Shape getPath(double x,double y){
    GeneralPath g2 = new GeneralPath();
    Rectangle2D rect = new Rectangle2D.Double(x-0.5*lineSpace, y-
0.5*lineSpace, 1.5*lineSpace, lineSpace);
    //Ellipse2D ellipse = new Ellipse2D.Double();
   // ellipse.setFrame(rect);

    g2.append(new Ellipse2D.Double(x-0.5*lineSpace, y-0.5*lineSpace,
1.4*lineSpace, 0.95*lineSpace),false);

    return g2;

  }
}
```

```
package ScoreSVG;

import java.util.*;
import java.lang.Math;


public class NoteXML extends XMLObject {


    private double x;
    private double y;
    private String name;
    private String stem;
    private String octave;
    private String duration;
    private String accidentals;
    private int seqNumber = 0;
    private String output;
    private String clefStr;
    private Storage storage;
```

```
  private double startY;
  private double posDouble;
  private double halflineD;
  private int posFactor;
  private int factor;
  private double finalY;
  private int spaceBetweenStaffs;
  private int upperLeftY;
  private int lineSpace;

  public NoteXML() {

      super();
       name = "";
       stem = "";
      octave = "";
      duration = "";
      accidentals = "";
      output = "";
      storage = Storage.getInstance();
      clefStr = "";
      posFactor = 0;
      factor = 0;
      posDouble = 0;
      x = 0;
      y = 0;
      finalY = 0;

      storage = Storage.getInstance();
      lineSpace = storage.getLineSpace();
      halflineD = lineSpace*0.5;


      upperLeftY = storage.getUpperLeftY();
      spaceBetweenStaffs = storage.getSpaceBetweenStaffs();

  }

protected void finalize(){ }

public void setName(String nameP) {name = new String(nameP);}
public String getName() {return name;}
public void setOctave(String octaveP) {octave = new String(octaveP);}
public String getOctave() {return octave;}
public void setS(String s){ }
public void setStem(String s){stem = s;}
public  String getStem(){return stem;}

public void setDuration(String durationP, String dot) {
    if(dot.equals("no") || dot.equals("ZZ")){
    duration = new String(durationP);
    }
    else if(dot.equals("yes")){
        if(durationP.equals("1/2")){
            duration = new String("3/4");
```

```
          }
          else if(durationP.equals("1/4")){
              duration = new String("3/8");
          }
          else if(durationP.equals("1/8")){
              duration = new String("3/16");
          }

      }
  }
  public String getDuration() {return duration;}
  public void setAccidentals(String accP) {accidentals = new String(accP);}
  public String getAccidentals() {return accidentals;}
  public double getX(){return x;}
  public double getY(){return y;}
  public double getFinalY(){return finalY;}
   public void setI(String i){ }



 public void setNoteNameOctave(int posFactor, int factor){
     String posFactorStr = Integer.toString(posFactor);
    if(factor > -2){
      clefStr = storage.getStaffClef(factor);
      if(clefStr.equals("G clef")){
      String noteOctaveStr =
(String)storage.getNoteHashG().get(posFactorStr);
        StringTokenizer tok = new StringTokenizer(noteOctaveStr,",");
        name = tok.nextToken();
        octave = tok.nextToken();
       // this.setName(name);
       // this.setOctave(octave);
      }
      else if(clefStr.equals("F clef")){

      String noteOctaveStr =
(String)storage.getNoteHashF().get(posFactorStr);
        StringTokenizer tok = new StringTokenizer(noteOctaveStr,",");
        name = tok.nextToken();
        octave = tok.nextToken();
       // this.setName(name);
       // this.setOctave(octave);

      }
    }



 }

 public void setNoteNameOctave(){
     String posFactorStr = Integer.toString(posFactor);

    if(factor > -1){
      clefStr = storage.getStaffClef(factor);
      if(clefStr.equals("G clef")){
```

```
       HashMap noteHashG = storage.getNoteHashG();
      String noteOctaveStr =  (String)noteHashG.get(posFactorStr);
//      System.out.println(posFactorStr);
        StringTokenizer tok = new StringTokenizer(noteOctaveStr,",");
        name = tok.nextToken();
        octave = tok.nextToken();
       // this.setName(name);
       // this.setOctave(octave);
      }
      else if(clefStr.equals("F clef")){

      String noteOctaveStr =
(String)storage.getNoteHashF().get(posFactorStr);
        StringTokenizer tok = new StringTokenizer(noteOctaveStr,",");
        name = tok.nextToken();
        octave = tok.nextToken();
       // this.setName(name);
       // this.setOctave(octave);

      }
    }



 }

     public void setXY(double xP,double yP){
        x = xP;
        y = yP;

    }

  public void setPosFactors(double y){

      factor = (int)Math.floor((y - upperLeftY + ((spaceBetweenStaffs -
4*lineSpace)/2))/spaceBetweenStaffs);
    //  System.out.println("in setPosFactors  factor:" +
Integer.toString(factor));
      startY = y  - upperLeftY - factor*spaceBetweenStaffs;
      posDouble = startY/halflineD;
      posFactor = (int)Math.rint(posDouble);
     //  System.out.println("in setPosFactors  posFactor:" +
Integer.toString(posFactor));
      finalY = posFactor*halflineD  + upperLeftY +
factor*spaceBetweenStaffs;
  }

   public String toString(){

      output += "<note name = '" + getName() + "' octave = '" + getOctave()
+ "'" + " stem = " + "'" + getStem() + "'" +
                " duration = '" + getDuration() + "' accidentals = '" +
                 getAccidentals() + "' x = '" + getX() + "' y = '" +
getFinalY() + "' />" + "\n";

      return output;
```

```
    }
}
```